NetSim:

A model oriented code generation service for simulating Wireless Sensor Networks

Julie Vlachou

November 2015

Master Thesis

School of Electronic & Computer Engineering

Techical University of Crete

Supervisor:

Asst. Prof. Vasilis Samoladas

Committee:

Asst. Prof. Vasilis Samoladas

Prof. Minos Garofalakis

Assoc. Prof. Antonios Deligiannakis

Abstract

In current Master thesis we present NetSim, a service for automatically generating simulations using the Castalia Network Simulator. NetSim was developed as part of an integrated Wireless Sensor Network design platform. We introduce the Network Simulation Descriptor (NSD) Model which integrates all the information necessary for defining, configuring and executing a simulation job, in a single model. The NSD model can be easily enhanced, manipulated and configured during design and runtime, causing the minimum programming effort as far as it concerns 1)the integration process with the rest platform tools & 2) the automated mapping process to the Castalia Network Simulator. NetSim is characterized by a model oriented programming philosophy, which abstracts new software specification definitions, their development process and their integration with the existing software, from great changes to the already developed software architecture and functionality. NetSim's Model driven programming design and philosophy offers great flexibility and customization abilities to the developed software, which makes further development of the existing software simple and straightforward.

Acknowledgment

I would like to thank my family and my beloved friends, for their support and patience during my Master study.

I would like to express my sincere gratitude to my advisor Prof. Vasilis Samoladas for the continuous support of my Master study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Furthermore he helped me to be confident in order to accept and handle any challenge in an academic and professional level. The knowledge that I gained through all these years of collaboration with my advisor, really helped me so as to evolve as a software engineer.

Julie

Contents

Chapter 1

Introduction

From the recent years of computer engineering data modeling was a dominant software engineering design technique, in all different kinds of the developed information systems. During the last years where web application services are flourished, their architecture and design becomes more mature and sophisticated. Due to the fact that recent web services need to manipulate and persist greater amount of data, data modeling techniques inevitably enter to the web services design, in order to create more efficient model oriented web services.

A data model organizes data elements and standardizes how the data elements relate to one another. Since data elements document real life people, the data model represents reality, computers are used for the accounting of these real life things and events and therefore the data model is a necessary standard to ensure exact human computer interaction.

Data modeling is a process used to define data models and analyze data requirements needed to support the business processes, meaning a collection of related, structured activities or tasks that produce a specific service or product (serve a particular goal), within the scope of corresponding information systems in organizations. Data modeling is a process of high importance during the design of a web service which determines the efficiency and the easiness of understanding and using from the end user perspective.

When data modeling is viewed from an application service context perspective, the discussion is about model oriented services. Model oriented services is the domain where the design and structure of the developed applications relies in the major data entities of a data model. The concept of the model oriented services is based to the idea that if the the actions are in a direct

relation with a specific data entity included in the model, is more easy for the user to understand and use the service, and for the system to manipulate and persist the entities. Model oriented services provide to the user more clear actions over the data, making the web service's notion easily to be understood and used. In model oriented services, all the actions are related with the entities included in the data model, and each action represents a CRUD action over a persisted entity included in the data model.

1.1 Definition of the problem

In the current work we present a model oriented code generation web service, for simulating Wireless Sensor Networks, called NetSim. NetSim was built for the needs of an overall Wireless Sensor Network design platform. We present NetSim's functionality and architecture, the data models used during the data collection and code generation phases, and the NetSim's model oriented concept.

1.2 Our approach

NetSim is a model oriented code generation web service for simulating Wireless Sensor Networks, developed for the needs of a greater Wireless Sensor Network design platform. This means that NetSim has to be built in a way which efficiently integrates its functionality with the rest platform. For this reason data models describing the NetSim tool's input and output data, have been wisely constructed during the NetSim architecture design process. These data models are transformed during NetSim runtime, so as to the initial data can be mapped to the intermediate data, necessary for NetSim execution runtime phases. The transformation and mapping process is straightforward in cases where the data values are related with the system runtime itself, or follows a model to model transformation process if we have more complex data contexts, where the data must be reassigned to other frameworks than the current (according to each execution phase), during the code generation process.

NetSim has two major components where each is consisted of multiple subcomponents. The main NetSim components are:

- **A model oriented web service:** Main purpose of this service is to collect and gather in a single entity model, the Network Simulation Descriptor (NSD) model, all the information necessary for a single simulation execution. The service's functionality has been designed and built in order to construct and manipulate NSD entities. The NSD entity comprises of data which are initialized and configured, via the NetSim's Tool Web Interface and via the rest platform's tools user interfaces. So additionally NetSim service provides interoperability with the overall platform's tools and modules in an NSD model oriented concept.

- **An execution manager:** For the needs of WSN simulation process the Castalia network simulator has been selected, so as to avoid building a new WSN simulator from the beginning. Main purpose of the execution manager is to start, inspect and complete the WSN simulation executions. So the execution manager is responsible for transforming the initial NSD model to the target Castalia model, and backwards (the reverse flow is necessary for the Castalia output data interpretation and display process). The execution manager consists the back-end of the NetSim web service, and is responsible of performing all the model transformation processes necessary for a complete simulation execution.

Chapter 2

Simulation Tools for Wireless Sensor Network

In this chapter existing simulation tools for Wireless Sensor Networks (WSN's) will be listed and presented, the basic simulation concepts of the common WSN simulators will be covered, and finally the simulation framework Omnet++ and the Castalia simulator in top of which NetSim is built, will be extensively presented.

2.1 A general model for Wireless Sensor Networks Simulations

The development of a WSN simulation tool, includes the definition of a model which describes the WSN simulation framework. Every simulation framework must include components for each different tier of the simulated WSN. In this section we will describe a general component model for WSN simulators, as it is presented in [19] , derived from [30, 34], which describes the general component model followed from the most of the WSN simulation tools. All the WSN simulation tools include the components described below, despite the fact that the WSN modelling perspective may vary for each different simulation tool. In the proposed WSN simulation decription framework are introduced two main model components, the network and the node model, where each consists of an object set which describe all the parameters necessary for simulating each different tier and aspect of a WSN.

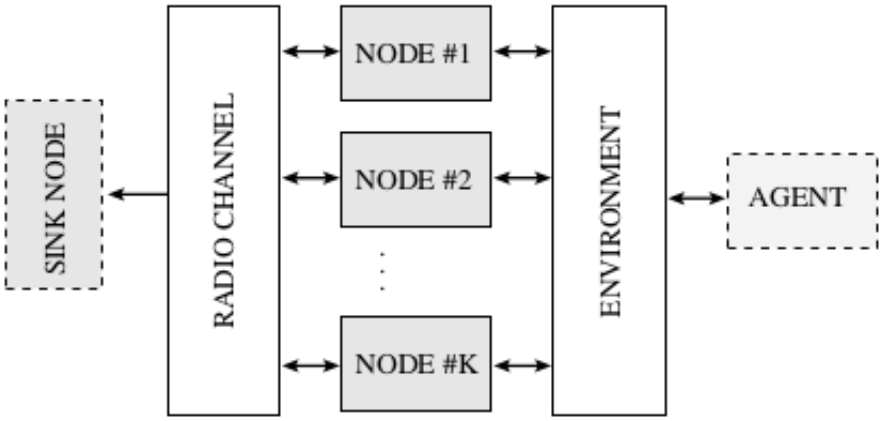

Figure 2.1: *The Network Model components.*

2.1.1 The Network Model

This model include all the parameters that are common or commonly cofigured at Network level. The main components of the Network model are:

- Nodes: Each node is a physical device monitoring a set of physical variables. Nodes communicate with each other via a common radio channel. Internally, a protocol stack controls communications. Unlike classical network models, sensor modes include a second group of components: The physical node tier, which is connected to the environment. Nodes are usually positioned in a two or three dimensional world. An additional "topology" component, may control node coordinates. Depending on the application and deployment scenario, a WSN can contain from a few to several thousands of nodes.

- Environment: The main difference between classical and WSN models is the additional "environment" component. This component models the generation and propagation of events that are sensed by the nodes, and trigger sensor actions, i.e. communication among nodes in the network. The events of interest are generally a physical magnitude such as sound, temperature, motion etc.

- Radio channel: It characterizes the propagation of radio signals among the nodes in the network. Very detailed models use a "terrain" component, connected to the environment and radio channel components. The terrain component is taken into consideration to

compute the propagation as part of the radio channel, and also influences the physical magnitude.

- Sink nodes: These are special nodes that, if present, receive data from the net, and process it. They may interrogate sensors about an event of interest. The use of sinks depends on the application and the tests performed by the simulator.

- Agents: A generator of events of interest for the nodes. The agent may cause a variation in a physical magnitude, which propagates through the environment and stimulates the sensor. This component is useful when its behavior can be implemented independently from the environment, e.g., a mobile vehicle. Otherwise, the environment itself can generate events.

2.1.2 The Node Model

The Node Model refers to node behavior, which mainly depends on interacting factors that cause cross-layer interdependencies and differ from node to node during node operation simulation. Thus this factors needs to be classified and configured in order to be taken into consideration during the node behavior simulation process. A convenient way to describe these factor is to divide a node into the following abstract tiers, which can be considered as the Node Model main components:

- The Protocol-tier comprises all the communication protocols. Typically, three layers coexist at this tier: A MAC layer, a routing layer and a specific application layer. Note that the operation of the protocol tier usually depends on the state of the physical tier described below, e.g. a routing layer can consider battery constraints to decide on packet route. Hence, an efficient method to interchange tier information must be developed.

- The physical-node tier represents the hardware platform and its effects on the performance of the equipment. Actual composition of this tier may change depending on the specific application. The common elements of this tier are the set of physical sensors, the energy module and the mobility module. Physical Sensors describe the behavior of the

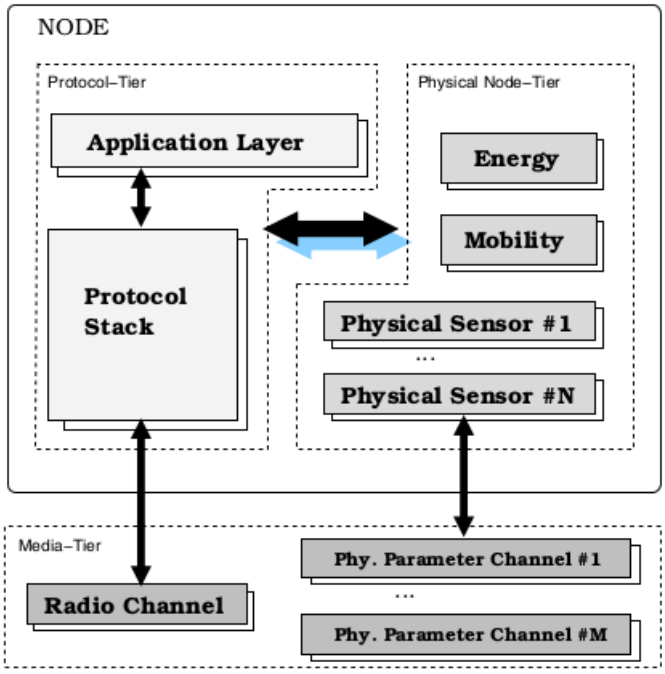

Figure 2.2: *The Node Model components.*

monitoring hardware. Energy module simulates power consumption in the component hardware, a critical issue in WSN evaluation. Mobility module controls sensor position.

- The media-tier is the link of the node with the "real world". A node is connected with the environment through: (1) A radio channel, and (2) through one or more physical channels. Physical channels receive environmental events as described in section.

2.2 Basic simulation concepts

There are two basic simulation concepts widely used in the WSN simulation tools: Trace-Driven Simulation and Discrete-Event Simulation [25].

Discrete Event Simulation: [25] is widely used in WSNs, because it can easily simulate lots of jobs running on different sensor nodes. Discrete-event simulation includes some of components. This simulation can list pending events, which can be simulated by routines. The global variables, which describe the system state, can represent the simulation time, which allow the scheduler to predict this time in advance. This simulation includes input routines, output rou-

tines, initial routines, and trace routines. In addition, this simulation provides dynamic memory management, which can add new entities and drop old entities in the model. Debugger breakpoints are usually provided in most of the discrete-event simulators, thus users can check the code step by step without disrupting the program operation.

Trace-Driven Simulation:[25] provides different services. This kind of simulation is commonly used in real system. The simulation results have more credibility. It provides more accurate workload; these detail information allow users to deeply study the simulation model. Usually, input values in this simulation are constants therefore unchanged. However, this simulation also contains some drawbacks. For example, the high-level detail information increases the complexity of the simulation, workloads may change, and thus the representativeness of the simulation needs to be suspicious.

2.3 Wireless Sensor Network Simulators

This section illustrates the main-stream simulation tools widely used for simulating WSNs and analyzes the advantages and disadvantages of each simulation tool.

2.3.1 TOSSIM TinyOS

TOSSIM [23, 6, 19, 32, 28, 33, 36] is a simulator specifically designed for WSN running on TinyOS [17], which is an open source operating system targeting embedded operating system. In 2003, TOSSIM was first developed by UC Berkeley's TinyOS project team. TOSSIM is a bit-level discrete event network simulator built in Python and C++. People can run TOSSIM on Linux Operating Systems or on Cygwin on Windows. TOSSIM also provides open sources and online documents.

TOSSIM has a GUI, TinyViz, which is very convenient for the user to interact with electronic devices because it provides images instead of text commands. In addition, TOSSIM is a very simple but powerful simulator for WSN. Each node can be evaluated under perfect transmission conditions, and using this simulator can capture the hidden terminal problems. As a specific network simulator, TOSSIM can support thousands of nodes simulation. This is a very good feature, because it can more accurately simulate the real world situation. Besides network, TOSSIM

can simulate radio models and code executions, where the 2nd technique provides more precise simulation results at component level because of compiling directly to native codes.

However, this simulator still has some limitations. Firstly, TOSSIM is designed to simulate behaviours and applications of TinyOS, and it is not designed to simulate the performance metrics of other new protocols. Therefore, TOSSIM can not correctly simulate issues of the energy consumption in WSN; people can use PowerTOSSIM [16], another TinyOS simulator extending the power model to TOSSIM, to estimate the power consumption of each node. Secondly, every node has to run on NesC code, a programming language that is event-driven, component-based and implemented on TinyOS, thus TOSSIM can only simulate the type of homogeneous applications. Thirdly, because TOSSIM is specifically designed for WSN simulation, motes-like nodes are the only thing that TOSSIM can simulate. In sum, TOSSIM as simulator of WSN has great capabilities which can be fully exploited only using the TinyOS framework.

2.3.2 Cooja

Cooja [9, 35] is a simulator of another operating system – Contiki [8]. The applications for Contiki are written in C and the simulations are written in Java. The simulator Cooja is similar to TOSSIM, in such a way that its main purpose is to simulate the behaviour of an operating system. Each node in the simulated network can be different not only concerning its installed software but also the hardware platform may vary. Cooja is a flexible simulator and many parts may be replaced or extended [29]. On the other hand, some crucial functions (e.g. radio models) are still waiting for the extension. The authors of Cooja claim in [29] that their simulator can work on different levels that it enables the so-called cross level simulations [29], and that is principally a simulator designed for network and application levels without taking the hardware properties into its account, while TOSSIM (operating system level) is intended particularly for simulating the behaviour of the operating system TinyOS. Cooja provides simulations on the below levels:

- Networking: This operation level concerns mainly the network routing process. Radio propagation, radio devices and the wireless channel model are the most important parts of this level. The users of Cooja may develop and modify certain modules. The sensor nodes provided by Cooja, can be replaced by abstract Java implementations without con-

fronting any compatibility issues with the operating system Contiki. Also Cooja provides the ability to simulate heterogeneous networks, where a set of nodes can be replaced by real motes.

- Operating System: The aim of this level is to simulate Contiki by executing native operating system code. This can be useful especially for the developers of Contiki to allow testing and evaluation of changes in Contiki libraries.

- Machine code instruction set: Nodes having different underlying structure may be simulated using Java-based microcontroller simulator instead of a compiled Contiki system.

2.3.3 Ns-2 & Ns-3

Ns-2[13] is a famous and known simulator for network traffic simulations, is free software, publicly available and licensed for use under version 2 of the GNU General Public License. Despite the fact that Ns-2 is not actively maintained, we will present its features due to the great simulation capabilities for WSN which the simulator offers. It is a discrete event simulator targeted at networking research. Ns-2 provides substantial support for simulation of TCP (Transmission Control Protocol), routing, and multicast protocols over wired and wireless (local and satellite) networks [12, 35]. Ns-2 was first intended for wired networks but nowadays it also offers features for wireless networks like wireless channel model, energy model and traffic and movement patterns. However, traffic and mobility are typically produced beforehand and they are not an integral part of Ns-2 architecture [27]. Another Ns-2 plus is that a large number of external protocols are already implemented.

In the Ns-2 simulator all the nodes consist of a link layer (LL), an Address Resolution Protocol (ARP) module (to resolve the Internet Protocol (IP) address to the MAC address) which is connected to the link layer, an interface priority queue (IFq), a MAC layer and a network interface (netIF) which is connected to the channel [12]. Also Ns-2 provides the ability to simulate mobile nodes, such nodes can be “moved” around the network terrain and simulate movements of an original network device.

Ns-2 implements five different routing protocols for ad-hoc wireless network – DSDV (Destination-Sequenced Distance Vector), DSR (Dynamic Source Routing), TORA (Tempo- rally-Ordered Rout-

ing Algorithm), AODV (Ad hoc On-Demand Distance Vector) and PUMA (Protocol for Unified Multicasting Through Announcements). Also Ns-2 implements three different radio propagation models, so as to predict in a more realistic way the received signal power of each packet.

Finally Ns-2, provides a very detailed Energy model ideal for energy consumption oriented simulations. The Energy model operates as follows: The level of energy of a mobile node is represented with a special attribute which is initialized with some value. Then the level of energy is being decreased for each every transmitted and received packet by some given values. When the energy level at the node goes down to the number of zero (complete depletion), no more packets can be received or transmitted by the node [12]. This can be used for experiments and a comparison of the energy efficiency of different protocols.

Ns-3[14] is the latest version of Ns-2, despite that, during the development process of Ns-3 it was decided to completely abandon backward-compatibility with Ns-2. The new simulator was written from scratch, using the C++ and Python languages. The general process of creating a simulation using the Ns-3 can be divided into several steps:

1. Topology definition: to ease the creation of basic facilities and define their interrelationships, Ns-3 has a system of containers and helpers that facilitates this process.

2. Model development: models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.

3. Node and link configuration: models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.

4. Execution: simulation facilities generate events, data requested by the user is logged.

5. Performance analysis: after the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.

6. Graphical Visualization: raw or processed data collected in a simulation can be graphed using tools like Gnuplot, matplotlib or XGRAPH.

Ns-3 is often criticized for its lack of support for protocols (like WSN, MANET etc.) which were supported in ns-2, as well as for the lack of backward compatibility with ns-2. Finally both Ns-2 and Ns-3 do not offer a GUI so are considered to be time consuming during learn and design in comparison to GUI-based simulators.

2.3.4 JSim

J-Sim [4, 34] is a discrete event network simulator built in Java. This simulator provides GUI library, which facilities users to model or compile the Mathematical Modelling Language, a "text-based language" written to J-Sim models. J-Sim provides open source models and online documents. This simulator is commonly used in physiology and biomedicine areas, but it also can be used in WSN simulation. In addition, J-Sim can simulate real-time processes.

As far as it concerns WSN simulation, the corresponding models provided by J-Sim have good reusability and interchangeability, fact which facilitates easily the simulation process. J-Sim also contains large number of protocols, this simulator can also support data diffusions, routings and localization simulations in WSNs by detailed models and implemented protocols. J-Sim can simulate radio channels and power consumptions in WSNs. A J-Sim aspect which makes the simulator user friendly is the GUI library, which can help users to trace and debug programs. The independent platform is easy for users to choose specific components to solve the individual problem. However, this simulator has some limitations. Comparing with NS-2, J-Sim can simulate relatively larger number of sensor nodes, around 500, and J-Sim can save lots of memory size. The execution time is much longer than most of the WSN simulator. Because J-Sim was not originally designed to simulate WSNs, the inherently design of J-Sim makes users hardly add new protocols or node components.

2.3.5 EmStar

EmStar [23, 20, 22, 19] is a simulator specifically designed for WSN built in C, and it was first developed by University of California, Los Angeles. EmStar is a trace-driven simulator [22] running in real-time. People can run this simulator on Linux operating system. This simulator supports to develop WSN application on better hardware sensors. Besides libraries, tools and services, an

extension of Linux microkernel is included in EmStar simulator.

EmStar has a modular programming model, which allows users to run each module separately without sacrificing the reusability of the software. EmStar has a robustness feature that it can mitigate faults among the sensors, and it provides many modes during debugging. There is a flexible environment in EmStar that users can freely change between deployment and simulation among sensors. Also with a of standard interfaces, each service can easily be interconnected. EmStar has a GUI, which is very helpful for users to control electronic devices. Also in EmStar every execution platform is defined by the common source code files, fact which decrease bugs when iterate the separate modes. In addition, EmStar provides online documentation. However, EmStar has some drawbacks. EmStar does not support large number of sensors during simulations, therefore the limited scalability may decrease the reliability of simulation results. In addition, EmStar can only run real time simulations. Moreover, this simulator can only apply to iPAQ-class sensor nodes and MICA2 [11] motes. All these drawbacks limit the use of this simulator.

2.3.6 ATEMU

ATEMU [1, 24, 19] is a simulator of an AVR processor for WSN built in C, is open source and provides online documents. AVR is a single chip microcontroller commonly used in the MICA platform. ATEMU provides a GUI, called Xatdb, people can use this GUI to run codes on sensor nodes, debug codes and monitor program executions. People can run ATEMU on Solaris and Linux operating system. ATEMU is a specific simulator for WSNs, and can support running of TinyOS on MICA2 hardware. ATEMU can simulate not only the communication among the sensors, but also every instruction implemented in each sensor.

ATEMU can simulate multiple sensor nodes at the same time, and each sensor node can run different programs. Also ATEMU has a large library of a wide rage of hardware devices. Additionally ATEMU can provide a very high level of detail emulation in WSNs. For example, it can simulate different sensor nodes in homogeneous networks or heterogeneous networks. ATEMU can simulate different application run on MICA. Also users can simulate power consumptions or radio channels by ATEMU. The provided GUI can help users debug programs, and monitor program executions. ATEMU can provide an accurate model, which helps users

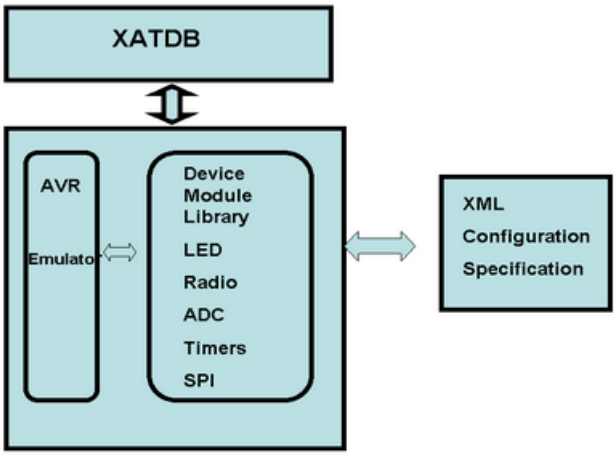

Figure 2.3: *ATEMU component architecture.*

to give unbiased comparisons and get more realistic results. The ATEMU components architecture is shown in schema below. However, this simulator also has some limitations. For instance, although ATEMU can give a highly accuracy results, the simulation time is much longer than other simulation tools. In addition, ATEMU has fewer functions to simulate routing and clustering problems.

2.3.7 Avrora

Avrora [2, 32, 33] is a simulator specifically designed for WSNs built in Java. Similar to ATEMU, Avrora can also simulate AVR-based microcontroller MICA2 sensor nodes. This simulator was developed by University of California, Los Angeles Compilers Group, is open source and provides online documentation. Avrora provides a wide range of tools that can be used in simulating WSNs. This simulator combines the merits of TOSSIM and ATEMU, and limits their drawbacks. Although Avrora does not provide GUI. Avrora also supports energy consumption simulation. However, this simulator has some drawbacks, it does not have any GUI and, can not simulate network management algorithms because it does not provide network communication tools.

Avrora is an instruction-level simulator, which removes the gap between TOSSIM and ATEMU. The codes in Avrora run instruction by instruction, which provides high speed and better scal-

ability. Avrora can support thousands of nodes simulation, and can save much more execution time with similar accuracy. Avrora provides larger scalability than ATEMU does, with equivalent accuracy. Also Avrora provides more accuracy than TOSSIM, with equivalent scales of sensor nodes. Unlike TOSSIM and ATEMU, Avrora is built in Java language, which provides much flexibility. Finally Avrora can simulate different programming code projects, but TOSSIM can only support TinyOS simulation.

2.3.8 MiXiM

MiXiM[5, 35] is an OMNeT++ modelling framework created for mobile and fixed wireless networks (wireless sensor networks, body area networks, ad-hoc networks, vehicular networks, etc.). It offers detailed models of radio wave propagation, interference estimation, radio transceiver power consumption and wireless MAC protocols. It is a merger of several OMNeT++ frameworks written to support mobile and wireless simulations [5] and thus its name comes from "mixed simulator". The general structure, connection management and mobility support is taken from Mobility Framework (MF), the radio propagation model is taken from Channel Simulator (ChSim) and the protocol library comes from the MAC simulator, the Postif Framework and from the Mobility Framework [26].

MiXim has a hierarchical modular structure, so it is easily configured by the user. In general it provides a manager class for each module where each module represents a different layer of WSN design and structure. Multiple managers can be employed to enable different frequency ranges such as radio waves or ultra sound. Different kinds of nodes such as APs or terminals can be represented by nodes module [26]. In MiXim three modules are used to represent standard network layers: the application layer, the network layer, and the MAC layer which is actually grouped together with the physical layer into a Network Interface Card (NIC) module. A single node can have several NICs to model, for example, a laptop which contains Bluetooth and IEEE 802.11 (Wi-Fi). In MiXim the connectivity of the wireless devices is a hard task because the communication channel is the air which is actually a broadcast medium. Therefore the connectivity is handled similarly to the implementation of Castalia – nodes are connected only when they are within maximal interference distance. The interference distance is usually higher then the distance necessary for flawless communication. As far as it concerns routing there are sev-

eral routing paradigms that can be used within MiXiM (e.g. source-to-sink, any-to-any, local neighbourhood) [26]. The routes are specified in a configuration file. Finally in MiXim battery capacity and energy consuming operations are modelled by the Energy Framework [21] which was incorporated into the Mobility Framework of MiXiM as well.

Table 2.1: Comparison of existing of WSN Simulators

Simulator	Simulation Concept	GUI	General or WSN simulator	Open Source
TOSSIM	Discrete-Event-Simulator	Yes	WSN simulator	Yes
Cooja	Discrete-Event-Simulator	Yes	WSN simulator	Yes
Ns-2	Discrete-Event-Simulator	No	General simulator	Yes
JSim	Discrete-Event-Simulator	Yes	General simulator	Yes
EmStar	Trace-Driven-Simulator	Yes	WSN simulator	Yes
Avrora	Discrete-Event-Simulator	Yes	WSN simulator	Yes
MiXim	Discrete-Event-Simulator	Yes	General simulator	Yes

2.4 OMNeT++ Simulation framework

OMNeT++ [15, 37] is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects. OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools. There are extensions for real-time simulation, network emulation, database integration, SystemC integration, and several other functions.

Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes or free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

The basic OMNeT++ components are:

- simulation kernel library
- NED topology description language
- OMNeT++ IDE based on the Eclipse platform
- GUI for simulation execution, links into simulation executable (Tkenv)
- command-line user interface for simulation execution (Cmdenv)
- utilities (makefile creation tool, etc.)
- documentation, sample simulations, etc.

OMNeT++ supports multiple simulation models ideal for many different simulation environments from peer-to-peer networks, wired and wirelless communication protocols, road traffic networks to complete operation and function of Wireless Sensor Networks. For the needs of the present work the Castalia Wireless Sensor Network simulation model has been used.

2.5 Castalia Network Simulator

Castalia [7, 31] is a simulator for Wireless Sensor Networks (WSN), Body Area Networks (BAN) and generally networks of low-power embedded devices. It is based on the OMNeT++ platform and can be used by researchers and developers who want to test their distributed algorithms and/or protocols in realistic wireless channel and radio models, with a realistic node behaviour especially relating to access of the radio. Castalia can also be used to evaluate different platform characteristics for specific applications, since it is highly parametric, and can simulate a wide range of platforms. The main features of Castalia based in [18] are:

- **Advanced channel model based on empirically measured data** :
  - Model defines a map of path loss, not simply connections between nodes
  - Complex model for temporal variation of path loss
  - Fully supports mobility of the nodes
  - Interference is handled as received signal strength, not as separate feature
- **Advanced radio model based on real radios for low-power communication:**
  - Probability of reception based on SINR, packet size, modulation type. PSK FSK supported, custom modulation allowed by defining SNR-BER curve
  - Multiple TX power levels with individual node variations allowed
  - States with different power consumption and delays switching between them
  - Flexible carrier sensing (polling-based and interrupt-based)
- **Extended sensing modelling provisions:**
  - Highly flexible physical process model
  - Sensing device noise, bias, and power consumption
  - Node clock drift, CPU power consumption.
  - MAC and routing protocols available
  - Designed for adaptation and expansion.

Castalia was designed right from the beginning so that the users can easily implement/import their algorithms and protocols into Castalia while making use of the features the simulator is providing. Proper modularization and a configurable, automated build procedure help towards this end. The modularity, reliability, and speed of Castalia is partly enabled by OMNeT++, an excellent framework to build event-driven simulators.

Castalia Network Simulator structure

Castalia is using OMNeT++ structure concept as a base. OMNeT's basic concepts are modules and messages. A simple module is the basic unit of execution. It accepts messages from

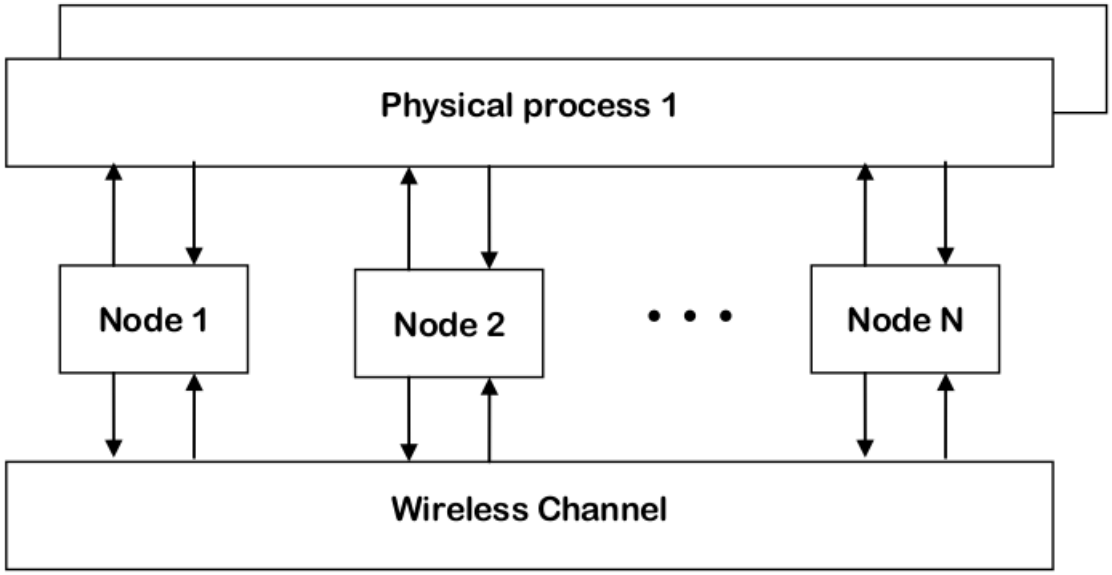

Figure 2.4: *The modules and their connection in Castalia.*

other modules or itself, and according to the message, it executes a piece of code. The code can keep state that is altered when messages are received and can send (or schedule) new messages. There are also composite modules. A composite module is just a construction of simple and/or other composite modules. Castalia's basic module structure is shown in the diagram 2.1.

Notice that the nodes do not connect to each other directly but through the wireless channel module(s). The arrows signify message passing from one module to another. When a node has a packet to send this goes to the wireless channel which then decides which nodes should receive the packet. The nodes are also linked through the physical processes that they monitor. For every physical process there is one module which holds the "truth" on the quantity the physical process is representing. The nodes sample the physical process in space and time (by sending a message to the corresponding module) to get their sensor readings.

There can be multiple physical processes, representing the multiple sensing devices (multiple sensing modalities) that a node has. The node module is a composite one. Figure 2 shows the internal structure of the node composite module. The solid arrows signify message passing and the dashed arrows signify simple function calling. For instance, most of the modules call a function of the resource manager to signal that energy has been consumed. The Application

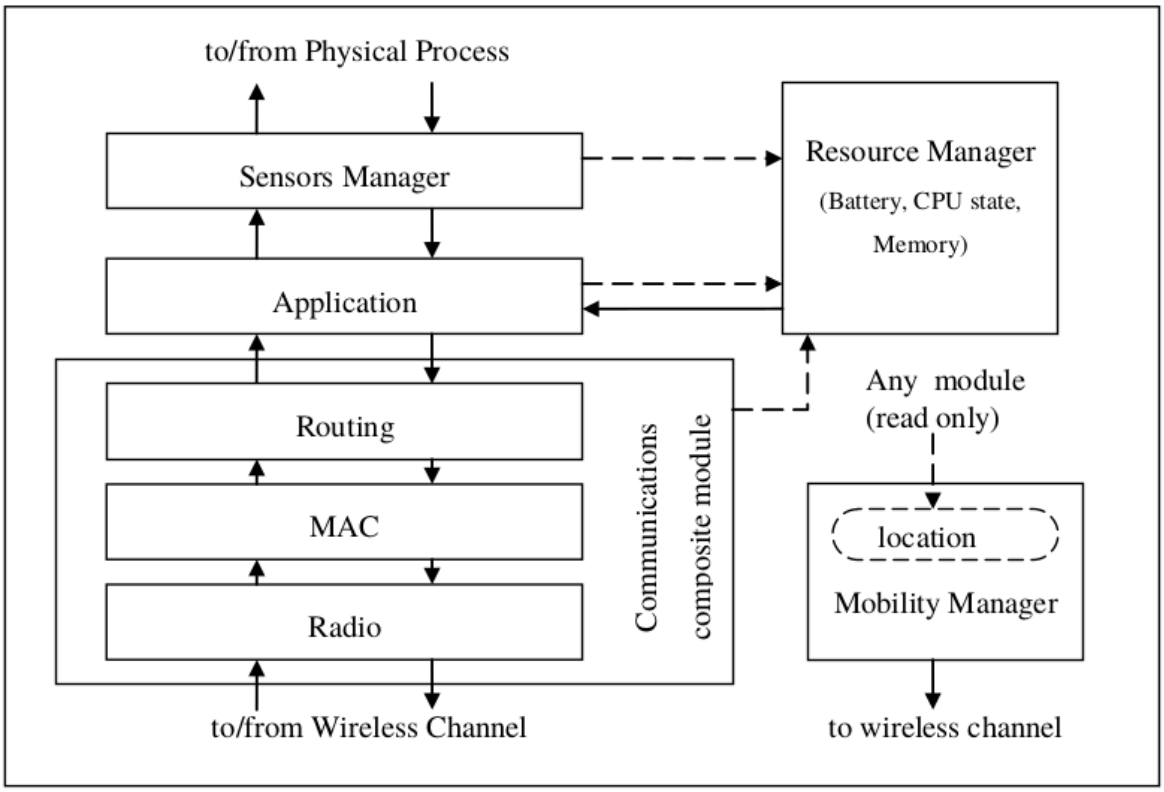

Figure 2.5: *The node composite module in Castalia.*

module is the one that the user will most commonly change, usually by creating a new module to implement a. new algorithm. The communications MAC and Routing modules, as well as the Mobility Manager module, are also good candidates for change by the user, again usually by creating a new module to implement a new protocol or mobility pattern. Castalia offers support for building your own protocols, or applications by defining appropriate abstract classes (for greater details please advise Castalia User's Manual). All existing modules are highly tuneable by many parameters.

Castalia has a modular structure with many interconnecting modules as depicted in the diagrams 2.1 and 2.1. ach one of the modules has one or more parameters that affect its behavior. An OMNeT++ NED language NED file (file with extension .ned in the Castalia source code) defines the basic structure of a module by defining its input/output gates and its parameters, and also define default values for the parameters. . With this language we can also define possible submodule structure (if this is a composite module).

The Castalia structure is also reflected in the hierarchy of directories in the source code. Ev-

ery module corresponds to a directory which always contains a .ned file that defines the module. If the module is composite then there are subdirectories to define the submodules. If it is a simple module then there is C++ code (.cc, .h files) to define its behaviour. This complete hierarchy of .ned files defines the overall structure of the Castalia simulator. Normally the user will not alter these files. Nevertheless, these files are dynamically loaded and processed (using a feature of OMNeT) so that any change does not require the recompilation of Castalia (unless of course new simple modules with new functionality appear).

Finally Castalia has a configuration file (usually named OMNeTpp.ini and residing in the Simulations dir tree), via the configuration file the user can assign values to parameters, or just reassign them to a different value from their default one. This way the user can build a great variety of simulation scenarios according to the user's needs.

Chapter 3

WSN-DPCM architecture and tools

As it was mentioned in introduction NetSim was built for the needs of an integrated WSN design platform called WSN-DPCM. The platform was named after the initial of the main platform tools: **The Development Tool, the Planning Tool, the Commissioning&Maintenance Tool**. In order to understand the objectives and the role of NetSim Simulator in the WSN-DPCM platform, the overall platform architecture will be thoroughly described in this chapter. The main tools that comprise the platform will be presented, and furthermore each tool's objectives, architecture and functionality will be discussed respectively.

3.1 The WSN-DPCM Concept and Objectives

The objective of WSN-DPCM is to develop a full platform to address the main Wireless Sensor Network (WSN) challenges for smart environments that include the middleware for heterogeneous wireless technologies and an integrated engineering tool-set for Development, Planning, Commissioning, and Maintenance activities for expert and non-expert users. The main goal of this project is to provide tools to support the work of stakeholders involved in WSN applications.

3.1.1 Development cycle of a WSN application

The development cycle of an application consists of certain phases. These phases reflect the different concerns addressed during the development cycle, and are carried out by individuals

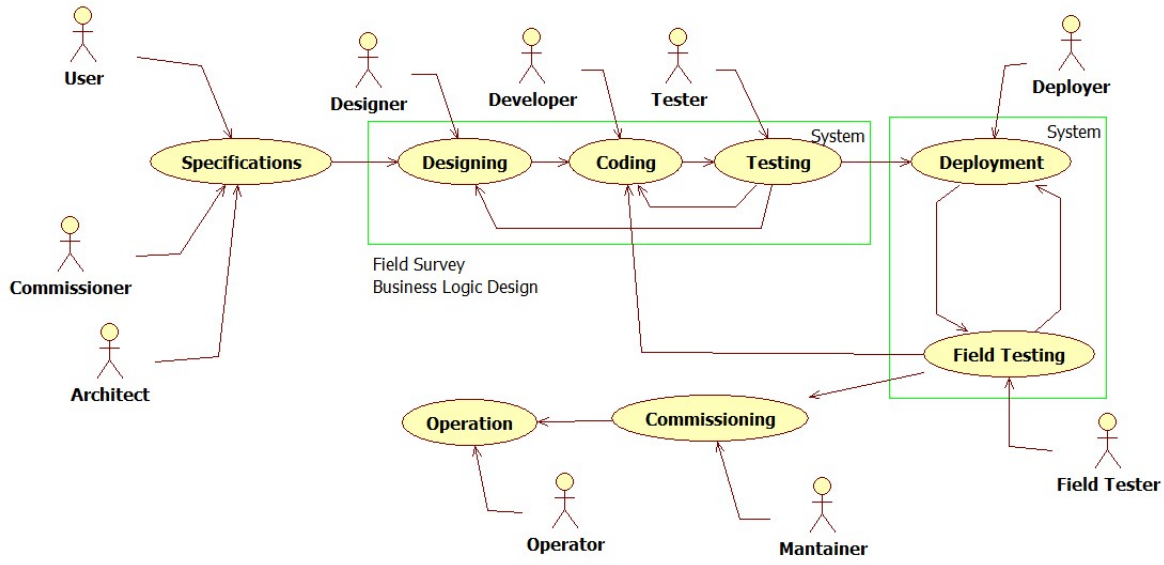

Figure 3.1: WSN-DPCM Platform Development Flow.

acting under certain roles. The main phases in the development cycle of a WSN application are shown in *Figure 1.1*.

Comment, it would be nice to be a schema that corresponds the above tasks to specific tools

Specification phase: The main requirements of the application are developed during this phase. These are formulated between the Commissioner of the application (i.e. the person who requests the development of the application) and the application Architect. The outcome of this phase is the set of functional, operational and cost-related requirements. These requirements are then used to drive the subsequent stages of the development. **Main tasks in this phase are: Define the main network components and requirements of the WSN network.**

Designing phase: There are two main tasks during this phase. The first task is field survey, which relates to activities involving the physical sites of sensor deployment. These activities include surveying the terrain, planning the location of installation for individual sensors, and assessing the RF features of the installation site. Business logic design is the detailed specification of the application's business logic. This design includes issues related to the choice of hardware platforms (motes, base and servers) and software components (drivers and external libraries as required by the application), network topology, timing and synchronization, issues

related to sensor operation (sampling-related) and in-network processing (compression, aggregation, etc.).

Coding phase: During this phase the actual coding of the application is performed. The topology development and the integration of the desired functionality of the network components, WSN nodes and sinks, must be implemented according to the requirements and standards defined in the designing phase. Simulation test benches also need to be implemented, simulating measurement collection and testing the ranges of the configurable network parameters. The simulation process of the total network operation should extend to a sufficient time span and several iterations should be performed, in order to extend the testing to infrequent events. NetSim serves the simulation needs of the platform.

Testing phase: This phase involves the development of testing infrastructure and performing tests of the developed system. This phase coexists in time with the coding phase and there is a strong exchange of information. In this phase, the data collected during the simulation is evaluated, so as to perform the necessary optimizations in the overall network's behaviour through the readjustment of the individual network characteristics. There is a consecutive interactive feedback between testing-designing-simulate phases until the initial requirements will be fulfilled. After the completion of testing phase it should be possible to estimate the main network characteristics and upper limits such as number of nodes, the node hardware and functional characteristics, the network maintenance cost, parameters which concern the communication protocol among WSN nodes, mean expected WSN node lifetime, WSN channel latency etc.

Deployment phase: During this phase the physical components of the application (motes, base nodes etc.) are installed at the installation site and the developed application software is initially deployed on the platform. The installation process is based to the conclusion extracted from the former designing, coding and testing phases. During this phase the real set-up of the WSN takes place, we define all the node configuration parameters based on t he evaluation and conclusion extracted from the simulation process. Finally we adjust the actual WSN node positions according to the real environmental conditions of the actual field and transfer the simulated WSN data to deployment tools.

Field testing phase: This phase involves testing related to the physical components of the application, with the application deployed on these components. This phase may occur concur-

rently with the deployment phase in order to make node location readjustment for optimal performance and communication between nodes. This is the time to address possible limitations emerging from environmental conditions. The WSN is configured for run-time auto-testing, and numerous test benches are preformed, possibly even before the full network is deployed.

Commissioning phase: The commissioning phase involves the final installation of the completed application on the physical components and the initiation of their functionality. We define the functionality, the monitoring and data processing abilities of the server application.

Operation phase: During this phase the application is in regular use. Main activities during this phase (apart from activities related to the application logic) include maintenance and repair in order to satisfy all the specified requirements which were thoroughly defined, tested and examined in the previous phases.

3.2 WSN-DPCM Tools

In this section each WSN-DPCM platform's tool functionality and purpose will be presented and described respectively.

3.2.1 The Planning Tool

In this section the structure and functionality of the Planning Tool (PT) is described. It must be noted that the Planning Tool is described in greater detail than the rest platform tools, due to the fact that is the only tool that communicates with the NetSim Simulator except from the Project Repository. Furthermore all the information related with network topology, mote types and characteristics defined by the user, is stored in the Project Repository from the Planning Tool and further retrieved from NetSim in order to start the network simulation process. Also the Planning Tool Web Interface is the first phase for creating a WSN application so all major entities that represent the platform design and commissioning concept are introduced here.

Planning Tool Purpose

The planning tool purpose is to assist the WSN designer during all activities involved in the network deployment in order to avoid long lasting deployments by minimizing trial-and-error

approach while also reducing the number of nodes. The WSN planning activities are essentially linked to determining the suitability of proposed point-to-point communication links and also the coverage area of a proposed mote. The former is concerned with providing channels of communications between sites together with any obstructions or clutter and atmospheric conditions, which may cause interference. Furthermore, the planning tool provides an electromagnetic field propagation simulator that can simulate accurately the real situation of communication between two or more motes and this capability avoids any assumption from part of the WSN designer on this aspect during your activities of planning. The main feature of the planning tool by which the user saves effort and time during the WSN deployment phase is the visualization capability. Via Planning Tool are visualized:

- Landscape Maps
- Mote location on the maps
- WSN simulation data geographically
- RF simulation data geographically

Moreover, the Planning tool is integrated with Development and Commissioning & Monitoring tools in order to provide the data flow between the tools in a transparent way for the end-user.

Basic Functionality

The Planning tool functionality integrates:

- Visualization and browse of spatial maps to identify the geographic area of interest for the WSN deployment. The spatial maps are displayed by a map viewer within the web browser.
- Provide tools to define the WSN Plan: configuration, insert/move/delete motes, etc. These tools define a WSN plan selecting the mote type (e.g. sink, gateway) and providing the configuration for them. Moreover the mote position is defined by a click on the displayed map and the mode can be changed or deleted always selecting the mote by a click on the screen.

- Provide the RF simulator in order to simulate the RF propagation for a defined WSN plan and the motes in the simulated plan can be updated in order to obtain the optimal RF propagation signal.
- Display of the WSN simulation results in order to evaluate the performance of the WSN application under develops.
- Display of the network graph of the WSN plan in order to provide a graphical view of the mote type distribution within the WSN under definition.
- Display the node on the map in order to show a WSN plan on the appropriate 2D or 3D maps. The user can view if any mote is closed to any obstacle or it could be located in a place where the mote deployment on the field will be more ease.
- Save and restore of a WSN plan in order to complete or refine the WSN plan in a next time without to redo always a lot of work.

Entities

The DPCM platform manages different data entities and these entities are read and/or wrote through the DPCM tools and middleware. The DPCM platform manages mainly the following data entities: Project, Planning, Mote and Users. These data are important for the whole platform because these are transversal to all tools and middleware.

The Project entity is the common entity between the DPCM tools and the Middleware. The project definition is the starting point to use the DPCM platform during the development of a WSN application. The defined project provides the user WSN requirements by features and constraints of the WSN application will be developed with the DPCM platform. A specific project is created providing the following items: Project name, WSN application purpose, maximum distance between motes, sampling frequency, number of motes, sensing precision, parameters environmental has to be sensed and others important and specific WSN application constraints.

The User entity represents somebody that uses the DPCM platform to perform specific activities involved during the development of a WSN application. Each registered user in the DPCM system will have assigned a set of the privileges to access the DPCM platform function-

alities. The user registered within of the DPCM platform will access to the DPCM platform's tools in order to use all offered functionalities to design your WSN application. The role defines what and user could do within of the platform. The group defines a group of users which are assigned a set well defined of roles The registered user could create and manage your projects, if your DPCM privileges will enable these kinds of project functionalities. Moreover, someone user with appropriate privileges could manage projects of others users.

The planning entity defines the motes position to be deployed on the field. Each created Project will have at a certain point the relate planning. The registered user defines a WSN planning providing a triple (x, y, z) of coordinates and high from ground for each mote that will arrange the WSN, which will be deployed. A project can be created without to define the relate planning. The planning can be defined in a second time. Furthermore, the planning definition is independent of the mote development activities. Finally a Project can be related with multiple Plannings.

The Mote entity is composed of different components and each component is defined providing specific component's parameters value. A mote could be already available within the mote library. The Mote class represents the wireless nodes physically deployed in the environment.The user could design the planning before that the motes develop.

The RF simulation entity with a specific simulation parameters values can be run for a given Planning. The RF simulation can be iterated changing the simulation parameters values as long as the user target will be achieved. The RF simulation results are stored in the platform linked to the planning and RF simulation configuration which had provided them.

Network design flow

The Planning Tool provides a WebTop interface by which the user can create an account, create new projects related with the overall design of a WSN Network, modify its own projects and run RF Simulations.

After the user login, the platform GUI is showed with the only authorized functionalities enabled. Considering that the logged user has the privileges to access the project manager GUI, can create a new project or open a saved project. The user can open/create a project in order to be able to work with the PT's functionalities. The project is the base entity in the DPCM platform

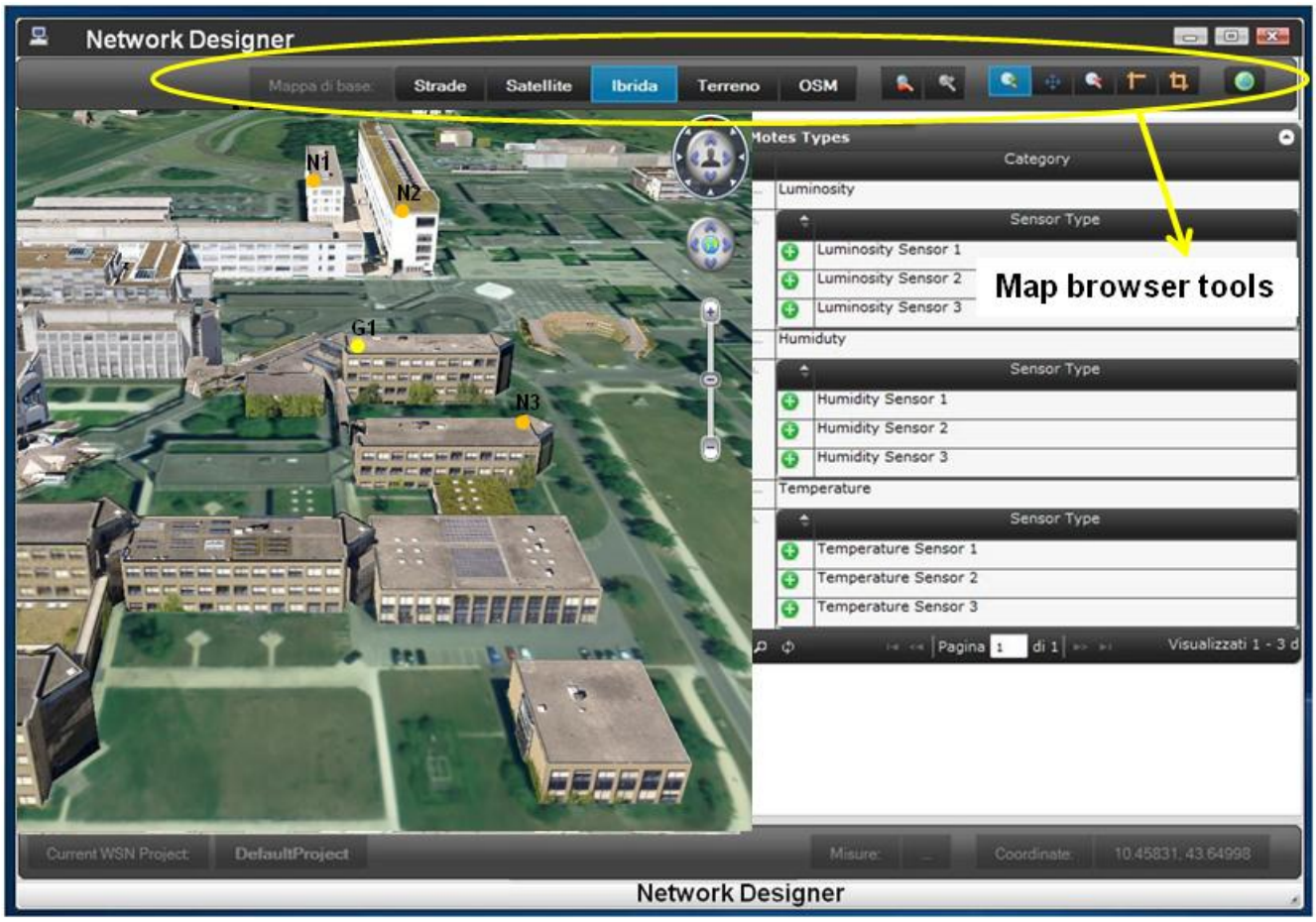

Figure 3.2: The Network Design Web Interface.

and all operations performed by the PT will be connected to the project. So all data retrieved from, and saved to the DPCM repository are always connected to the project. The project data are stored in the project repository and these data are available to all platform tools and middleware. Each project can be related with multiple Network Designs. A Network Design which is represented as a plan, contains information about the network topology, the type and the number of motes, the sensors enabled at each node and also information about RF simulation initialization and execution process.

The user after creating or opening a project, can create a new plan or open a saved plan. Additionally the PT GUI provides to the user the "Network design" functionality, in which a user can start the activity of mote positioning for the WSN application (defined by the project) under design. The "Network Design" GUI provides all planning tools necessary to easily define the mote geo-localization. The user selects the area of interest (AOI) on the map viewer by the map browser tools available from the GUI. The Map viewer displays maps as Digital Terrain Model, building, high resolution images, 3D building models, etc. useful for the planning purpose. The AOI defines the extent of area where all next processing will be applied. Defined the AOI, the user can start the plan definition inserting the mode by a click on the displayed map. Moreover the mote configuration will be defined when a mote is inserted.

Plans can be saved and opened in any time. The plan is stored in the DPCM repository and it is available to all DPCM platform tools and middleware when the plan is fine for the user.

Also the RF simulation function will be available by PT GUI. The RF simulation can be performed on a defined plan. The RF simulation function can be accessed in anytime. It can be performed immediately after the plan definition or in a next time.

The RF simulation GUI is inner the DPCM WebTop. It shows the list of available RF simulators within the DPCM platform. The user can display the description of each RF simulator in order to help the user to select that more suitable for your needs. Only one RF simulator can be selected for each simulation but several simulation cycles can be performed each cycle with different simulator or with the same simulator but different configuration. The selected RF simulator shall be configured providing the parameters values need. So the specific configuration panel will be provided. The simulation results are be displayed in a suitable graphical form dependently of the data type that shall be showed as represented in figure. The configuration and

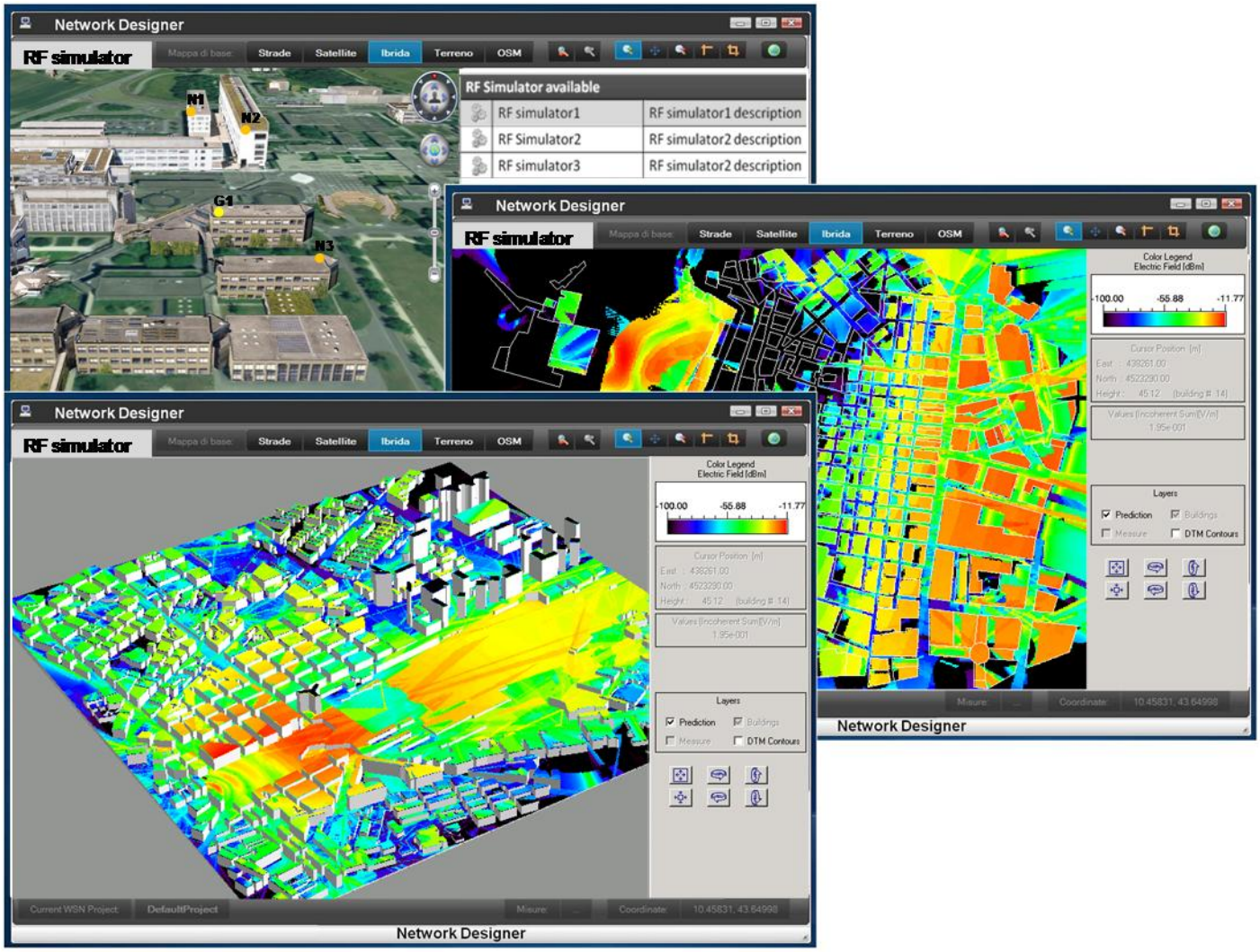

Figure 3.3: The RF Simulator Web Interface.

the simulation results are stored in the DPCM Repository when the user wants to save them.

3.2.2 The Development Tool

The Development Tool (DT) purpose within the WSN-DPCM platform is to synthesize and generate an application code, starting from designer specifications (components, constraints) and library components. DT also provide support for design test and refinement phases through application simulation. The NetSim Simulator is a part of the Development Tool and will be described in great detail in the following chapters, here it will be just outlined the DT basic modules and functionalities.

The Development Tool is made of a suite of the following top-level components:

- design check
- design synthesis
- simulator
- simulation data post-processor (validation)
- target development tools
- library
- project repository
- task drivers

3.2.3 The Commissioning and Maintenance Tool

The Commissioning & Maintenance Tool (C&MT) plays a significant role in overseeing deployed WSN networks and in providing the WSN DPCM platform with network feedback information in real-time. In short, the main capabilities and functional objectives of the C&MT within DPCM platform are:

- WSN real-time data and metrics monitoring

- Off-line analysis and statistics over collected network data
- UI data visualization (graphical & textual)
- WSN nodes and control through network commands
- Software-aided support for WSN deployment validation
- Support for in-field WSN commissioning and maintenance

The Commissioning & Maintenance Tool is mainly devoted to the operation of WSN-based networks but equally offers a valuable support for network deployers, testers and maintainers during the installing, setting up and commissioning works involved in a deployment. Its main functions are:

- **Network diagnostics** (actual routing, links quality) to validate network status. The services related to diagnosis operations focus on network properties that may be provided for monitoring and inspecting the behavior of a WSN in terms of coverage, connectivity and communication.
- **Node diagnostics** (battery level, status, location). The current status of network nodes and its location is essential information to be retrieved and visualized in order to assess if network units are operating as expected or are suffering from malfunctioning or energy exhaustion, among other problems.
- **Network data visualization.** On-site GUI for non-experts (PDA-like touch-screen device) and off-site GUI for engineers with additional capabilities to perform root-cause analysis (Coral2K®-based web application and back-end web server). These two natures comprised into the C&MT system ease the commissioning and maintenance activities. On either case, graphical interfaces for easy understanding of the WSN are available to the user.
- **Event and data logging**, stands for tool's support to the logging of events indicating normal or abnormal operation of the tool as well as data-related events like out-of-limits values of monitored sensors, network parameters or properties for instance. Likewise, network data received of any nature should be properly made persistent for on-line/off-line

statistical analysis. Log traces and data values are available at any time (GUI and log files) for inspection.

- **Alarms programming based on network and node events.** In response to certain received data values, the C&MT should allow controlling the environment by defining actions that should be taken over if an out-of-limits or rule-raised alarm has been raised.
  - *Command sending.* Users are able to configure the sending of a command to WSN as an action to be executed when particular alarm event arises.
  - *Notification sending.* Users are able to configure the sending of an e-mail to administrator users as an action to be executed when particular alarm event arises. API to allow easy integration with WSN applications is provided. Given its modular and plugin-based pattern-oriented design, the C&MT architecture is flexible enough as for new displays and/or core functionality (additional servers or services) to be added with a highly reasonable time for effort rate.

The Commissioning & Maintenance Tool consists of two main SW systems:

- A hand-held device, providing an on-site GUI for non-experts (tablet-like tough screen device).
- A server application, providing an off-site GUI for engineers to perform monitoring, control and easy inspection of the WSN network from a standard web browser.

In particular, the high-level software architecture of the whole system is depicted in figure 3.4. The C&MT has been conceived as a SW multi-layered structure, whose components are organized in a hierarchy of layers, each layer offering to the layer above/below it, a narrow and well-defined interface. The Web Service technology will enrich the system offering Services through Internet to web applications.

3.2.4 The Project Repository

The Project Repository (PR) is the tool which serves storage and integration purposes among the rest platform tools. The PR has been implemented in the Apache CouchDB [3] database management systems. The Apache CouchDB tool was reviewed to cover the WSN-DPCM platform

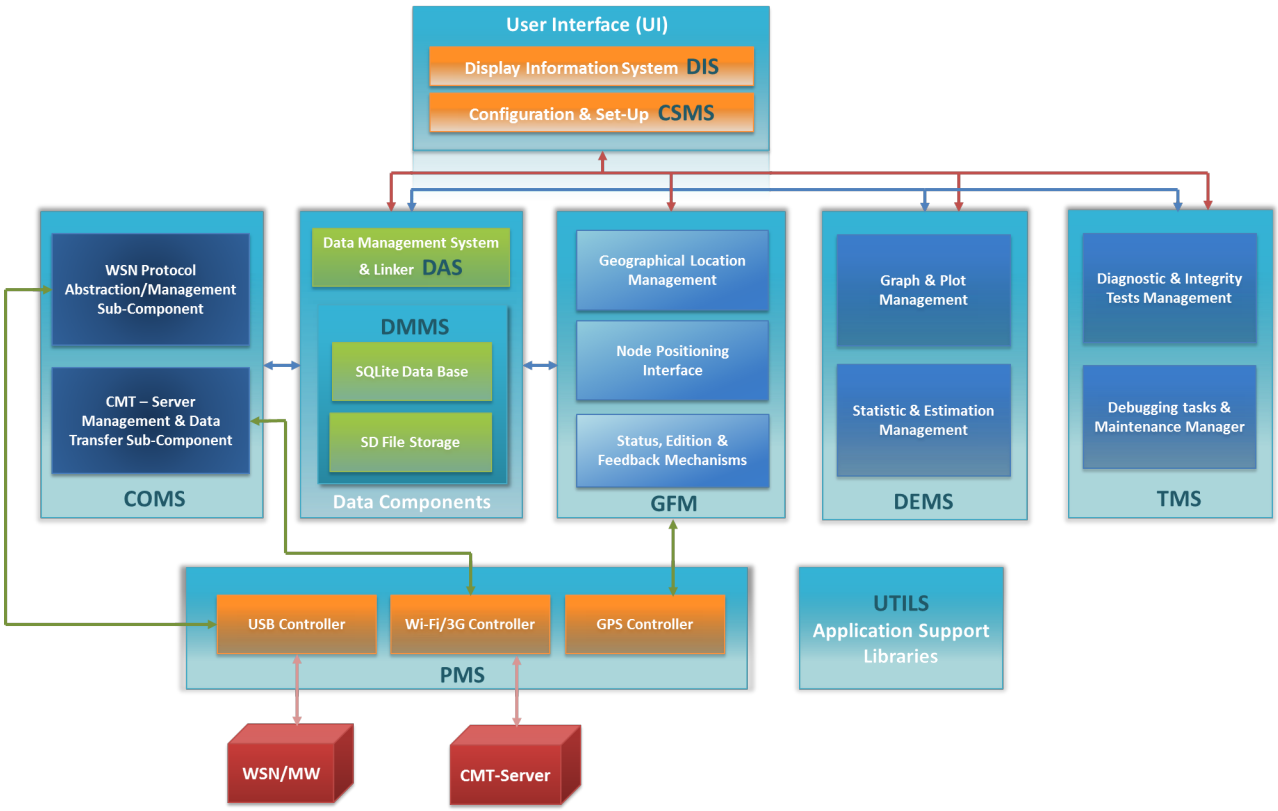

Figure 3.4: Commissioning and Maintenance Tool software architecture.

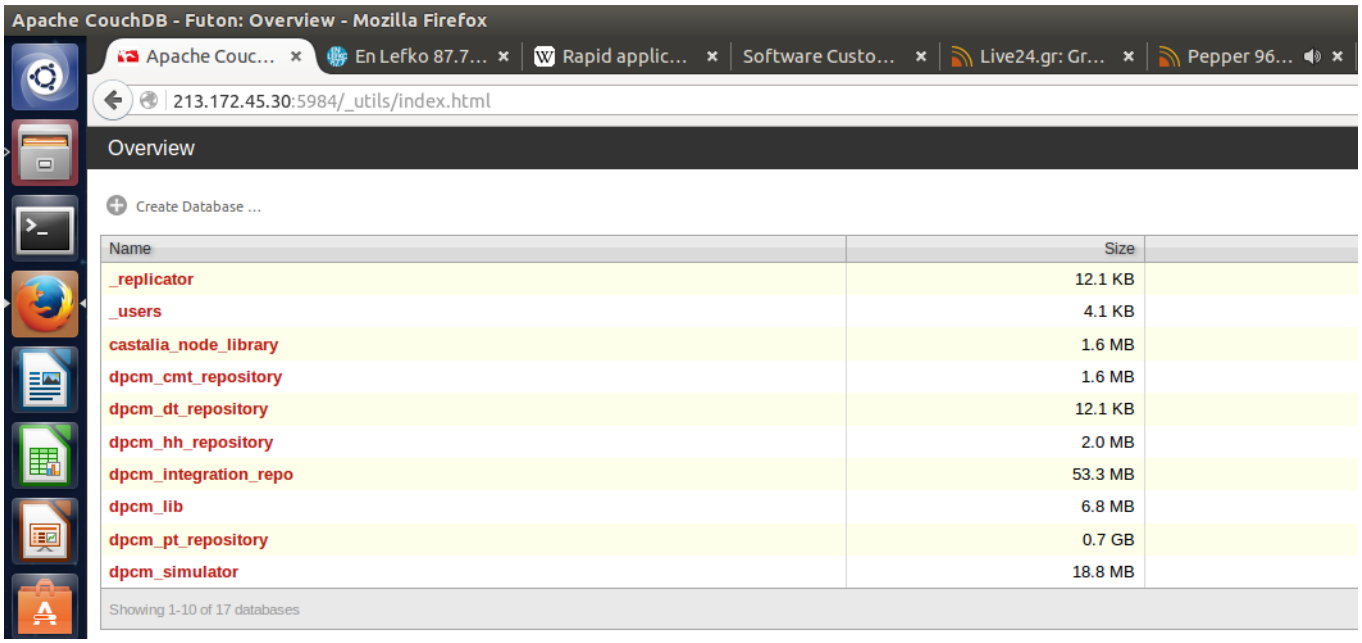

Figure 3.5: *Project repository list of databases created for the WSN-DPCM platform.*

needs, and it was selected to host the platform's repository instead of building and developing a new platform tool which would be more vulnerable and prune to inconsistency of the stored data.

The Apache CouchDB database management system.

CouchDB [3] combines an intuitive document storage model with a powerful query engine in simple and efficient way. CouchDB's design borrows heavily from web architecture and the concepts of resources, methods, and representations. It augments this with powerful ways to query, map, combine, and filter your data. Add fault tolerance, extreme scalability, and incremental replication, and CouchDB defines a sweet spot for document databases. Main features of Apache CouchDB are the following:

- Data are stored in JSON documents.
- Access documents and query indexes with web browser, via HTTP requests. Data can be created, retrieved and updated with simple HTTP requests from the WSN-DPCM platform tools.
- Index, combine, and transform your documents with JavaScript. Documents or groups of

document can be created, retrieved and updated via Javascript. Ease of integration with the WSN-DPCM platform's individual Web Interfaces.

The WSN-DPCM CouchDB Structure

For the needs of the WSN-DPCM project have been designed and created numerous JSON objects, which represent the data entities that each tool needs to operate, and a set of individual databases in which these JSON objects are stored. This set of databases consists the platform's Project Repository. In this section will be presented only the databases which are accessed and updated from the NetSim Simulator, since there is no need to unfold details which concern the rest platform's tools operation and runtime. These databases are:

- **Planning Tool Database**: All the information initiated and entered via the Planning Tool Web Interface is stored here. This information is related with user authorization, creation and manipulation of a project, and of course network topology and node placement. The JSON objects which are stored in the PT database and are accessed from the NetSim simulator are:
  - *Project*: The project object is a representation of a WSN application. Contains information such us the set of users which are authorized to access the project, the plan objects which are related with the current project (each project object can be related with multiple plan files) and some general information such us the timezone and the Coordinate Reference Systems which are selected by the user.
  - *Plan*: The plan object is a complete network representation. Contains information about network topology, the exact coordinates of the network nodes, information about the mote type of each node in the network, and also information about the connectivity range among network nodes. All the information necessary for the node positioning during the simulation process is retrieved from the plan object. Also the plan object contains information about the device specifications of each node mote device, and communication protocol's specification information which will be retrieved from the platforms's node library.

- **NetSim Simulator Database**: All the information related with a simulation process, execution, runtime and output is stored here. The type of JSON Objects which are stored here are:

  - *Network simulation descriptor (NSD) object:* This object contains all the information necessary for a simulation run. Consists of the following types of information: application, environment, data analysis and some global variables. The application level information regards all the information retrieved from the Node Library database, and gathers all the technical specifications of the devices and protocols that are simulated. The environment information regards all values that are necessary in order to feed the simulator with realistic input values, e.g the values which are measured from sensors during simulation. The environment values of a simulation, are modeled and configured via the VectorL Language, a language which has been designed for the needs of NetSim Simulator. The data analysis information regards the type of results and result visualization of a simulation run. The user can define all the data analysis information via the NetSim Web Interface. The rest global variables are values necessary for a simulation, e.g global simulation time, which cannot be classified in any of the above types of information.
  - *Simulation Output (SIMOUTPUT) Object:* Contains all the results generated from a simulation. The results can be either scalar values or figures. All the results generated from a simulation, are configured by the user during the simulation initialization phase via the NetSim Web Interface.

- **Node Library Database**: This database contains all the information and definitions related with the application level of a network. Here are stored objects such us technical details and definitions related with the type of motes, the communication protocols among network nodes, and similar information which is considered to be static since it concerns device and protocol specification. All this information must be retrieved from the NetSim simulator, must be mapped to the corresponding simulation configuration values so as to be exploited during the simulation execution. Here will be given some samples of the objects stored in the current database, just to outline what type of information is stored

here, since all the object types stored, contain information of device and protocol specifications.

- *Radio and Sensor objects:* Contain all the physical metrics related with technical specifications of a radio or a sensor module.
- *Mote Device:* This object except from the technical specification of a mote device also contains the JSON objects ids of the corresponding external-additional mote modules. For example a mote device object can contain the ids of a radio and a set of sensor JSON objects, which correspond to the radio and the sensors which the real mote device carries.
- *Mac:* This object contains all the information necessary to simulate the MAC communication protocol among motes.

Moreover in this database there is a JSON object for each mote type, each network communication protocol, each radio communication protocol, each different sensor type respectively. As far as the number of different mote types and protocols that the WSN-DPCM platform supports increases, new JSON objects must be created for the corresponding instances of data entities.

3.3 NetSim in WSN-DPCM platform

NetSim is a substantial part of the WSN-DPCM platform since it provides network performance optimization abilities to the user. The user can run multiple different simulation via the NetSim so as to inspect the network's performance, and furthermore to optimize and correct undesirable operation. Despite that, the user can watch in a user friendly manner the network's operation via the data analysis module. A facility for post-processing collected statistics which allows users to summarize, aggregate and transform collected statistics. Users can design custom plots of collected or computed statistics. These plots can either be generated as images by the simulation engine (via Gnuplot), or, alternatively, the data can be sent to the Planning Tool in order to be displayed on the project's map. Another feature of the NetSim simulator is the ability for the user to completely specify the simulated environment model via the VectorL language .

3.3.1 NetSim functionality in the overall platform

NetSim offers three major functionalities in the overall platform: to retrieve all the simulation related information from the PT, to offer the ability to the user to create and configure an NSD file via the NetSim Web Interface and to upload the simulation results in the Project Repository in order to be accessed from the rest platform tools and mainly the PT. You should notice that NetSim functionality is strictly connected with the NSD entity and the simulation output entity, the second is directly related with the first. Since NetSim implements actions over these two types of entities, NetSim is build in an NSD model oriented philosophy, meaning that all the methods of the NetSim service are designed to manipulate NSD entities and NSD related entities.

Retrieve information from Planning Tool

As it was described in the previous section, the user uses the PT Web Interface so as to create a project and to design project associated plans. All this project information is stored in the PR with the necessary user ownership fields. So when the user visits the NetSim Web Interface to design a NSD file, all the information related with the user's projects must be retrieved from the PT. So NetSim firstly retrieves all the project objects associated with the current user from the PT, secondly all the plan objects related with the selected project, and finally all the existing NSD objects related with the selected plan file. From the retrieved information the NetSim identifies the mote models and additional devices and protocols defined by the user, and retrieves all the necessary JSON objects from the Node library database. So at this phase the NetSim retrieves all the information related with the network topology, the application specification of mote devices and communication protocols defined by the user via the PT Web Interface. All these JSON objects are retrieved via the NetSim Rest API from the PR, API which will be thoroughly presented in the following chapters, during the simulation initialization phase before the NSD design phase begins.

NSD file creation and configuration

As it was mentioned in previous sections, the NSD file contains all the information related with a complete simulation process. Except from the information fetched during the initialization phase, the user must design a simulation by entering additional information, which regards the simulation environment and the data analysis parts, tasks which are implemented via the VectorL editor and data analysis utilities respectevely. Also at this phase the user defines a small set of simulation variables such us simulation time. After the user enters all these information, the NSD file is saved and uploaded in the PR, and is available for downloading and further editing from the user.

Store simulation output information in the PR

Each simulation execution must be related with exactly one NSD file. The NSD file contains all the information necessary for a simulation run. After the NSD file creation the user can run a simulation. The simulation start is fetched via the corresponding NetSim Rest API method. With the invocation of this method a simulation result JSON file is created and uploaded in the PR with some initial configuration parameters, after the simulation execution completion the file is updated with all the simulation results and further simulation execution details. The simulation output file will be thoroughly described in the following chapters.

Chapter 4

Description of the NetSim architecture and modules

This chapter demonstrates the overall NetSim architecture and the complete simulation execution runtime. The NetSim modules are extensively described and analyzed so as to understand NSD model oriented design, and the complete simulation runtime from initialization to the simulation output generation. At first we present the NSD model, the data formats and the data types which the simulator uses and produces. The overall NetSim functionality is presented by explaining both front-end and back-end interfaces and software components. The NetSim REST API, Admin Gui and server's database are demonstrated, and features such us NetSim NSD Editor and data analysis are covered. Great attention is given to the code generation functionality, which integrates the Castalia Network Simulator and Omnet++ simulation framework, and is considered to be the heart of NetSim and furthermore the most innovative feature in comparison with other network simulation frameworks. Finally the validation process of NetSim's auto code generation flow is explained.

4.1 The NSD Model

The NSD model is the entity where all the information necessary for simulating a Wireless Sensor Network is gathered. This information is classified according to different system and design layers. As it would be shown later the architecture of NetSim service is NSD oriented since every

action of NetSim is over NSD object instances or NSD related objects. The information included in an NSD object is categorized in the following types:

- **Physical process:** All the parameters that concern the simulation environment e.g the measured-input values of the sensors and the general models that generate these values. In general the definition of the simulation environment is not so straightforward, and for that reason is designed and developed a programming language called VectorL. The user can write VectorL in order to define all the environmental parameters that concern the physical process, using the VectorL editor which is included in the NSD Editor.

- **Network topology:** All the parameters that concern network topology and exact node positioning. All this information comes from the planning tool, and is retrieved via the NetSim REST API from the Project Repository.

- **Application:** All the parameters that concern the mote characteristics, mote functions and operation and all the characteristics of the mote hardware. This information is retrieved from the Castalia node library [3.2.4].

- **Data analysis:** The parameters that concern the simulation output data in which the user is interested in. These data include information which is given by the user and concern the metrics that the user wishes to observe and monitor, the output data type either scalar or graphs, and their exact format.

By introducing this NSD schema we achieve to include all information necessary to model and execute a simulation, in an integrated schema which comprises of four main different types of information. In figure *4.1* we can see all the information included in the NSD model, and from which json objects each information type is retrieved.

4.2 NetSim Functionality

The main purpose for developing NetSim, was to be able to map via an auto code generation process the user input and simulation configuration parameters to an already existing network simulation framework. This operation demands a powerful, fully configurable tool

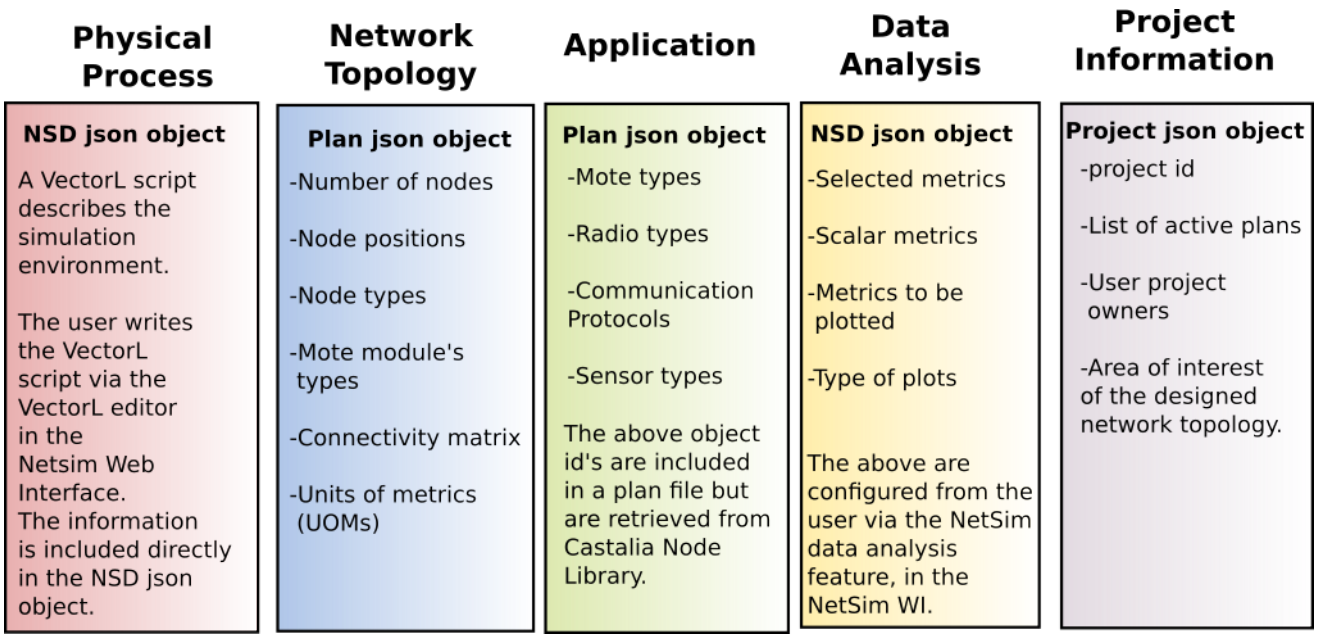

Figure 4.1: NSD Synthesis.

- with great flexibility, so as to enclose all the existing parameters and scopes, and to support many different simulation scenarios, and
- with high performance so as to achieve efficient system responsiveness and scalability, to the concurrent platform users .

In order to build a light and fast code generation tool, the Python language was selected for the NetSim development cycle. The main functionalities of the NetSim tool is:

1. To retrieve all the existing simulation related information stored in the PR from the rest WSN-DPCM platform tools.
2. To provide a Web Interface (the NSD Editor) via which the user will define and configure exquisite simulation parameters and values.
3. To provide a mini data analysis framework via which the user will select the simulation results he/she is interested to observe and furthermore the way that wishes these results to be presented, either scalar or figures. Of course the last ability is limited from the result type (e.g the total number of messages exchanged during simulation is scalar and cannot be represented as a time related figure).

4. To generate all the build and make files for a simulation process.

5. To generate all the Castalia simulator files and to map the collected parameter values to the corresponding parameters of Castalia simulator.

6. To validate the code generation and the parameter value mapping processes.

7. To select only the output values which the user wishes to be stored and displayed from the default output values which the Castalia simulator produces.

8. To persist the simulation output to the PR so as to be available to the user and to the rest platform tools.

4.3 NetSim Architecture and Modules

The functionality and operation of NetSim is separated in three main phases:

- *Simulation initialization and configuration:* During this phase all the input data necessary for a simulation run, must be either retrieved for the PR or must be configured by the user from the NetSim NSD Editor. The data retrieved from the PT mainly regard network topology, mote characteristics and other network design parameters. The data entered via the NetSim NSD Editor regard more simulation oriented parameters which will be analyzed in the following sections. After the completion of this phase the simulation execution can begin since all the input data have been collected and stored locally at the NetSim Server.

- *Simulation code generation and execution:* This is the phase where the code generation takes place. All the build (makefile, configuration etc), source and library files are produced and all the collected data from the initialization phase are mapped to the corresponding parameters of the Castalia simulator. Finally after the code generation completion, the simulation job is executed and starts to run.

- *Generation and persistence of the simulation output:* During this phase , the output produced from the Castalia simulator is configured and processed so as to retrieve all the

metrics in which the user is interested to observe. Furthermore the output data are transformed in order to be presented and visualized to the user in the form which the user requested via the Data analysis feature of the NetSim Web Interface. Finally the simulation output file is produced and uploaded in the PR, file which contains all simulation results and additionally all the information produced during simulation execution which might be useful for the user.

4.4 NetSim Input

In order to create and run a simulation, the NetSim tool needs to gather all the input given by a user via the rest platform tools and to integrate this information to the NSD model . This information is stored in the PR in json format. A set of specific json objects has been designed and developed in order to fulfill every aspect of a WSN design. The json objects necessary for the NSD model built and the information contained at each of them are presented in figure *4.1* and described in section *3.4.2*.

4.5 NSD Editor

The NSD Editor is an on line tool meant to aid DPCM platform users define, access and manipulate NSD files and VectorL modules for their projects. The user after completing the network design in the PT is redirected from the PT to the NSD Editor in order to continue with the NSD design. The user is redirected to the NSD Listing screen. In the NSD Listing screen the user can see for each NSD its name, the project under which this simulation descriptor was created and the plan it utilizes. Every NSD file can be cloned, deleted and edited.

Each part of the NSD file can be configured from the equivalent NSD editor section. The NSD editor sections will be just outlined since are not work of the current thesis, although is necessary to be mentioned in order to understand the complete NSD model design flow: So the NSD editor main sections are the following:

- **Network section:** In the Network section information related to network's set up has to be defined. User has already entered this information using the planning tool, so the user

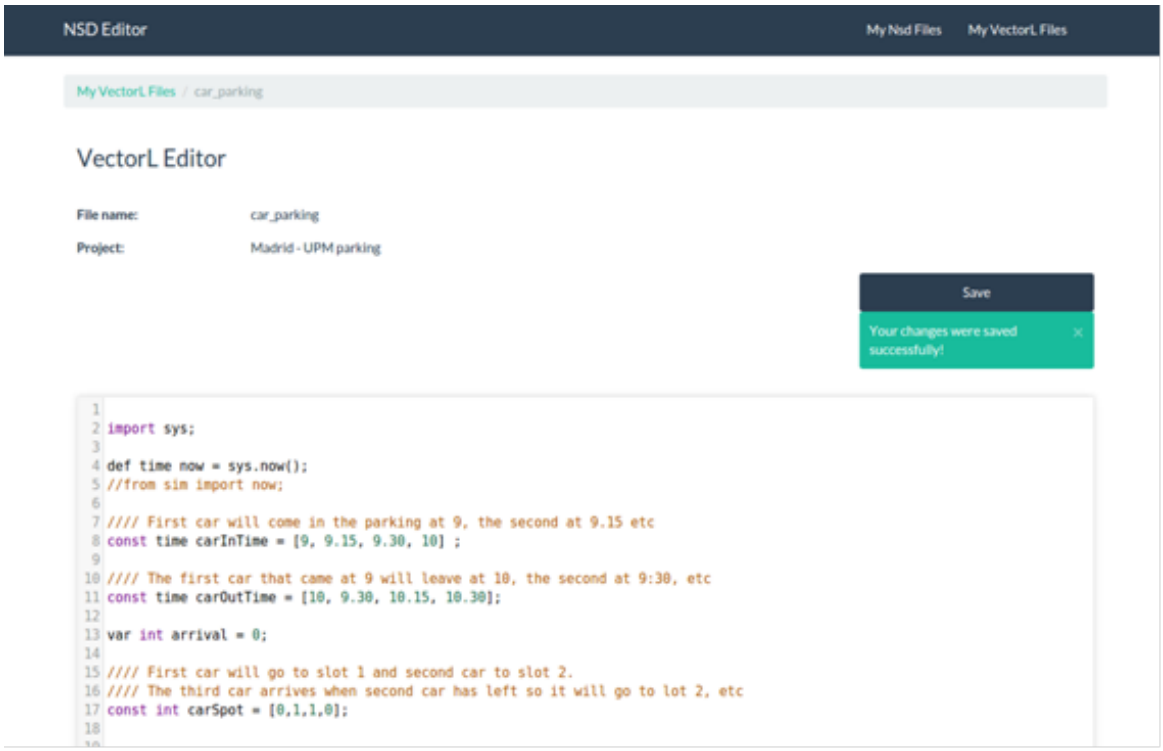

Figure 4.2: VectorL Editor

simply selects the plan that is going to be used for this simulation. User is been provided only with the plan names that are created under the project that current NSD belongs to.

- **Simulation Parameters section:** In this section the user needs to provide the values of the parameters related to the simulation process, these are simulation time limit, simulation time scale and CPU time limit.

- **Environment-VectorL editor:**In this section the user has to define the model to be used for environment simulation. Environment simulation will produce realistic sensor values that are going to be used during network simulation, leading to more interesting and accurate simulation results. The user can choose either a default module implementation from Castalia library or write his own model using the VectorL domain specific language and VectorL Editor. Those VectorL modules are attached to an NSD file and are mapped by NetSim to the Castalia external environment simulation process.

- **Hardware in the Loop (HiL) section:** The user can design a WSN by using HiL module of the Development Tool, in order to simulate the behavior of a network application. To

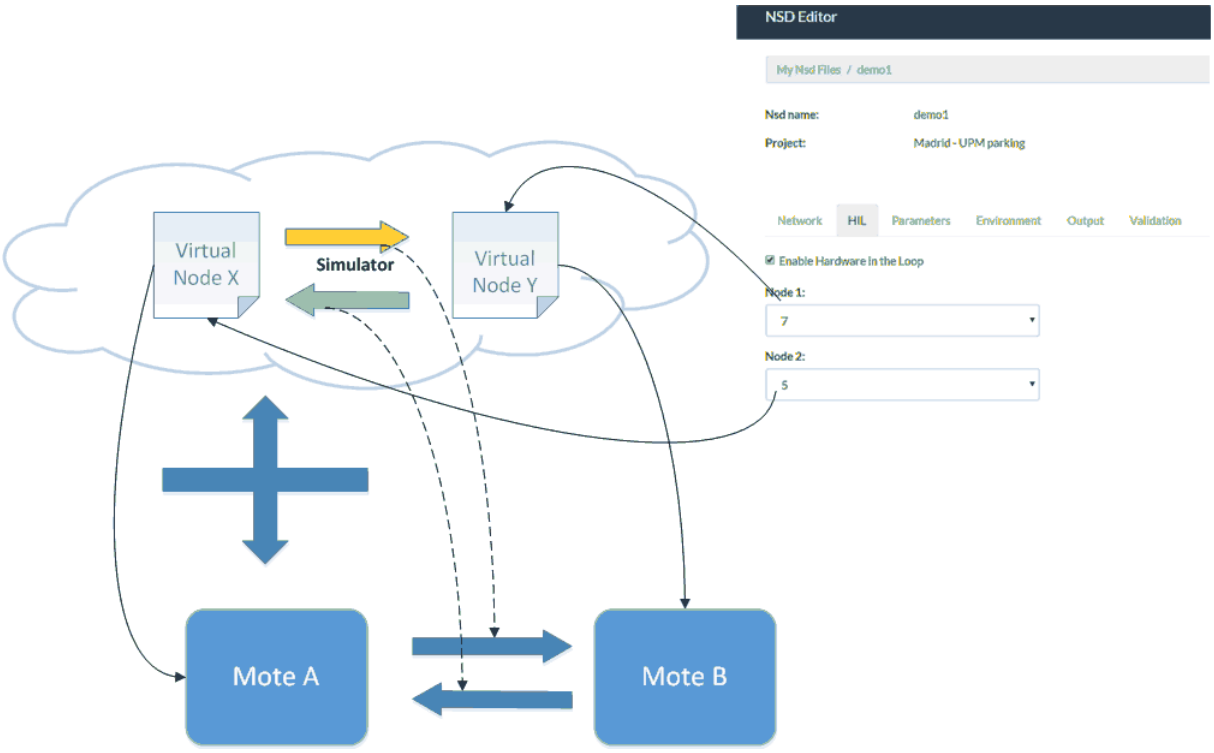

Figure 4.3: HiL section of the NSD Editor. The user selects which nodes are simulated and which are executed in real time.

incorporate the HiL into the simulation procedure the user can plug in physical nodes. For each node the user is able to define which parts of the software stack will be simulated and which will be executed by the available hardware. The Hil section offers the ability to the user to select which nodes will be simulated and which nodes will be executed in real time. Additionally offers the ability to the user to select one or more modules of the software stack of the selected HiL nodes, in order to be simulated or executed in real time. The HiL integration to the NetSim tool is not part of the current work so it will not be further analyzed. In general through the HiL section the user is able to activate the integrated HiL capabilities of the DPCM Network Simulator. To do that the NSD editor offers all required configuration capabilities for the particular HiL feature.

- **Output-Data analysis feature:** This feature allows the user to produce custom statistics from the default Castalia simulator output. The Castalia simulator gives hundred of parameters as an output in plain text format. It would be very frustrating for the user to evaluate all these data in the current format. So the user can configure the metrics that

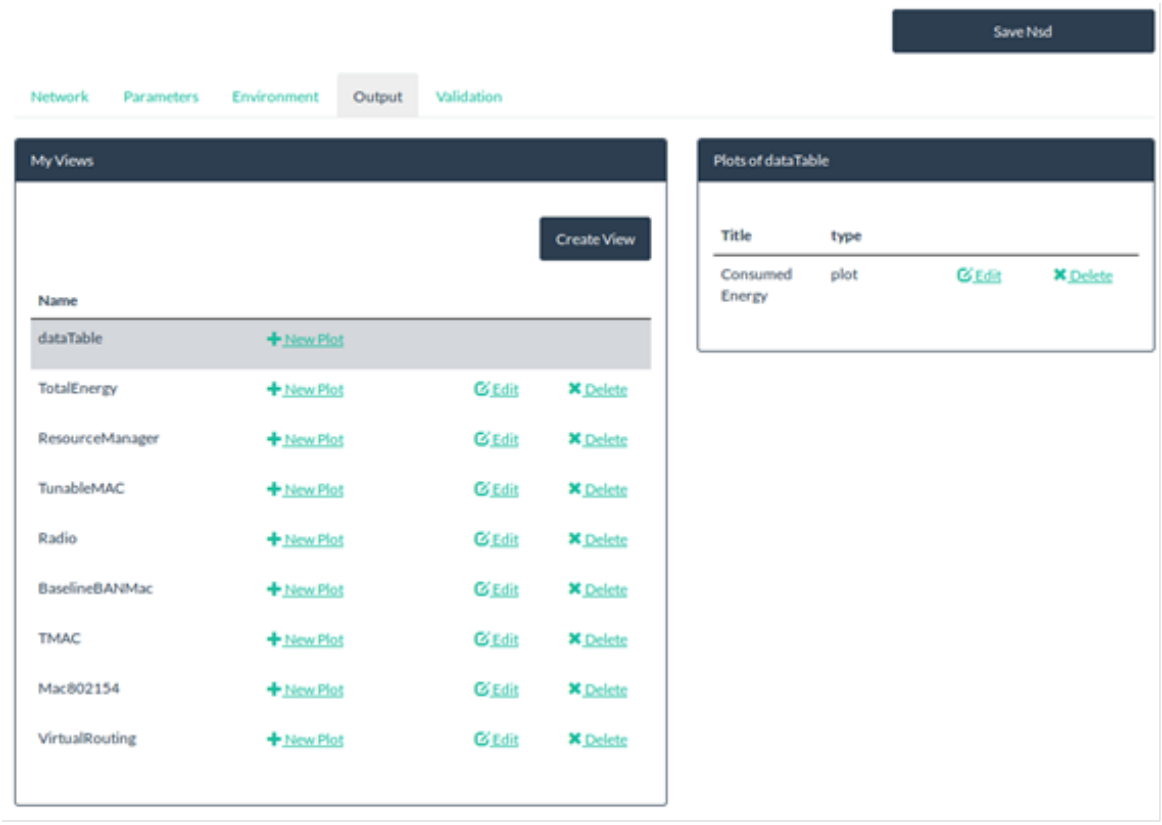

Figure 4.4: Data analysis feature in the NSD Editor

wishes to observe and monitor. In this section the user sets all the configuration in order for the NetSim to produce custom statistics, either graphs or scalar values, from the default output generated by Castalia simulator. The Data analysis feature offers the functionality to the user to create custom tables from the default Castalia output data, using a mysql like syntax. So after the simulation is finished NetSim produces a custom dataTable as it was declared from the user in the data analysis feature. By applying basic functions to the custom dataTable, NetSim extracts the scalar values and the graphs using the user configuration.

4.6 NetSim REST API

The NetSim REST API is a service which implements communication and integration functionality with the PR. Via the REST API the NetSim can create and manipulate NSD and NSD related json objects in the PR, so the REST API performs the NSD json object persistence and retrieval

actions in the PR. As it was described in previous sections the NSD synthesis process requires information which is stored in json objects persisted in the PR by the rest platform tools. Via the REST API NetSim is able to retrieve these NSD related objects in order for the NetSim tool to synthesize a NSD model. The NetSim REST API except from performing actions which are parts of the automated NSD model generation and synthesis process, performs all the communication with the PR which is necessary in order for the user to build, create, configure or edit an NSD object via the NSD editor. In other words manipulates the overall NSD construction and configuration process as far as it concerns communication needs of the NetSim with the PR. In general the NetSim Rest API is designed in an NSD model oriented philosophy so as to cover the purposes of the overall NSD model synthesis and persistence in the PR. The functionality which the REST API implements is :

- The retrieval of the plan and project json objects related with an NSD, files necessary for the NSD network layer configuration.
- The retrieval of all the json objects necessary for configuring the NSD application layer, objects which are persisted in the PR Castalia node library.
- The retrieval of all the json objects which are necessary in order for the user to configure an NSD model via the NSD Editor.
- The persistence, update and manipulation of the NSD json objects in the PR.
- The persistence and update of the SIMOUTPUT json objects which contain the simulation output related with the an NSD object.

4.7 NetSim Admin GUI

The NetSim Admin GUI is a tool ideal for inspecting simulation tasks in different execution phases and offers a global overview of the server setup. This tool gives the ability to the user to observe the overall simulation execution process and resolve more easily any problems that may arise. Of course, as the NetSim server implementation matures, hopefully the need for this kind of intervention will lessen. However, experience shows that it is often necessary to provide

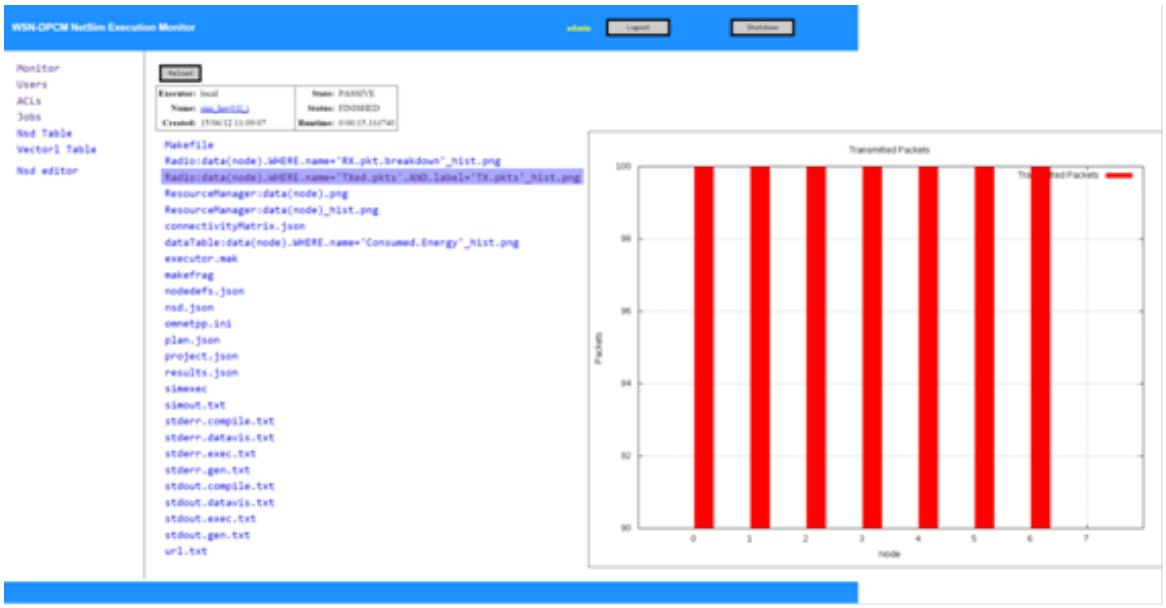

Figure 4.5: Simulation job inspection module.

this kind of "internal view" into the workings of a complex system. Finally must be noted that the Admin GUI gives the ability to perform more administrative tasks that exceed the role of a typical user. The main functionality of the NetSim Admin GUI is summarized in the following features:

- **Simulation job inspection module:** The NetSim Admin GUI has the Jobs console, from where the administrator can inspect all jobs in the system and their state. If the administrator wishes to inspect a particular job in more detail, has the ability to select a job and to go to the Job Inspector screen. Here, the administrator can actually see the contents of the simulation job's home directory, which include the inputs and outputs files generated during the various stages of the job execution.

- **Project repository inspection module:** It is often useful to interact directly with the project repository, in order to solve problems related to the overall server execution (such as the code generation). In order to support this, the Admin Console offers the ability to inspect the contents of the Project Repository, and in particular interact with the two main types of objects pertaining to simulations: Network Simulation Descriptors (NSDs) and VectorL modules. For both of these types of objects, the Admin Console offers a Browser, which displays all the objects stored in the Project Repository. Also, an administrator can execute

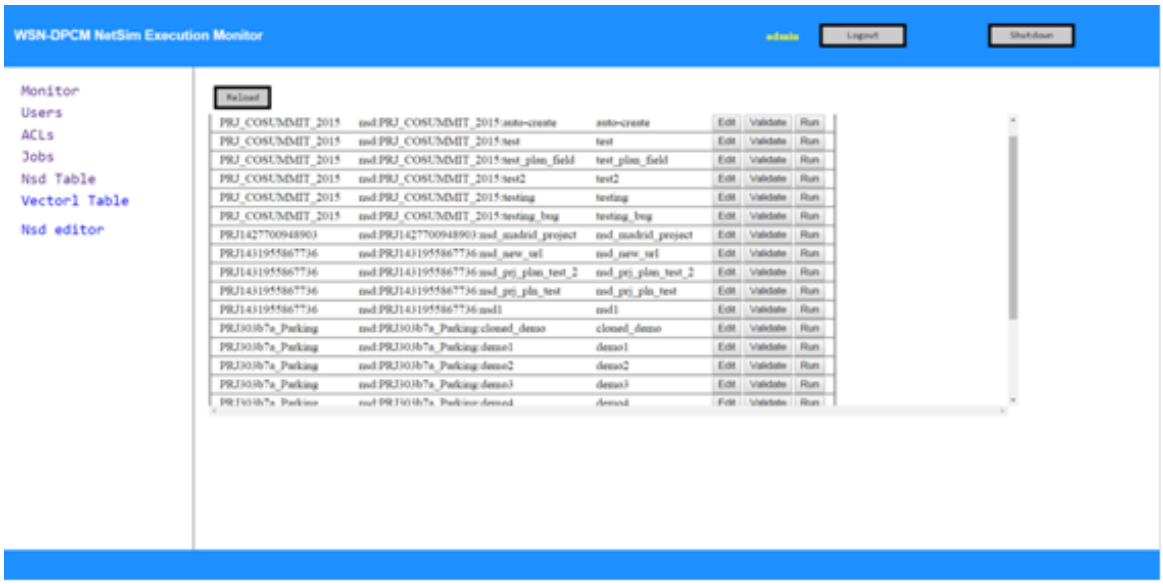

Figure 4.6: NSD Inspection module.

simulation jobs using these objects, or validate them. This functionality is also available to regular users of the NetSim platform, but the administrator has the added ability to monitor the lifetime of jobs once they started.

- **NSD file and VectorL module inspection:** The inspection browser allows one to see all the NSD files and VectorL modules visible by the NetSim server, providing also the actual Project Repository IDs (which are not available to regular users). Furthermore, the administrator can validate and/or execute an NSD (create a simulation from it) and to inspect the compilation and execution behavior of a VectorL module.

4.8 NetSim and Code generation

In this section the code generation process of the NetSim runtime will be explained. Firstly the NetSim database server will be presented, in which all the information related to running simulation processes is organized and persisted. Afterwards the operations and the steps which occur during each individual execution stage of a simulation run will be covered, so as to unfold the complete simulation runtime.

4.8.1 NetSim Scheduler and Database Server

At this point the structure of NetSim server will be described, meaning the overall simulation runtime framework and architecture of simulation jobs' scheduler. The NetSim server has a database used for storing runtime information, meaning information related with the state of all the jobs which have been executed, are executing or have been created and are going to be executed immediately. Each of the NetSim's runtime database entities has a corresponding implementation class written in Python, with a set of methods which implement object's functionality and behavior during all stages of execution.

simjob: The entity simjob represents a simulation task-process and stores all information related with a simulation job. Some parameters stored are the current execution state of the job, the local file directory of the job, a directory where all the files necessary for the execution are stored (build files, simulation input and output files, source and header files, stderr and logfiles etc). A simulation job can be in one of three states:

- **ACTIVE:** some stage of the job is currently being executed by a worker thread.
- **READY:** there are stages of the job that are waiting to be run.
- **PASSIVE:** the job has finished execution (successfully or not) but the job files are still retained by the server for inspection.

Manager: The entity Manager represents the task scheduler of NetSim. The Manager is the object responsible for creating a simulation job, assigning a job to an execution thread and creating the job record in NetSim database.

Executor: The entity executor represents the thread which is assigned with the complete execution of a simulation job by the Manager. The executor creates a simulation, the simulation's local file system, inspects the simulation execution and is responsible for all the simulation execution state transitions.

Greater coding details will not be given, since main purpose is to outline NetSim's architecture. In the existing implementation the maximum number of simulation jobs can be up to 4, meaning the number of the executors in the Manager is 4, so the maximum number of simulation jobs

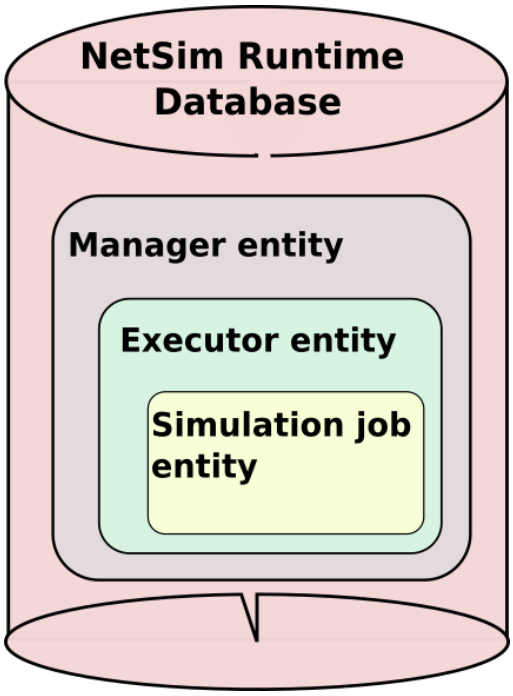

Figure 4.7: *NetSim Runtime Database structure.*

that can be executed in NetSim simultaneously is 4, since each executor is assigned with the execution of only one simulation job at each time. Of course this can be configured to change so as to cover the overall platform's needs, regarding the number of users which the platform serves and the complexity of the designed networks.

4.8.2 Initialization

During this phase a simulation job is created and initialized. The simulation job's local file directory is created and all the initial files of a job are generated. At first inside the job's local file system are placed the json files downloaded from the PR, which will be further processed during the next execution phases, and a file with the simulation output json file location in the PR repository. Also the simulation output json file is created and uploaded in the PR. The simulation output json file at first contains information related with the ids of the relative with the current job json objects (NSD, project , plan etc.) located in the PR, and some extra simulation initialization parameters. This json file has the field job state which is updated at each execution phase accordingly, and only when the simulation job finishes successfully is updated with the simulation results, during the finalization phase. Also a simulation job entry is created in the

NetSim server database.

4.8.3 Generation

Main target of the NetSim tool, during this stage, is to map all the information entered by the user to the existing Castalia simulator. In order to save effort and reduce errors during the mapping process, the model to model (M2M) transformation technique has been selected. For the M2M transformation and NetSim code generation, an initial implementation attempt has been done using the Eclipse Modeling Framework [10], a code generation facility for building tools and other applications based on a structured data model. The EMF soon was proved to be over-engineering, with greater complication degree than necessary, fact which caused great frustration and lose of time during development. Therefore the NetSim M2M transformation and code generation process was designed and built from the beginning in Python.

Model to Model (M2M) transformation

Model to Model (M2M) transformation is a process of mapping the information included in a model to another, via an automated transformation engine. In order to describe the M2M transformation design stages, we consider the initial information as the initial model and the generated information as the target model. So during the M2M transformation the following tasks must be performed:

1. Define the initial model, meaning a representation of all the initial information, which is going to be mapped to the target model, in a consistent and concrete model definition. All the possible attributes of the initial information must be explicitly defined, and all the relationships between these attributes must be represented in a definite way.

2. Define likewise the target model if does not exist.

3. Define in an explicit way the mapping rules and relationships between each attribute of the two models.

4. Define default values for the target model's attributes that cannot be mapped to any of the initial model attributes.

5. Enrich the target model with additional attributes in order to map information of the initial model, which cannot be mapped with the existing target model representation.

6. Design and implement the transformation engine, which will automatically take the initial model as input and will produce the corresponding target model as output. The representation formats of the two models, as far as the input and output formats of the transformation engine, are selected by the developer according to the type of information included in the models and the overall transformation process.

Castalia model in NetSim

During the generation phase all information included in the NSD file is extracted and mapped to the Castalia model. The NSD file, which constitutes the simulation model representation, was described in previous sections. The Castalia model explicitly designed for the needs of NetSim tool includes the following objects, each one of which represents a different kind of information required for the configuration and execution of the Castalia simulator:

1. Network: Includes all the network parameters and values.

2. Mote: Include all the mote device parameters and values.

3. Position: Describe the x,y,z global coordinates of a node in the network topology.

4. Plan: Includes all the parameters and values of a plan designed by the user via the PT Web Interface.

5. Project: Includes all the parameters and values of a project created by the user via the PT Web Interface.

6. VectorlEnvironment: Includes all the parameters and values that describe the simulation environment, refer to the physical quantities sensed by the sensors, and are extracted from the VectorL script written by the user via the NSD Editor.

7. NodeDef: Includes all the parameters and values related with the behavior and the role of a node inside the network. In the PR is stored a specific number of predefined node types, which consist fixed node behavior templates.

8. ConnectivityMatrix: Include all the information about the links between the nodes according to the corresponding node positions. These links are virtual, since are related with the position and radio frequency of each node but need to be defined somehow, so are defined inside the connectivity matrix file. For further detail about in the connectivity matrix contents please advise the Castalia documentation.

9. Sensor: Includes all the parameters and values related with the corresponding real sensor which is installed in a real mote device.

10. MoteType: Includes all the parameters and values related with the corresponding real mote device.

11. RadioDevice: Includes all the parameters and values related with the corresponding real radio device which a mote wears.

12. Application: Includes all the additional application related information downloaded from the node library, and cannot be included in any of the above objects, e.g the communication protocol between network nodes.

The above objects contain all the information necessary to run a Castalia simulation inside the WSN-DPCM framework. This means that these objects, do not only contain all the configuration information necessary for a Castalia simulation to run, but additional identification, persistence and relationship information in order to run it at the WSN-DPCM platform. For example we need to relate each Castalia simulation with a plan object, plan designed by a user via the PT Web Interface and persisted in the PR. So we need to keep relation information between the json objects stored in the PR, during runtime, since there might be need for downloading additional information from the PR during Castalia simulation configuration and execution. Likewise there is the need for relating each Castalia simulation execution with the corresponding NSD and Project json files stored in the PR.

NetSim NSD to Castalia model transformation

Now consider the NSD as the initial model, the Castalia simulator as the target model and NetSim as the transformation engine. During the generation stage all M2M transformations from

an NSD to a Castalia simulator model takes place. All the simulation parameters included in the NSD file are mapped to the corresponding parameters of the Castalia simulator. **Of course all the steps required for the M2M transformation, have been implemented during the design and the development of the NetSim tool, since the M2M transformation is a model oriented process difficult to be generalized above a precise model definition.**

After the completion of the NSD to Castalia model transformation, all the parameters from the Castalia model are passed to the configuration files of the Castalia simulator which are necessary for a simulation run. These files are the omnetpp.ini file and the connectivity_matrix.txt file, the first contains all the parameter initialization values and the second the information about the strength of the connection links between the network nodes. Once the generation phase is completed, the simulation job is ready to start being executed.

4.8.4 Validation

The software Validation is defined as the engineering process of ensuring that the conceptual system is sufficiently accurate to meet expectations from the perspective of its intended uses. In other words, it pursues evaluating and assessing the correctness, the reliability, and the quality of the developed system to ascertain that the right solution according to system and user requirements was built and is compliant with technical specifications, but not only.

A network simulation contains a lot of information and depends on a large number of parameters. It is not difficult for errors or omissions to enter into the provided data. Although NetSim tries its best to assure that such problems do not arise, it is not feasible to foresee all possibilities.

In NetSim the validation process is part of the code generation process, or the first part of the code generation process. Each NSD json object contain, as it was described above, the id's of the node descriptor object and other application descriptor objects which must be retrieved from the Castalia node library located in the PR. So the validator checks at first if these id's are valid and then authorizes NetSim to download them. Once the above objects are retrieved the NetSim validation module traverses the objects of the NSD model and checks the validity of the data, by accurate checking of the syntax and the parameter values of all the gathered object data.

Once the validation process is completed the simulation execution can be started.

4.8.5 Compilation

During the NetSim compilation phase all the build files of a simulation job are generated. In order to execute a simulation task, NetSim needs to generate all the Omnet++ build files for the current job, since each simulation in Castalia is executed via the Omnet++ framework.

The NetSim server where the simulations are going to be executed must have:

- A working installation of Omnet++ version 4.2 and above.
- The Castalia simulation library stored in the same folder with Omnet++.

After ensuring the above, NetSim server must be "informed" about the installation directories of Castalia and Omnet++ so as for the NetSim to be able to execute simulations using Castalia and Omnet++ framework. This setup process takes place only once, during NetSim Server setup and configuration, via Python scripts. Once NetSim is being informed about the relative installation paths then can invoke and build Castalia simulations.

For each simulation job submitted, NetSim performs the following steps during the compilation phase:

1. Create the makefile and additional build files of the simulation job.
2. Link to the Castalia and Omnet++ libraries.
3. Compile any source files if needed.

4.8.6 Finalization

This is the last phase of the simulation execution life-cycle. At this phase we suppose that the simulation job execution has finished either successfully or aborted. NetSim needs to update the simulation output file located in the PR so as to hold the simulation results and the final simulation job execution *state*, either "ABORTED" or "FINISHED". In case where the simulation job did not ended successfully, the Castalia simulator has not generated any results, so the only task NetSim needs to perform is to update the SIMOUTPUT object (located in the PR) *state* field to "ABORTED". In that way the user is informed about the simulation job status, and can

examine the simulation log files using the NetSim Admin GUI in order to figure out the reason why the simulation execution has not terminated successfully.

In case where the simulation job has terminated successfully, the Castalia simulator has produced the default simulation output in a text format. So this default output must be filtered and transformed in order for the NetSim to extract the results in the user configured format, as it was defined via the data analysis module of the NSD editor. NetSim tool has a data analysis module called Datavis (Data visualization), which is responsible for performing all the transformations necessary for extracting results from the default Castalia simulator output to the form defined in the user configuration. Datavis is a substantial NetSim module, implemented in Python which uses Gnuplot library for creating result graphs, but is not part of the current work so it will not be further analyzed. So NetSim performs the following steps after a successful simulation job execution completion:

1. Using the user configuration defined via the data analysis NSD editor module, Datavis creates from the Castalia default output the datatable with only the results the user is interested in.

2. Using the Datavis module NetSim transforms the results contained in the datatable in the user configured format, which may refers to either scalar values or graphs.

3. These result data are translated into json format and are included or attached, according to the result format, to the simulation output json file using the appropriate json syntax.

4. NetSim updates the existing simulation json object located in the PR with the one which contains the user configured simulation results, where the simulation output json object *state* field is defined to be 'FINISHED'.

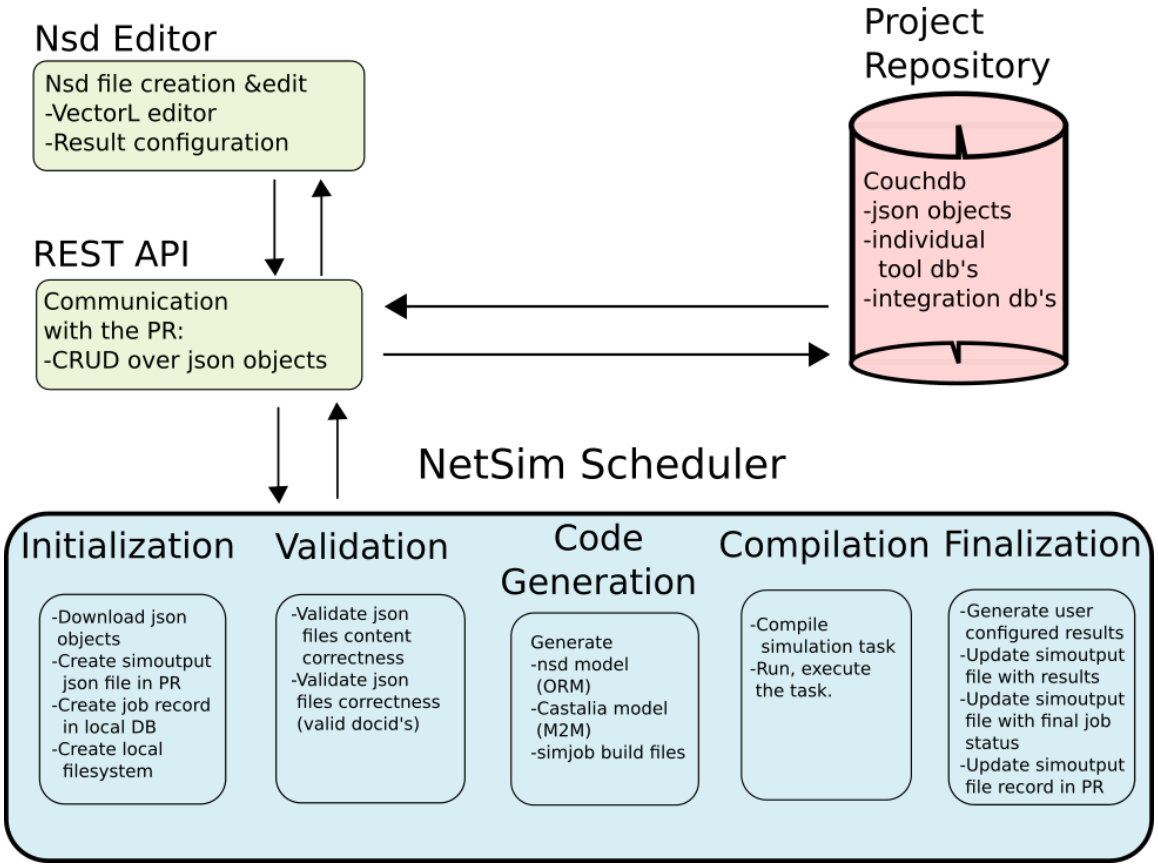

Figure 4.8: *NetSim overall architecture.*

Chapter 5

NetSim Implementation and Development Methodology

5.1 NetSim Model-oriented philosophy

As it was mentioned in previous chapters, NetSim was designed in a NSD model oriented philosophy. In this chapter we will describe the main reasons for which a model oriented design development has been selected. The advantages of a model oriented design and architecture will be shown, during 1) the initial development lifecycle of an application, 2) the integration process of the application modules and 3) the error correction process of the application.

5.2 Main aspects during NetSim development

During the development process of the NetSim service we had to confront and rapidly solve integration issues with the rest WSN-DPCM modules, which where designed and implemented from other development teams. So we had to select a flexible development philosophy, easily adjustable to continuous changes and specification redefinitions which occur during the development of an integrated software platform. For the above reasons during the NetSim development lifecycle we had as guideline and target to be fulfilled the following software requirements:

- **Rapid application development (RAD)**: The RAD approaches emphasize the necessity of adjusting requirements in reaction to knowledge gained as the project progresses. This

causes RAD to use prototypes in addition to or even sometimes in place of design specifications. RAD approaches also emphasize a flexible process that can adapt as the project evolves rather than rigorously defining specifications and plans correctly from the start so it offers rapid change capabilities during development process.

- **Software testability**: Is the degree to which a software artifact supports testing in a given test context. If the testability of the software artifact is high, then finding faults in the system (if it has any) by means of testing is easier. Testability is not an intrinsic property of a software artifact and can not be measured directly (such as software size). Instead testability is an extrinsic property which results from interdependency of the software to be tested and the test goals, test methods used, and test resources (i.e., the test context).

- **Software customization:** Is the effort to built software with high flexibility which is rich in features, which offer efficient and easy adaptation of different functionalities to the application.

Taking in mind the above requirements, we concluded that it would be efficient to design a model which integrates the parameters necessary to be defined and the actions necessary to be performed during a complete simulation execution. We gathered all the information necessary for a simulation execution (input, output, build and execution parameters) so as to define the corresponding model properties. Additionally we defined all the actions necessary to be performed during a simulation execution, and we translated them in methods over the model properties. In that way the NSD model was born. Also in the WSN-DPCM application integration level, we designed all the NetSim modules and overall functionality as actions over NSD model objects. This architecture philosophy fulfill the above requirements, and offers the ability to the developer to easily integrate specification redefinitions which constantly occur during the development lifecycle of an application.

5.3 NetSim Model Oriented Programming characteristics

In this section the NetSim NSD model oriented design of the integration process with the rest platform tools, will be explained. During the integration phase all the methods written and

developed, were configured to manipulate NSD and NSD related objects. This methodology offers NetSim high software customization abilities, since any attributes necessary to be added to the initial specification, are included to these objects, and any additional action necessary to be performed over these objects is transferred in a lower level of execution, separate from the integration that already exists. Since NetSim has already a functionality by which we can manipulate all the NSD related information (in the top level meaning NetSim definition), the further processing of this type of information is abstracted from the integration process with the rest platform tools. Furthermore the processing of NSD related information is shifted in a lower level of runtime, to object mapping and model transformation process, beyond retrieval and storage to the PR where the integration takes place.

5.3.1 REST API integration philosophy

All the methods of NetSim REST API (section 4.6) endpoint are designed so as to retrieve, store and update NSD json objects and NSD json related objects located in the PR. The CRUD methods over a CouchDB json object need as parameters the unique id and url of the corresponding object. Furthermore queries can be submitted to the CouchDB using an object id, so as to retrieve the object url or the complete json file. As far as it concerns the NSD related objects, their id's are included to the corresponding NSD object. So if we have the id of an NSD object we can retrieve using the REST API all the NSD related information. In case where the initial json objects schemes is necessary to change or to be enhanced with additional parameters and arguments, the design of the NetSim REST API still provides CRUD functionality over these objects, without any change. In case where additional NSD related objects need to be defined, CRUD functions over these objects need to be implemented and added to the REST API. So the REST API functionality can be easily enriched without dropping or transforming the existing methods. In general the REST API design offers NetSim high flexibility and customization abilities in addition to simplicity.

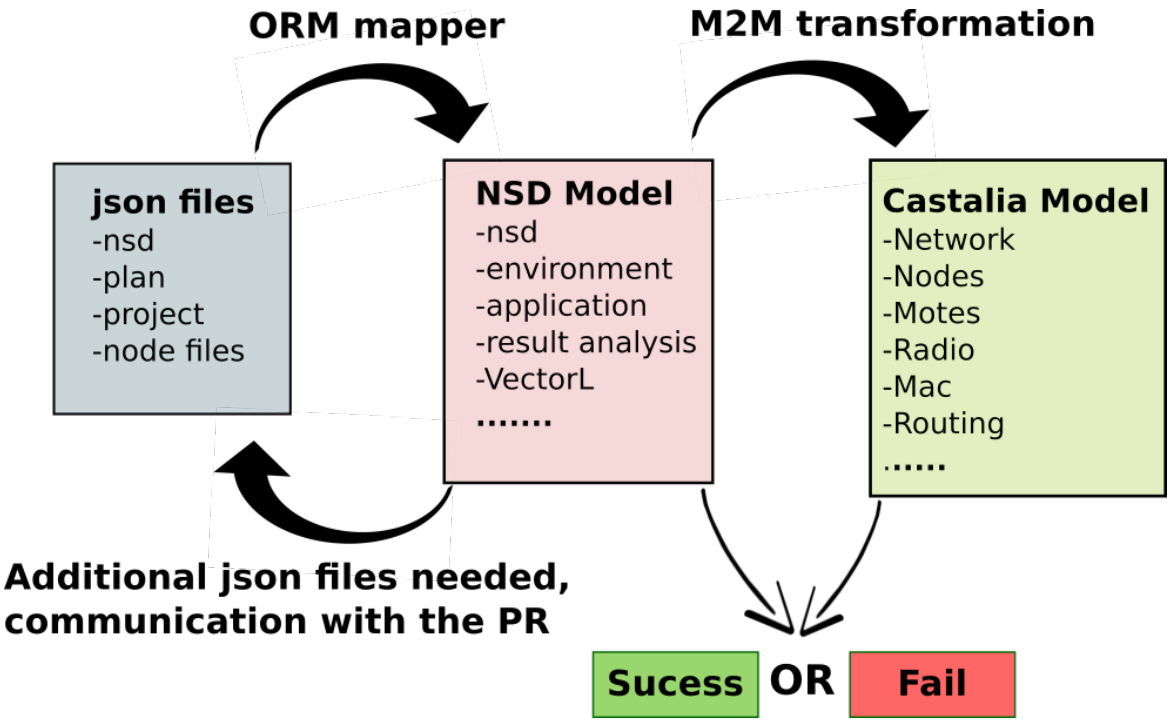

Figure 5.1: *Mapping and transformation flow.*

5.3.2 NetSim Model driven programming framework

In order to perform 1) the mapping process between json objects and NSD model objects and 2) the M2M transformation process from the initial NSD model to Castalia target model, NetSim has a simple model driven programming framework. A NSD model library has been built, which contains all the model definitions necessary for the json object to NSD model object mapping process, for the M2M transformation process and for communication with the PR. Communication with the PR might be necessary during the M2M transformation process besides the initial mapping process (from json objects to NSD Model objects).

The models implemented and included in the NSD model library are classified as follows:

- **Models necessary for the overall M2M Transformation process**, from the NSD initial model to Castalia target model. These models include objects which:

  1. are declared in the NSD model (initial model).
  2. are declared in Castalia description model (process model).
  3. are necessary for the M2M transformation process and perform the mapping be-

tween the initial and target model objects.

- **Models for the restful api with the PR**: These models include objects which represent the objects located in the PR and the definitions necessary for performing all the actions over the objects located in the PR.

Inside a model object definition, contained in the NSD model, the following type of attributes can be declared:

- **Properties**: The object parameters which can be either basic data types or strings.
- **Ref**: A reference to another model object.
- **Refs**: A list of references to a set of model objects.
- **Checked_constraints**: Enumeration attributes with specific constant values.

Using the above set of attributes, we can easily declare parameters inside model objects, and relationships between model objects.

The NetSim model driven framework gives the ability to create instances of the models included in the NSD model library using annotation, so NetSim dynamically generates all the model objects necessary for 1) the object relational mapping process between json objects and NSD model objects, 2) the M2M transformation process between initial NSD model and Castalia target model, during runtime. Additionally since the NetSim model driven framework supports dynamic mapping between json objects located in the PR and NSD model objects, NetSim has the functionality to retrieve any additional json object necessary for the M2M transformation during runtime.

Finally another feature of NetSim is that during the mapping process between json and model objects, creates properties of the object instances which are data encoded instead of generating hard code. For every property of a json object NetSim reads three parameters: name, type and value. Afterwards during the mapping process, creates inside the model object instance the corresponding property , with the specific name and value by generating dynamically the property type. **So during the mapping process, NetSim dynamically generates the data types of the model object properties**, characteristic which makes NetSim flexible and sufficient

to data types changes over object properties. So since the mapping process is data encoded can be easily reconfigured and enriched without major changes in the NetSim design and structure.

5.3.3 NetSim Project Repository query generator

During the design and execution of a simulation process there is need of continuous communication with the PR. The PR which was extensively described in section 3.4.2, is an online database consisted of json objects, where NetSim performs CRUD operations over objects related with the simulation design and execution lifecycle. The cases where NetSim needs to communicate with the PR are:

1. During the creation and manipulation of NSD objects via the NSD Editor.
2. During the mapping of json objects to NSD model objects.
3. During the M2M transformation from the NSD model to the Castalia model (there is the ability for communication at this stage, but no communication is performed for the current NetSim implementation).

The CRUD operations over the PR ,are performed as JavaScript queries over the CouchDB. The queries that NetSim needs to perform over the PR during the NSD design and during runtime are part of the code generation NetSim functionality, and are implemented by the NetSim query generator. This means that **the JavaScript queries are not hard coded, instead are generated and submitted to the PR at runtime**, according to the type of json objects and the actions over these objects that NetSim needs to perform at each different runtime phase.

The NetSim query generator feature, saves a lot of effort in comparison with the case where the queries would have been developed in a hard coded manner. Additionally offers to the tool great flexibility and customization, as far as it concerns the range of queries that can be performed to the PR. Furthermore having this feature NetSim can easily adopt any new functionality related with CRUD operations over the CouchDB.

```
@lru_cache(10)
@docstring_template
def gen_object_synopsis(entity, var):
    '''
    {
        _id : {{var}}._id
        % for a in entity.fields:
           , {{a.name}} : {{var}}.{{a.name}}
        % end
    }
    '''
    return locals()


@docstring_template
def gen_map_func(entity, key, value=gen_object_synopsis):
    '''
    function (doc) {
        if(doc.type && doc.type=='{{entity.name}}' {{! "&& doc.%s" % key.name if not key.is_key() else ""}})
            emit(doc.{{key.name}}, {{value(entity, 'doc')}} );
    }
    '''
    return locals()


MODELS = [CouchDesign(e) for e in ENTITIES if isinstance(e,ApiEntity)]
```

Figure 5.2: *Query generator code. The generator is written in Python and generates JavaScript queries, which are submitted to the PR during runtime.*

5.4 NetSim tests during development

Finally during the development of each individual NetSim module and execution phase, python test scripts were written so as to check the corresponding module's proper and correct operation 1) individually, and 2) in integration with the rest application. This technique offers great easiness to the developers since it makes clear and straightforward the error correction and debugging process, especially when developing an integrated software with complex functionality and numerous modules.

Chapter 6

NetSim Usability and Performance

NetSim is a major component of the WSN-DPCM platform, whose operation relates to a large part of the whole platform. In this section we present the evaluation and validation results obtained for the developed system, in terms of functionality, scalability and productivity. Some of our results are of a qualitative nature, collecting and organizing qualitative assessments made during development, testing and usage of NetSim in the development of the demonstrators. Quantitative results are also reported, together with a description of the methodology used to collect them.

We report results on three aspects of functionality, namely correctness, efficiency and usability.

6.1 NetSim Traceability

Network simulations are complex computations incorporating a very large number of details into the simulated model. Simulations can fail (e.g., because of bugs that cause the simulated system crash). Also, some user mistake (e.g., not saving a change to the simulation model) may lead to unexpected, misleading results.

To assist the user in such occasions, a number of functionalities were built:

1. Model validation functions. The network simulator integrates most of the information contained in the project repository. Inconsistencies in that information may not be detected in other tools (e.g., the planning tool) but may nevertheless cause simulations to

behave incorrectly. To alleviate this situation, we implemented mode extensive consistency checks on the models imported from the project repository, issuing both warnings and errors.

2. We implemented the ability to execute VectorL code outside a simulation, so that the behavior of VectorL programs can be observed in isolation.

3. We have implemented an additional user interface (inside the Admin GUI of NetSim) for simulation introspection. In this way, an advanced user or a knowledgeable system administrator can inspect the internal data (disk files generated by NetSim, the compiler, etc) of particular simulations and help the user locate difficult bugs.

6.2 NetSim Consistency

To assure consistency in the executed simulations, we have implemented NetSim simulations to be pseudo-deterministic, that is, deterministic up to the choice of random seed. Thus, repeating the same simulation without changes in the input model, generates identical results.

Pseudo-determinism was achieved by the following two features:

- given two logically equivalent input models, the simulation generator generates identical code, and

- the generated simulation code is (pseudo-)deterministic. Feature (a) was validated by tests. Feature (b) was validated by inspection of the generated code in test cases and by relying on similar features of the OmNet++ simulator.

It is worth noting that the user may set the random seed manually, or allow for different random seeds to be used per simulation run, so that both deterministic and non-deterministic behavior is available.

6.3 NetSim Runtime Efficiency

There are two aspects to efficiency in NetSim:

- the efficiency of the NetSim server itself
- the efficiency of the generated simulations.

6.3.1 Efficiency of the generated simulations

The runtime efficiency of generated simulations is not easily quantifiable, since there is no benchmark to compare against. We assessed it by the following manual testing method: for a number of test cases, after the NetSim code generator produced a simulation source code, we changed the generated files by hand, striving to achieve greater performance. Then, we compared the runtime of the original generated simulation to the modified. In every case, the runtimes of the two simulations were almost equal (less than 5% difference in all cases). We conclude that generated simulations are efficient in terms of generator quality.

In order to assess the runtime scalability of simulations with respect to model size, we conducted a series of experiments with models of WSN varying the number of sensors.

There are two simulation parameters that affect simulation runtime:

- the total simulation time, and
- the number of wireless sensors in the model.

In terms of simulation time, the performance of the generated simulations was completely linear; this behavior reflects the underlying implementations of OmNet++ and Castalia, therefore we do not analyze it further.

Figure 6.1 depicts the mean runtime of a simulation, while the number of wireless sensors in the network increases. Here must be noted that the simulations performed for the above experiments have the same configuration and arguments, including simulation time, number of measurements recorded and plotted, the communication protocols between nodes, the mote types etc.

It is observed that the mean simulation runtime is related with to the network size. This is totally reasonable thus the simulator needs to read for each simulation the project, plan and node definition JSON files from the project repository. The node definition JSON files which are located in the node library in the project repository include all the information related with the

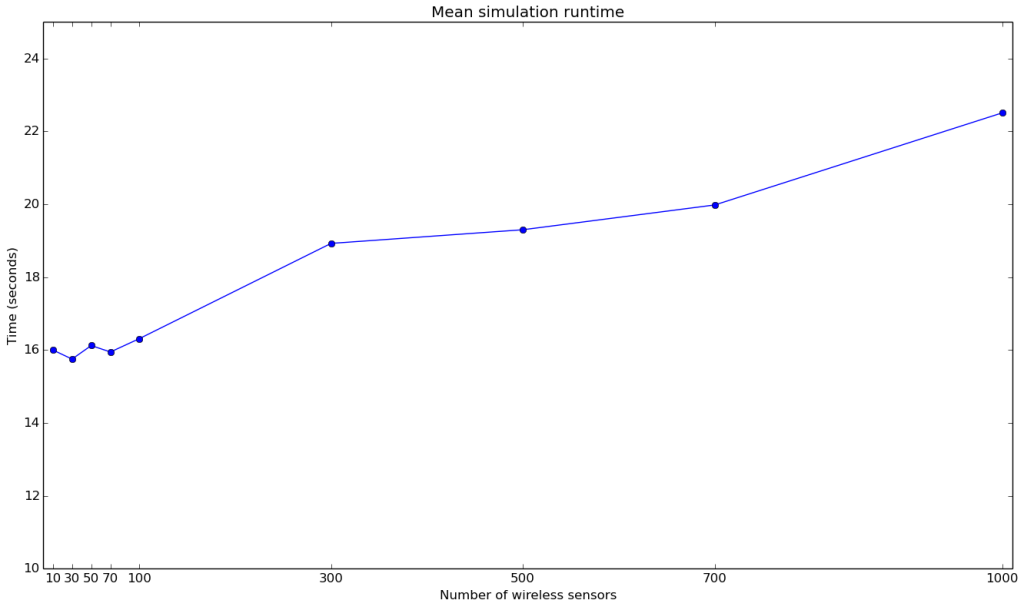

Figure 6.1: *Mean simulation runtime for different numbers of wireless nodes in the model.*

application logic, which is necessary for initializing and running a simulation. As the number of node increases the number of JSON files that need to be downloaded increases, thus the simulation initialization phase duration increases. Furthermore during the code generation phase, while the number of nodes in the network increases, more arguments related with the nodes needs to be initialized and configured, tasks which can deteriorate the simulation execution time.

Despite the fact that the simulation runtime increases as the number of wireless sensors increases, we can say that does not increase dramatically so as to deteriorate system's performance. The increase is not completely proportional to network size and it follows smooth with high regularity behavior.

6.3.2 NetSim Server Storage Efficiency

Storage efficiency of the generated simulations was evaluated by measuring the amount of RAM and disk space used per simulation. The amount of RAM consumed per simulation is quite modest (less than 10 Mbytes). The amount of disk space used to store the simulation's files is in the order of 1 Mbyte per simulation, plus an overhead linear to the simulation time, when trace

statistics are collected.

6.4 NetSim Usability

NetSim is offered both as a service, accessible via the WebTop platform and as a standalone program, that the user can run on a personal workstation, running Linux. The standalone mode is particularly useful in combination with the Hardware-in-the-Loop functionality. In standalone mode, the system is invoked via a command-line interface. We did not conduct a formal usability study. However, as we made NetSim available to other partners during code development and the early stages of demonstrator development, we collected usability improvement remarks and suggestions that we incorporated into the final deliverable. Hardware in the Loop functionality can be invoked either from the standalone program or WebTop platform. Of course the actual execution of the specific scenario must be executed locally since it must in the processing environment (e.g. PC or Server) where the actual physical nodes are connected and controlled.

6.5 NetSim Scalability

The NetSim service is invoked via a RESTful API. In order to assess the efficiency of the NetSim server itself, we performed a scalability study measuring the response time and the throughput of RESTful API requests, as the number of concurrent users increases. Experiments were conducted on a NetSim server running on a rather limited virtual machine, with 512 Mbytes of RAM and only 1 (virtual) core. To generate the HTTP requests and measure the results, we used the Apache Benchmark utility, which is provided as part of the distribution of the Apache web server.

The throughput of this setup is depicted in Fig. 6.2 above. It can be seen that, even on limited (virtual) hardware, the NetSim server achieves a very significant throughput rate, of about 350 requests per sec. Also, varying the number of users does not reduce throughput, which indicates that there are no significant congestion points in the implementation.

Figure 6.3 depicts the simulator's response time when concurrent requests are performed in the simulator server. More accurately we see the simulator's response time, when the create

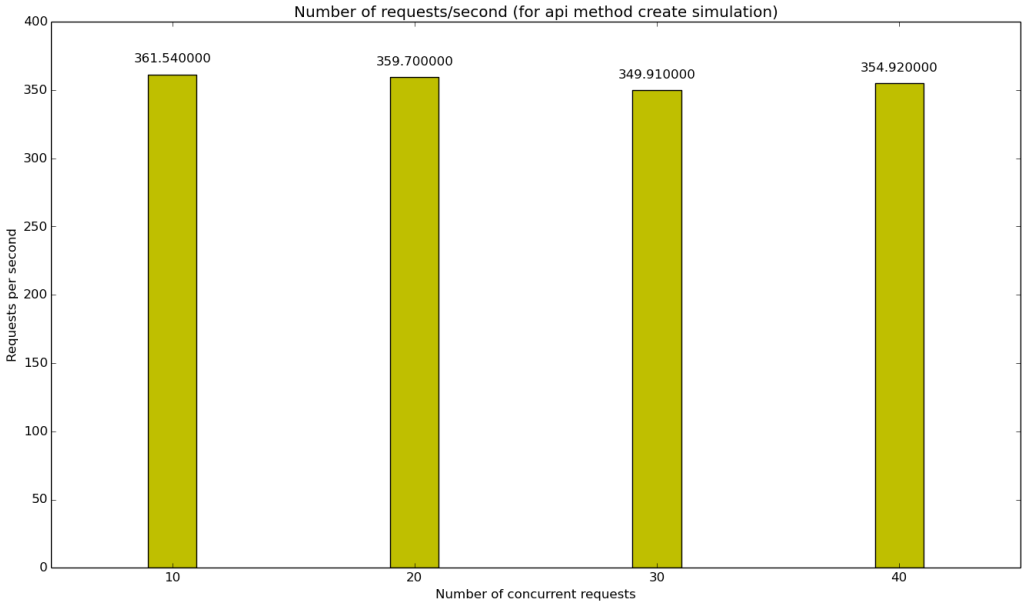

Figure 6.2: *Number of requests served per second. The numbers shown correspond to the create_simulation API call.*

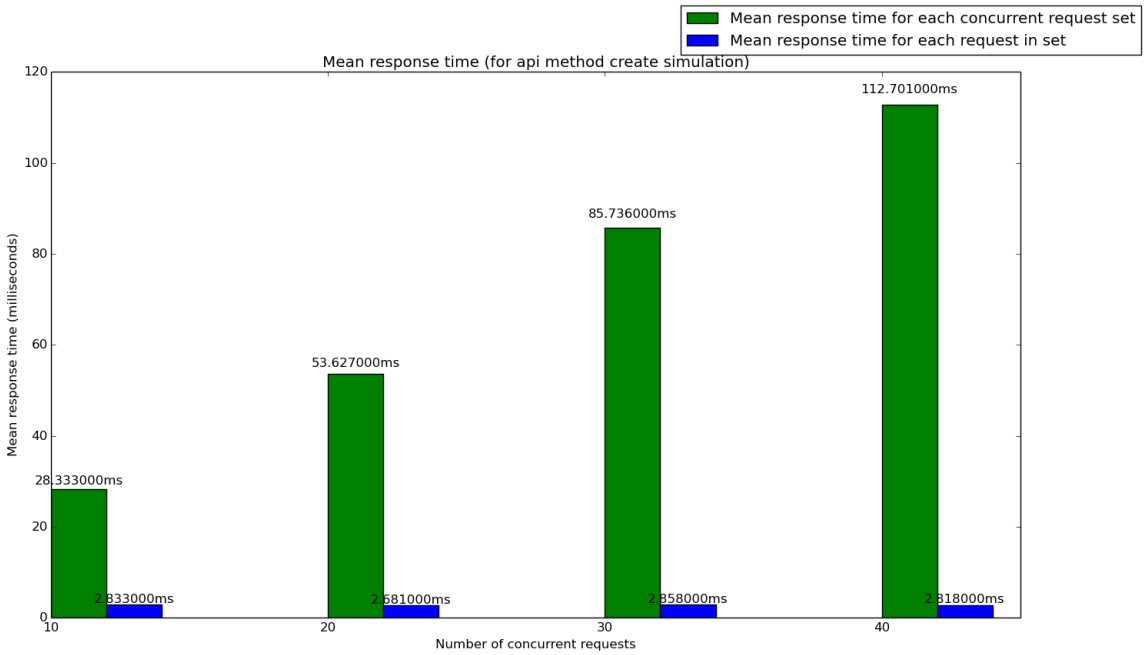

Figure 6.3: *Response time for create_simulation concurrent requests, as the response time increases.*

simulation method is called. This method creates, submits and finally executes in the simulator a simulation, communicates with the remote project repository in order to download and upload all the relevant with the current simulation JSON files, thus is a method with significant load. Each request in the figure can represent a single user. It is observed that the simulator server can serve multiple users simultaneously, without increasing the system's response time. Although the total service time of concurrent users is proportional to the number of concurrent users. It can also be observed that the number of requests which can be served per second is not affected by the number of concurrent requests. This means that as the number of simultaneous users increases, the simulator server can serve the users without any delay.

Overall, it is concluded that the simulator has scalable performance with respect to the number of concurrent users.

6.6 Productivity Gains

A quantification of productivity gains from the use of NetSim is challenging, both because there is no established benchmark and because productivity is hard to measure without a statistically significant number of experiments. However, some indirect quantitative measurement can be provided, if the effort of creating a simulation by use of NetSim is contrasted to the effort of creating the same simulation for Castalia/OmNet++, by hand. Under this choice of benchmark, a quasi-quantitative estimation can be reasonably made. A typical simulation generation by NetSim generates a number of files from the contents of the Project Repository, whose size comes close to the order of 1000 lines of code (loc).

For example, a simple simulation for one of the platform's demonstrator implemented, generates the following files (presided by lines of code per file):

133 app1.cc 94 app1.h 24 app1_packets.msg 186 environ.cc 43 environ.h 27 executor.mak 1 nodemap.json 322 omnetpp.ini 8 ParkingCollector.ned 8 ParkingForward.ned 7 ParkingSensors.ned

These files sum up to 852 LOC. Empirically, this amount of code requires in the order of 10 hours to generate, even for very experienced users of Castalia. Using the NetSim generator, the time to compose the required NSD, VectorL modules and application logic is quite small, for

a moderately experienced user, in the order of one hour. This estimate can be broken up as follows:

- 5 min to generate the NSD (including defining output plots)
- 15 min to code and debug an environment model using VectorL
- 45 min to code the application logic modules (assuming a simple application)

Overall, this very rough estimate implies productivity gains up to an order of magnitude. Similar gains should be obtained vs. the case of creating a simulation by hand, when the simulated model is changed (e.g., by the planning tool). Using NetSim, this cost can be as little as re-running an NSD (a few key presses) whereas, even small changes may take several minutes to update the simulation code by hand.

Furthermore, the extensive model validation performed by NetSim is likely to improve significantly the correctness of a simulation generated, vs. one created by hand. Additionally the actions mentioned above are made via the NetSim NSD Editor and the NetSim Admin GUI. This means that any user configuration takes place via a user friendly environment, instead of plain source and configuration file manipulation which the Castalia simulator demands. Finally, the implementation of NetSim as a service entails significant productivity gains when several developers collaborate in a project, as sharing between developers becomes straightforward.

Appendix A

Acronyms

NSD Network simulation descriptor

PR Project Repository

PT Planning Tool

C&MT Commissioning & Maintenance Tool

DT Development Tool

HiL Hardware in the Loop

MDD Model Driven Development

MDA Model Driven Architecture

ORM Object Relational Mapping

MOP Model Oriented Programming

EMF Eclipse Modeling Framework

RAD Rapid Application Development

Bibliography

[1] Atemu simulator official website. http://www.hynet.umd.edu/research/atemu/, 2015. [Online, Accessed 9-December-2015].

[2] Avrora simulator official website. http://compilers.cs.ucla.edu/avrora/, 2015. [Online, Accessed 9-December-2015].

[3] Apache couchdb official website. http://couchdb.apache.org/, 2015. [Online, Accessed 9-December-2015].

[4] J-sim simulator official website. https://sites.google.com/site/jsimofficial/, 2015. [Online, Accessed 9-December-2015].

[5] Mixim official website. http://mixim.sourceforge.net/, 2015. [Online, Accessed 9-December-2015].

[6] a webpage introduced tossim. http://docs.tinyos.net/index.php/TOSSIM, 2015. [Online, Accessed 9-December-2015].

[7] Castalia simulator official website. https://castalia.forge.nicta.com.au/index.php/en/, 2015. [Online, Accessed 9-December-2015].

[8] Contiki official webpage. http://www.contiki-os.org/, 2015. [Online, Accessed 9-December-2015].

[9] Cooja simulator official webpage. http://anrg.usc.edu/contiki/index.php/Cooja_Simulator, 2015. [Online, Accessed 9-December-2015].

[10] Eclipse modeling framework official webpage. https://eclipse.org/modeling/emf/, 2015. [Online, Accessed 9-December-2015].

[11] Mica2 specification website. http://www.capsil.org/capsilwiki/index.php/MICA2, 2015. [Online, Accessed 9-December-2015].

[12] Ns-2 simulator online documentation. http://www.isi.edu/nsnam/ns/ns-documentation.html, 2015. [Online, Accessed 9-December-2015].

[13] Ns-2 simulator official website. http://www.isi.edu/nsnam/ns/, 2015. [Online, Accessed 9-December-2015].

[14] Ns-3 simulator official website. https://www.nsnam.org/, 2015. [Online, Accessed 9-December-2015].

[15] Omnet++ simulation framework official website. https://omnetpp.org/, 2015. [Online, Accessed 9-December-2015].

[16] Powertossim official webpage. http://www.eecs.harvard.edu/~shnayder/ptossim/, 2015. [Online, Accessed 9-December-2015].

[17] Tinyos official webpage. http://www.tinyos.net/, 2015. [Online, Accessed 9-December-2015].

[18] A. Boulis. Castalia. a simulator for wireless sensor networks and body area networks. user's manual. https://www.google.gr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwjY6ca7_s_JAhVFPBoKHebSC1cQFggeMAA&url=https%3A%2F%2Fforge.nicta.com.au%2Fdocman%2Fview.php%2F301%2F592%2FCastalia%2B-%2BUser%2BManual.pdf&usg=AFQjCNHEDczbrtB1rwiW8VlYFZt88gw4sw&sig2=eJY8Gojfee8G17RXbTqO4g, 2011. [Online, Accessed 9-December-2015].

[19] E. Egea-Lopez, J. Vales-Alonso, A. S. Martinez-Sala, P. Pavon-Marino, and J. García-Haro. Simulation tools for wireless sensor networks. *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS05)*, page 24, 2005.

[20] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. Emstar: An environment for developing wireless embedded systems software, 2003.

[21] L. M. Feeney and D. Willkomm. Energy framework: An extensible framework for simulating battery consumption in wireless networks. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 20:1–20:4, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-87-5. doi: 10.4108/ICST.SIMUTOOLS2010.8725. URL http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8725.

[22] L. Girod, J. Elson, T. Stathopoulos, M. Lukac, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *In Proceedings of the 2004 USENIX Technical Conference*, pages 283–296, 2004.

[23] M. Imran, A. Said, and H. Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *Information Technology (ITSim), 2010 International Symposium in*, volume 2, pages 897–902, June 2010. doi: 10.1109/ITSIM.2010.5561571.

[24] D. J. Polley, Blazakis, J. McGee, D. Rusk, and J. Baras. Atemu: a fine-grained sensor network simulator. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 145–152, Oct 2004. doi: 10.1109/SAHCN.2004.1381912.

[25] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling by Raj Jain*. John Wiley & Sons, ISBN:0471503363, 1991.

[26] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08,

pages 71:1–71:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2. URL http://dl.acm.org/citation.cfm?id=1416222.1416302.

[27] J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus. Comparative study of wireless network simulators. In *Networking, 2008. ICN 2008. Seventh International Conference on*, pages 517–523, April 2008. doi: 10.1109/ICN.2008.97.

[28] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM. ISBN 1-58113-707-9. doi: 10.1145/958491.958506. URL http://doi.acm.org/10.1145/958491.958506.

[29] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648, Nov 2006. doi: 10.1109/LCN.2006.322172.

[30] S. Park, A. Savvides, and M. B. Srivastava. Sensorsim: A simulation framework for sensor networks. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '00, pages 104–111, New York, NY, USA, 2000. ACM. ISBN 1-58113-304-9. doi: 10.1145/346855.346870. URL http://doi.acm.org/10.1145/346855.346870.

[31] D. Pediaditakis, Y. Tselishchev, and A. Boulis. Performance and scalability evaluation of the castalia wireless sensor network simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 53. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[32] L. Shu, C. Wu, Y. Zhang, J. Chen, L. Wang, and M. Hauswirth. Nettopo: Beyond simulator and visualizer for wireless sensor networks. In *Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on*, volume 1, pages 17–20, Dec 2008. doi: 10.1109/FGCN.2008.18.

[33] S. Sinha, Z. Chaczko, and R. Klempous. Sniper: A wireless sensor network simulator. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2009*, volume 5717 of *Lecture Notes in Computer Science*, pages 913–920. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04771-8. doi: 10.1007/978-3-642-04772-5_117. URL http://dx.doi.org/10.1007/978-3-642-04772-5_117.

[34] A. Sobeih, J. Hou, L.-C. Kung, N. Li, H. Zhang, W.-P. Chen, H. ying Tyan, and H. Lim. J-sim: a simulation and emulation environment for wireless sensor networks. *Wireless Communications, IEEE*, 13(4):104–119, Aug 2006. ISSN 1536-1284. doi: 10.1109/MWC.2006.1678171.

[35] M. Stehlik. Comparison of simulators for wireless sensor networks. http://is.muni.cz/th/172616/fi_m/master_thesis.pdf, 2015. [Online, Accessed 9-December-2015].

[36] C. Stevens, C. Lyons, R. Hendrych, R. S. Carbajo, M. Huggard, and C. M. Goldrick. Simulating mobility in wsns: Bridging the gap between ns-2 and tossim 2.x. In *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '09, pages 247–250, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3868-6. doi: 10.1109/DS-RT.2009.22. URL http://dx.doi.org/10.1109/DS-RT.2009.22.

[37] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2. URL http://dl.acm.org/citation.cfm?id=1416222.1416290.