SensGrid: Modeling and simulation for wireless sensor grids

Article *in* SIMULATION: Transactions of The Society for Modeling and Simulation International · August 2012

DOI: 10.1177/0037549711434180

4 authors:

Raul Moreno
University of Castilla-La Mancha
1 PUBLICATION **2** CITATIONS
SEE PROFILE

Antonio Robles-Gómez
National Distance Education University
69 PUBLICATIONS **178** CITATIONS
SEE PROFILE

Aurelio Bermúdez
University of Castilla-La Mancha
49 PUBLICATIONS **297** CITATIONS
SEE PROFILE

Rafael Casado
University of Castilla-La Mancha
52 PUBLICATIONS **304** CITATIONS
SEE PROFILE

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
88(8) 972–987
© 2012 The Society for Modeling and Simulation International
DOI: 10.1177/0037549711434180
sim.sagepub.com

SensGrid: modeling and simulation for wireless sensor grids

Raúl Moreno[1], Antonio Robles-Gómez[2], Aurelio Bermúdez[1], and Rafael Casado[1]

Abstract
Grid computing technologies are emerging as the latest generation of distributed computing, as they allow the aggregation of resources that are geographically distributed over different locations. Since researchers need to conduct repeatable and controlled experiments, it is easier to use simulation as a means of studying complex scenarios. In this area, the GridSim simulation toolkit is a widely accepted option. We have designed and implemented a simulator of systems that integrates sensors and a grid. In particular, an extension of GridSim has been created in order to enable the execution of simulations in a way the users can perform queries on sensor networks. In this paper, we present the features of the system to be simulated, the modeling proposed, and the high-level user interfaces of the simulator. We also describe an experiment that demonstrates how our proposal works, which may well be helpful for the research community.

Keywords
simulation, modeling, GridSim, wireless sensor networks, grid systems

1 Introduction

Grid systems represent one of the latest advances in distributed computing. In such systems, a set of geographically distributed computational and storage resources are cooperatively used to act as a single powerful computer.[1] From the end-user point of view, grids can be used to provide computational services (execution of jobs on distributed computational resources), data services (access and management of distributed datasets), application services (access to remote software and libraries), information services (extraction and representation of data with meaning), and knowledge services (building new knowledge from the stored information). However, grids have a number of open issues, such as the job scheduling policy and resource allocation.

Owing to the high cost of accessing a real system where experiments can be executed, many researchers use simulation tools in order to verify the performance of their proposals. In this scenario the tools used for the simulation of grids become very important: one of them is GridSim Toolkit,[2] a programming library implemented in Java. GridSim has several advantages over other grid simulation tools. Firstly, the use of Java allows its use over any computer platform and facilitates the development of extensions. In addition, GridSim allows an intuitive design for them, in contrast to other simulators with a more complicated structure (e.g. SimGrid). Finally, a non-negligible advantage is that GridSim is completely open source.

In addition to this, wireless sensor networks (WSNs)[3] are one of the most promising technologies to emerge in the last few years. Recent advances in hardware have made possible the development of small low-cost devices with low power consumption that are suitable for remote sensing applications. A sensor network consists of a base station and a large number of intelligent sensor nodes – often referred to as 'motes' – which are densely deployed over a particular area of interest. These motes are capable of sensing, storing, processing, and communicating a large amount of physical information collected from the environment. To do this, each mote is composed of one or more sensors of analog values, such as temperature, humidity, etc., an embedded processor, a memory, a battery, and some wireless communication hardware (usually a radio

[1]Departamento de Sistemas Informáticos, Instituto de Investigación en Informática Albacete (I^3A), Universidad de Castilla-La Mancha, Spain
[2]Departamento de Sistemas de Comunicación y Control, ETSI Informática, Universidad Nacional de Educación a Distancia, Spain

Corresponding author:
Raúl Moreno, Universidad de Castilla-La Mancha, Instituto de Investigación en Informática de Albacete, Campus Universitario s/n, Albacete 02071, Spain.
Email: raulmoreno@dsi.uclm.es

frequency transceiver). Common applications for sensor networks are environmental monitoring, high-precision agriculture, smart buildings, and medical research. When sensor nodes incorporate actuators, the sensor network is also capable of controlling the environment.

The integration of one or more WSNs in a grid system is inherently attractive; a wide range of possibilities arises, since the system is provided with ‘senses’. These types of systems are capable of combining, in a single platform, a huge amount of real-time data from different sources, and about a wide area, with the vast computational and storage power provided by the grid architecture. Another advantage of this integration is that sensors can be shared transparently by multiple users and applications. It is worth mentioning the possibility of employing the power of grid computing to implement decisions concerning to the environment, that is, to control the environment, by using sophisticated artificial intelligence algorithms.

There are many examples of the application of these systems. In De Roure,[4] a disaster forecasting system is proposed. In Sanabria et al.,[5] the acoustic signals collected are being used to monitor the behavior of amphibians, such as the endemic Puerto Rican Crested Toad, which is in danger of extinction. In Aqeel-ur-Rehman et al.,[6] the authors show that this technology is well suited for addressing different problems in the field of agriculture. Finally, in Pallikonda-Rajasekaran et al.,[7] the authors propose an architecture for monitoring the health status of different groups of patients, to facilitate analysis, diagnosis, prognosis, and drug delivery. Creating a grid of the required characteristics, in the literature often called a sensor-grid or WSG (wireless sensor grid),[8,9] may require a significant financial investment. For this reason, prior to setting up a real system, it is highly desirable to check its behavior by means of simulation.

The analysis of WSGs performed by using a general network simulator could be a daunting task, as the user must model all the entities starting from scratch. Moreover, to our knowledge, current off-the-shelf WSN simulators do not provide native grid support, or vice versa. Fortunately, some simulation tools are easily extensible, providing a solution for integrating both technologies. In this work, we present SensGrid, an extension of GridSim to support WSGs. The main contribution of this work is our answers to the following technical challenges:

- to incorporate a resource within GridSim that consists of a sensor node with wireless communication capability, location awareness, and self-configurable routing, which is powered by a limited battery;
- to add to GridSim a resource consisting of a base station that is able to interconnect the WSN with the grid system;
- to implement a manager service that handles the available sensor networks, translating abstract user queries into data requests to specific sensors (according to their location), and managing the information gathered before returning it to the user;
- to provide GridSim with a two-dimensional (2D) continuous model for physical magnitudes;
- the definition of a set of grid commands to enable the user to interact with the WSN resources;
- the development of a friendly visual interface to define and show WSN deployments and physical magnitudes, to execute WSN commands, and to represent the corresponding results graphically.

The remainder of this paper is organized as follows: Section 2 discusses some related works that deal with WSGs and GridSim extensions; Section 3 explains some basic characteristics of GridSim, and the main features of our extension are presented in Section 4; the user interfaces that we developed for this extension are presented in Section 5; an experiment we performed that implements the proposal is described in Section 6; and finally, Section 7 presents our conclusions and future work.

2 Related work

To the best of our knowledge, no other proposals that aim to simulate grids that integrate WSNs have been made. Nonetheless, several efforts have been made regarding real WSG systems. Several GridSim extensions regarding matters other than WSN support have been developed too. In the following sections illustrative works of these two types are described.

2.1 Integration of WSNs and grids

CIMA (Common Instrument Middleware Architecture)[10] is probably the first work to tackle the integration of WSNs and grids. It consists of a middleware to integrate sensors into grids and is based on the OGSA (Open Grid Services Architecture) standard. It considers each sensor as an individual resource of the grid.

Other early works are SPRING (Scalable Proxy-based aRchItecture for seNsor Grid)[11] and Hourglass.[8] In the first one the most outstanding feature is the use of a proxy as an interface between the WSN and the grid. The second one is based on data collection and its authors show several interesting applications, such as health care, supply management, and assistance in Mass Casualty Incidents (e.g. terrorist attacks or train accidents).

OSWA (Open Sensor Web Architecture)[12] is focused on SOA (Service-oriented Architecture) and implements the SWE (Sensor Web Enablement) standard,[13] which specifies a set of interfaces and encodings that may be useful to facilitate the interoperability of the WSG systems. In Kobialka et al.,[14] a system mounted on an improved version of the OSWA middleware is described. The sensing devices of the system make use of SunSpot technology (Sun Small Programmable Object Technology) and the application is related to gesture recognition.

In the context of the CrisisGrid project,[15] an architecture based on SOA is also proposed. It focuses particularly on applications that make use of geographical information systems; these systems are part of the grid and they can access both historical data (the traditional way) and real-time data through the WSN. QuakeSim,[16] a NASA-funded project that uses information obtained by sensors in conjunction with data mining techniques implemented in a grid, was developed by taking the CrisisGrid architecture as a basis. QuakeSim's aim is to estimate the risk of earthquakes, seaquakes, and volcanic eruptions.

TinySOA[17] has also been developed following the SOA principles. Specifically, it is a framework that allows the grid to interact with WSNs through services. Its goal is to provide an application programming interface (API) to manage the grid sensors independently of the programming language employed by the user. There exists a TinySOA implementation written in Java, called TinyVisor.[17]

Also of interest is an approach based on the collaborative WSG idea.[18] Its authors propose the CSG (Collaborative Sensor Grid) middleware, which includes a module that is responsible for aggregating different WSGs.

Westfälische Wilhelms-Universität of Münster has recently carried out the project SoKNOS (Service-oriented architeKtures supporting Networks Of public Security).[19] This project implements the SWE standard through a WSG architecture, which is especially suitable for applications that are related to disaster management. Another recent project is ASGrid,[20] whose most interesting features are the capabilities of self-configuration, self-optimization, and self-protection.

Much of the work done, in terms of the integration of WSNs and grids, is addressed at building application-specific architectures instead of generic WSG architectures. Some examples include those designed for the monitoring of patients,[7] those designed for traffic management,[21] and those designed for surveillance.[22]

Finally, a review of the most important research issues concerning WSG systems can be found in Tham,[23] as well as two examples of applications for such systems: one based on distributed information fusion and the other based on distributed autonomous decision making. Other surveys about WSGs are presented in Ahuja and Myers[24] and Gang et al.[25]

2.2 GridSim extensions

A certain number of extensions for GridSim have been developed for different purposes. Some of the most important ones are outlined below.

A core extension[26] for making simulations of grids, which not only executes computational jobs but can manage file replicas too (data grids), was developed by the GridSim creators team. This team has also implemented an extension[27] to support advance resource allocation in simulations and another one[28] to provide a graphical interface for GridSim.

Poznan University of Technology, in Poland, has created GSSIM (Grid Scheduling SIMulator),[29] a simulator built on top of GridSim. It enables experimental studies of various scheduling algorithms by simulating computational grids with real or synthetic workloads.

Several extensions that are focused on scheduling have been created. In the Czech Republic, Masaryk University has developed Alea,[30] another simulator that enables the study of scheduling algorithms. It is also built on top of GridSim and is used to design and test complex scheduling algorithms for both static and dynamic grid scenarios. Recently, Alea II[31] has been developed.

Andong National University, in South Korea, has developed a web-based grid scheduling platform for GridSim. It allows users to test scheduling algorithms in a web environment. In this way, some technical complexities of GridSim are hidden from users.

A GridSim extension that enables an architecture for failure detection can be found in Caminero et al.[32] Its authors have also developed another extension[33] that introduces network buffers management policies. This one is aimed at making network QoS (Quality of Service), an integral part of job scheduling decisions in grids, and it includes both a resource scheduler and admission control.

In Dias de Assuncao and Buyya[34] a framework for GridSim is presented that includes auction protocols in the resource allocation of the simulated grids. In particular, this framework implements four types of protocols: First-price Sealed, English auction, Dutch auction, and Continuous Double auctions.

In Italy, the University of Catania has designed an extended library called GAP (Grid Agent Platform),[35] which is targeted at providing multimedia contents based on QoS requirements in simulated grids. GAP aims at minimizing the network delay when transferring multimedia contents to a grid resource.

Finally, some techniques to incorporate differentiated levels of service and background traffic into GridSim are presented by Sulistio et al.[36]

3 GridSim toolkit

GridSim extends the functionality of the SimJava package,[37] which offers classes (in the context of OOP – Object-oriented Programming) to implement general simulations based on discrete events.

A simulation entity is a special type of object (in the context of OOP) characterized by having its own thread and the ability to send and receive events. In a GridSim simulation, at least, an entity for the user, one or more entities for the resources, and an entity for the GIS (Grid Information Service; this meaning is in the context of GridSim, not to be confused with 'Geographical Information Service', a meaning often used with the same acronym) must be created. For each user entity, entities must be established for input and output buffers, respectively; it is advisable also to create these entities for the buffers of every resource. An entity associated to the user does not simulate the behavior of the human user: it refers to the process running on his/her computer. Sometimes, a separate entity is created for the job scheduler, referred to as a *broker*. Anyway, resources need to be registered in a GIS, which is responsible for maintaining a list of them. It is not mandatory for there to be a single GIS in the simulation; so, the possibility exists of defining some regional GISs (one for each wireless area network (WAN) or organization). The user entity, or, where appropriate, the broker entity, must fragment the initial job launched to the grid and, afterwards, it must assign resources for every fragment. Each fragment is represented by an object called a *gridlet*. The entities that simulate the buffers are the ones that are mainly responsible for simulating the delay caused by the network.

Any entity created in SimJava has communication with all other entities. Nevertheless, GridSim offers the option of establishing topologies and so limiting the entities with which an entity can communicate; the way to achieve this is to define every link and to bind it to two entities: source and destination. The links are also entities. An entity associated to a user or resource can only be bound to a single link. If it is necessary to have more links bound to the entity, this one will necessarily represent a router and not a user/resource.

The typical simulation sequence is as follows. Every resource sends a registration message to a GIS. Afterwards, the main application submits a job to the grid. The user (or broker) entity splits the job into gridlets and requests the list of registered resources from the GIS entity. Once the list is obtained, the user schedules the order in which gridlets will be sent and assigns them to the resources. Next, it sends each gridlet to the corresponding resource entity. Resource entities estimate the execution time of the gridlets and, at the right moment, they send messages to the user entity in order to notify the completion of the gridlets.

4 The SensGrid extension

Our extension of GridSim, named SensGrid, allows the creation of grid simulations in which a user obtains certain services related to WSNs, that is, allows the creation of simulations of WSGs. Its source code can be freely downloaded from http://i3a.uclm.es.

In the system modeled the user launches complex queries to the grid concerning the magnitudes sensed and, then, he/she receives the corresponding results without knowing which sensors have participated. For instance, the user wishes to obtain the maximum temperature of a particular geographical region but does not want or need to know which sensor/s has/have sensed the data employed for calculating the final information. Note that temperature monitoring in a given area is a traditional application of WSNs that we will use here as a case study for clarity and simplicity. The SensGrid extension can also be employed to monitor different magnitudes (noise, pollution, humidity, etc.). As the SensGrid is platform independent, such simulations can be run over Windows, Linux, or MacOS.

With regard to the WSNs considered in the grid, as it is necessary to generate values for the sensed magnitudes in order to achieve realistic simulations, a model for the data returned by sensors must be created (see Section 4.6). It is also necessary to model device failures and battery consumption (see Section 4.3). Likewise, the routing of packets within WSNs must be taken into account, including mechanisms to prevent the formation of cyclic paths (see Section 4.5). Moreover, it is advisable to model some kind of mutual exclusion for the communications of every node (see Section 4.4), because real sensor nodes cannot simultaneously attend to several links (except in some cases, e.g. devices that are able to work on more than one radio frequency).

4.1 Model overview

The real WSG to model can be conceived via two approaches. In the first one, the gateway that is connected to a hypothetical sensor network base station corresponds to the resources available in the grid; as a consequence, sensor nodes are invisible peripherals, such as a monitor or a printer connected to a traditional resource. The second approach considers every sensor node as an individual resource of the grid. The first option has the advantage of an easier deployment and (probably) a more efficient performance of complex queries involving many sensors. In contrast, the latter has the advantage of being more flexible and involving a lower computational load on the gateway. We have chosen the second approach.

The specific topology of the WSG modeled is shown in Figure 1. Every base station is connected to a router that enables it to communicate with the remainder of the grid. Every user computer and the so-called GeoIS server (see Section 4.2) is also connected in the same way. An additional router ('Remainder Grid/Internet') represents the other jumps over the Internet/grid. Its transmission rate can be used to simulate the delay of the packets through the Internet/grid.

The standard sequence of a SensGrid simulation (see Figure 2) is similar to the standard sequence of a GridSim

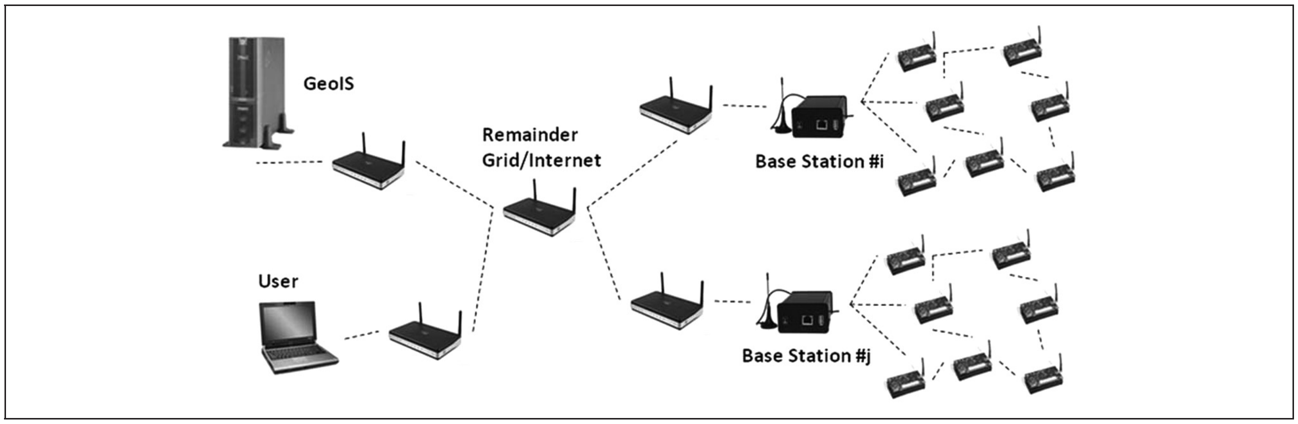


Figure 1. SensGrid topology.

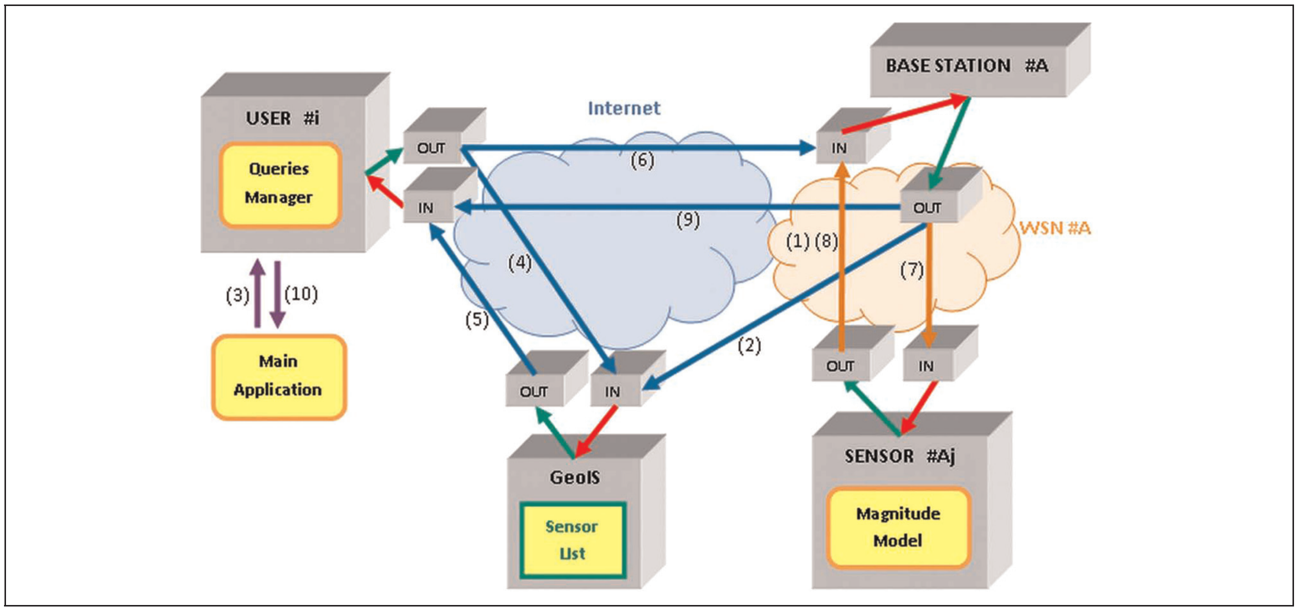


Figure 2. Sequence of a simulation.

simulation, subject to the following conditions. After the sensors are registered in the GIS (1)(2), the main application sends to the grid, instead of a large amount of code to execute, a query relating to the monitored magnitude (3). The user process requests to the GIS the identifiers of all sensors associated to the geographic region specified in the query, the base station which they belong to, and, where appropriate, the weighting of the response for each one (4). Note that when the sensors are registered in the GIS they should indicate their spatial position (in addition to their identifier). Once the user process has obtained the information (5), it sends requests to the sensors concerning the value of the magnitude (6)(7). Before responding to the main application (10), the user process aggregates the data received from the sensors (8)(9).

4.2 Implemented entities

The type of entity that represents resources must be extended to represent the sensors. Instead of processing gridlets, a sensor entity generates magnitude values that are based on a specific model. It also maintains a routing table with one entry for each one of the other nodes (including the base station) of the WSN that it belongs to. Unlike the traditional resource entity, it is not bound to a single wire link but instead can be bound to multiple wireless links. At the time of creating the entity of a sensor node, some parameters have to be specified for the magnitude model, as well as for the sensor's spatial position. We have assumed that the positions of the sensors do not change and the variance of their height above sea level is negligible, that is, they are contained in a 2D space.

It has been necessary to extend the functionality of the GridSim GIS entity, with the understanding that it should now contain, in addition to the identifiers of the sensor resources, information related to their position and the identifier of the base station via which they communicate with the remainder of the grid. To fulfill these new functionalities, we have created a new type of entity that we refer to as the GeoIS (Geographical Information Service). Like the standard GridSim GIS, it is responsible for receiving resource registration requests, maintaining a list of those resources that have been registered, and delivering this list when the user process requires it. The difference is that before delivering the list, the GeoIS performs a filtering based on the criteria that follow. It can accept two types of request, namely punctual region requests and limited area region requests. The first type refers to queries relating to the value of the magnitude at a point, for example, temperature at the position (3.78, 5.90). In the request the user process must include, in addition to the request type, the user identifier, and the X and Y coordinates of the point in question. The manner in which the GeoIS determines the list of sensors to be returned is to choose the closest ones to that position (up to a maximum of five). In the returned list the sensor identifier is included, together with its distance to the point of interest. The user process can use these distances to weight the values that are subsequently received from the sensors. Note that this criterion is more robust than using only the closest sensor. As regards the second type of request, it refers to queries relating to the average value of the magnitude for a zone, the maximum/minimum value for that zone, etc. The request must include the type, the X and Y coordinates of the center of the circular area, and the radius that delimits this area. The GeoIS criterion to filter returned sensors is to select those whose distance from the center of the area of interest is equal to or less than its radius. In the case that there is no sensor that meets the above condition, the criterion used in punctual region requests is employed, considering the center of the area as the point of interest. The returned list contains only the identifiers; in this case all sensors must have the same weight, provided that all points within the area of interest are equally relevant.

In addition to the above two entities, new features for the entity that simulates the process executed by the user have been defined. This entity asks the GeoIS for the sensors that are associated with the region corresponding to a query from the main application, it asks each of them for the value of the modeled magnitude, and, finally, it obtains the information required by the human user from the set of received values. The types of queries that this entity supports are as follows (in all cases the starting point is a set of values $V = \{v_1, v_2, \ldots, v_n\}$ obtained after receiving n responses corresponding to the n sensors in the list returned by the GeoIS).

- Find the average magnitude value in area A. The requested value, referred to as v, is computed via the expression $v = \sum_{i=1}^{n} v_i/n$.
- Find the magnitude value for point P. Besides the set V, in this case the set $D = \{d_1, d_2, \ldots, d_n\}$ is available, where the element d_i corresponds to the distance between P and the sensor that has delivered v_i. The returned value v is computed as $v = \sum_{i=1}^{n} (v_i d_{\min}/d_i) / \sum_{i=1}^{n} (d_{\min}/d_i)$, where $d_{\min}$ is a value such that $d_{\min} \leq d_i, \forall d_i \in D$. Note that the values are weighted over the factor $d_{\min}/d_i$, which takes values in the interval $(0, 1]$ and is equal to 1 for the v_i that is associated to the sensor that is closest to P.
- Find the maximum value in area A. The value v is returned such that $v \geq v_i, \forall v_i \in V$.
- Find the minimum value in area A. The value v is returned such that $v \leq v_i, \forall v_i \in V$.

4.3 Modeling of failures

Typical WSN deployments present a wide variety of failure situations, such as battery depletion, hardware damage due to hard environmental conditions, or, even, device theft. For this reason, our simulator supports the permanent failure of sensor devices. A related work that consists of a GridSim extension supporting non-permanent failures (suitable for regular grids, but not for WSGs) was proposed by Caminero et al.[32]

In our model, the user establishes a parameter n_T to set the average number of nodes that will fail during a simulated day. Internally, once per second, the simulation kernel checks each device liable to fail with a probability $P_F = n_T/(60 \times 60 \times 24 \times N)$, where N is the total number of nodes in the WSN that is being deployed. In the graphical interface of SensGrid, the status (active, failed) of any sensor node can be examined at any time.

A depleted battery is a special type of failure because it is independent of P_F ; in contrast with other types of failures, the energy consumption of sensor nodes is not an aleatory variable. In our simulator, we assume that the consumption due to data sensing and processing is almost insignificant compared with the consumption due to radio communications. The energy consumption has been modeled by means of the expression

$$E(t) = \sum_{i=1}^{n} C_R t_i + \left(t - \sum_{i=1}^{n} t_i\right) C_{R'} \qquad (1)$$

where C_R is the consumption of the communications hardware when it is active (when it is sending/receiving a message), $C_{R'}$ is the consumption of the communications hardware when it is 'inactive' (actually it is never completely inactive, but it can be in *sleeping* mode, which is practically equivalent to the inactive mode), n is the number of messages sent/received by the node, and t_i is the amount of time needed for transmitting the ith message.

Whenever a sensor node entity sends/receives a packet, its battery level is reduced according to C_R and the time required for sending/receiving the packet; whenever a time unit event happens and no packet is being sent/received by the sensor node entity, the battery level is reduced according to $C_{R'}$.

This model can be easily improved in order to be more realistic; normally sensor nodes have more operation modes for the radio transceiver than only active/sleep, for example, MicaZ motes have up to seven different modes. In addition, typical real devices incorporate microprocessors supporting sleeping modes to manage energy more efficiently. In the graphical interface of SensGrid, the battery level of any sensor node can be examined at any time.

4.4 Modeling of radio communications

A wireless link entity is created if and only if two sensor nodes are within coverage range of another. Currently we assume these coverages are always perfect circles, that is, that the radio signal of the sensor nodes always spreads in a circle. However, the real radio signal does not behave in this way; a real signal is heavily influenced by the geography of the environment, as obstacles deform the ideal circle. In order to have a more realistic model for radio propagation, the user should provide the simulator with environmental information regarding obstacles. Future versions of SensGrid will include this option and thus provide a better radio propagation model.

In addition to this, we have modeled radio access too. In real WSNs, except for exceptional cases of multi-frequency sensor nodes, different accesses to the wireless communication medium from a node are mutually excluded. Ergo a node cannot send/receive packets to/from multiple wireless links at the same time. For the purpose of preventing these forbidden behaviors in the simulated WSNs, a mechanism to ensure mutual exclusion in accessing the medium has been modeled as follows.

Every time a node a sends/receives a packet p_i to/from a node b, a packet $p_{i'}$ is put in all the link entities associated to a or b. Inserting these packets has two effects. Firstly, if it is necessary to transmit a packet p_j $(j \neq i)$ from/to node a while p_i is being transmitted, then p_j has to wait until the end of the $p_{i'}$ transmission. Similarly, if it is necessary to transmit a packet p_j $(j \neq i)$ from/to node b while p_i is being transmitted, then p_j has to wait until the end of the transmission of $p_{i'}$. Packets $p_{i'}$ do not carry payload, they are *filling* packets whose only mission is to ensure that when a link is being used, the other links are also in use. Note that the above simulates the state of 'occupied' in the radio transceiver of the node, leaving the inner workings of the simulated WSN more in line with that of a real WSN.

4.5 WSN routing algorithm

All sensors and base stations have static routing tables that are calculated when their corresponding entities are created, that is, prior to the start of the simulation. The routing table of a node has as many entries as existing nodes in the WSN that it belongs to. A routing algorithm is responsible for filling these tables.

We have used a backward algorithm to obtain network routes. This algorithm is based on the one presented by Dijkstra,[38] which was designed for graphs in general and adopted in computer networking (among other areas). In WSNs, physical distance between the source node and destination node is not as relevant as the number of hops the message must make before reaching the destination. Note that propagation time due to physical distance is usually negligible (speed of light) compared with retransmission time due to intermediate nodes (bit rate). So in our algorithm we use the number of hops for the weight concept as used by Dijkstra. All entries related to a destination node (each entry in a separate table) are filled in prior to filling in any entry related to another destination node, as shown below.

Let δ be a function representing the set of entries in a routing table, so that $\delta(i,j) = k$ means that the following jump to reach the destination node j, in the routing table of node i, is the node k. The way to fill in the entries is as follows. Those entries relating to the destination node j in the tables of the m nodes that have direct communication with j, that is, that are positioned within the coverage area of j, are assigned to the value j. Subsequently, those entries relating to the destination j in the nodes that have direct communication with any node i of the m nodes are assigned to the value i. This last step is performed recursively to fill in the entries associated with j in the table of each of the nodes in the network. The above is performed for every possible destination node.

Formally, the algorithm defines the function δ as

$$\delta(i,j) = \begin{cases} j, & d(i,j) \leq r \\ k, & d(i,j) > r \wedge \delta(k,j) = j \wedge d(k,i) \leq r \\ l, & d(i,j) > r \wedge \nexists\, k | (\delta(k,j) = j \wedge d(k,i) \leq r) \\ & \wedge \delta(\delta(l,j),j) = j \wedge d(l,i) \leq r \\ \vdots & \vdots \\ n, & d(i,j) > r \wedge \nexists\, k | (\delta(k,j) = j \wedge d(k,i) \leq r) \wedge \\ & \nexists\, l | (\delta(\delta(l,j),j) = j \wedge d(l,i) \leq r) \wedge \\ & \vdots \\ & \nexists\, m | \delta(\ldots\delta(m,j)\ldots,j) = j \wedge d(m,i) \leq r) \\ & \wedge\, \delta(\delta(\ldots\delta(n,j)\ldots,j),j) = j \wedge d(n,i) \leq r \end{cases}$$

where r is the range of the signal from any node and $d(x,y)$ is a function that returns the Euclidean distance between the geographic positions of nodes x and y.

This algorithm guarantees the absence of cyclic routes, never leaves empty entries if there is any path between the nodes concerned, and always generates the optimal routes (with regard to the number of intermediate hops).

4.6 Modeling of magnitude

To model the magnitude that the sensor entities return we have chosen to use *multivariate polynomial interpolation* (MPI)[39,40] of the second degree. This technique is a particular radial-basis function (RBF) interpolation and, as such, tries to construct a surface (in our case, a bidimensional polynomial surface) that passes through every initial known point. The current magnitude model intends to provide simplicity and efficiency. If a more accurate model were required, a MPI of higher degree (or RBF) would be a better option.

A MPI of the second degree requires nine known values for the magnitude over the area. The user can enter them in the simulator manually in order to obtain a more realistic scenario.[41] In addition, for demonstration purposes, they can be randomly generated. The interpolating function is obtained as described below.

Let us consider a 2D space of points with coordinates x and y. Let z be a third coordinate to represent the values of the studied magnitude. If there are n points with known values (x_i, y_i, z_i), (x_j, y_j, z_j), (x_n, y_n, z_n), then the task is to find a surface that passes through all of them. Once this surface has been calculated, any point with real coordinates x and y will have an associated value $z = f(x, y)$ for its magnitude. A surface with these characteristics should be satisfied by the system of equations

$$\begin{cases} z_i = & c_1x_i^2y_i^2 + c_2x_i^2y_i + c_3x_iy_i^2 + c_4x_i^2 + c_5y_i^2 + c_6x_iy_i \\ & + c_7x_i + c_8y_i + c_9 \\ z_j = & c_1x_j^2y_j^2 + c_2x_j^2y_j + c_3x_jy_j^2 + c_4x_j^2 + c_5y_j^2 + c_6x_jy_j \\ & + c_7x_j + c_8y_j + c_9 \\ & \vdots \\ z_n = & c_1x_n^2y_n^2 + c_2x_n^2y_n + c_3x_ny_n^2 + c_4x_n^2 + c_5y_n^2 + c_6x_ny_n \\ & + c_7x_n + c_8y_n + c_9 \end{cases}$$

Since this system has nine unknowns $(c_1, c_2, \ldots, c_9)$, in order to solve it, and thus to obtain a general expression to compute z for any (x, y), we need nine equations, that is, we need $n = 9$. That is the minimal number of nodes whose values must be provided by the user but, in theory, more nodes may be provided if they are available; the number must be a multiple of nine since there are never more than nine equations (if there are, the degree of the polynomial would increase accordingly). For the case $n = 9m$ where $m \in N \wedge m > 1$, the trick is to divide the region that contains x and y into m sub-regions and to apply the above interpolation $(n = 9)$ for each sub-region. That means not increasing the degree of the interpolating polynomial, which is very important with regards to the computing time. By solving the system of equations with Vandermonde matrices[42] we obtain the expression

$$z = k_1x^2y^2 + k_2x^2y + k_3xy^2 + k_4x^2 + k_5y^2 + k_6xy + k_7x + k_8y + k_9 \tag{2}$$

where any k_i is known.

However, (2) is a static model for the magnitude. When a time-dependent magnitude, for example the temperature of a place in the open, has to be modeled, sensor resources have two additional models that utilize (2) as a basis for generating values in the interval $[z - r/2, z + r/2]$, r being a range established by the user. In one of them, the sensors generate random values, so that $z(t) = z + rand(t) \times r/2$, where we take $Im(rand) = [-1, 1]$. In the other one, the sensors generate values by means of a sinusoidal function, so that $z(t) = z + \sin(\omega t) \times r/2$, where the frequency ω can be determined by the user.

Given that the interpolation function is a second-degree polynomial, the model is computed faster than when another function type is generated or when a higher degree polynomial is generated.

4.6.1 Model validation.

Although magnitude modeling based on bivariate polynomial interpolation has already been employed and validated in other works,[43] in this section we provide a validation of the model incorporated in the SensGrid extension by means of a real experiment, which has been carried out as follows.

A set of 57 samples was collected from a metropolitan area. Every sample is a pair composed of the value measured for the temperature at a given point, and the value measured for the humidity at that point. For each sample we registered its spatial location via Global Positioning System (GPS). Figure 3 shows the area considered and the locations where the measurements were performed, and Table 1 shows the values obtained for the measured magnitudes. By using the points highlighted in the table as known points for the model, we generate both magnitudes.

The average and standard deviation of the difference between the value provided by the model and the value sampled (i.e. the modeling error) is $e_T = 0.6 \pm 0.5$ °C for temperature and $e_H = 1.40 \pm 1.06\%$ for humidity; these statistic estimators are calculated for all the points sampled, except the known ones. Since the ranges of sampled values are $|19 - 26| = 7$ and $|44 - 56| = 12$, relative modeling errors are less than, respectively, 8.6% and 11.7%. We consider these results to be sufficiently accurate.

4.7 Scalability study

As discussed in Section 4.1, the SensGrid extension has been designed according to the approach that considers that every sensor node is a grid resource. This provides us with a high level of flexibility when deploying a very large number of sensors connected to several gateways. On the other hand, it presents scalability issues due to a

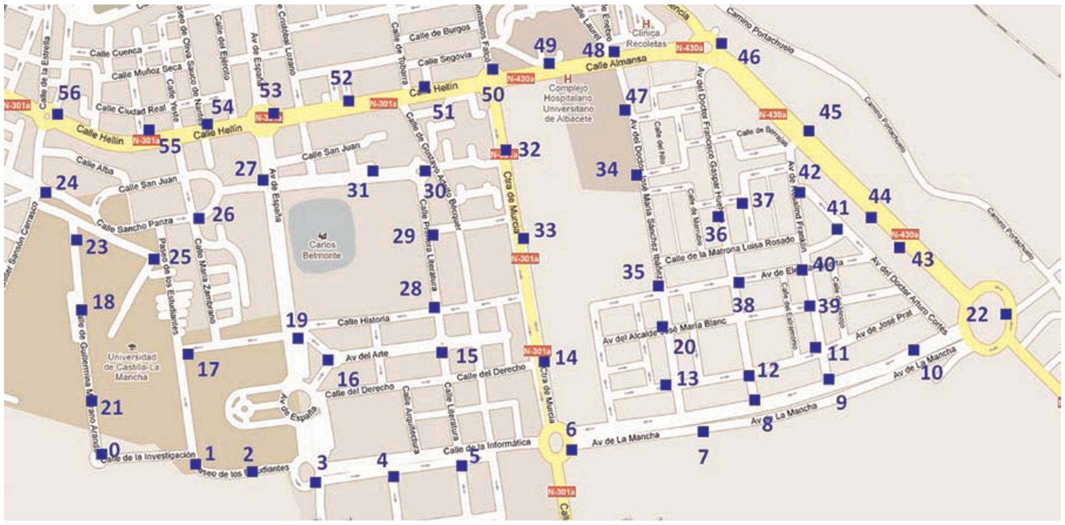


Figure 3. Localizations of sampled points.

well-known drawback of the GridSim thread model,[44] which limits the number of communicating entities.

In this section we analyze the scalability of the SensGrid extension. Several executions have been performed over an area by requesting the same query from a particular sub-area. Figure 4 shows the impact of WSN size on simulation time and computer memory usage. Each point represents the average values of 30 executions when maintaining the network size, but varying the topology to be used. Simulations have been performed on a conventional personal computer (PC).

As expected, memory use has a quasi-linear behavior, whereas simulation time increases in an exponential way. For network sizes around 1000 nodes, simulations take about one minute. For those studies demanding larger deployments (which is not very realistic today), an alternative implementation considering the entire WSN as a unique resource would be more suitable.

5 User interfaces

We have developed both a graphical interface and a command line interface to allow the user to interact with the simulator. Furthermore, it is possible to interact with the simulator in batch jobs (this is especially interesting for large experiments); a configuration file is used in such cases.

5.1 Graphical interface

If the end user's computer does not impose severe restrictions on memory and/or the processor, then the graphical interface is more appropriate for using SensGrid interactively. This interface has been developed to be as user friendly as possible. In this respect, the interface guides the user closely so that it is not possible for him/her to commit errors caused by actions in the wrong order, for example, launching queries before deploying any WSNs or deploying a WSN before generating a model for the 'sensed' magnitude. If the user attempts to perform an action in the wrong order, he/she will be informed in a timely manner of the actions that must be undertaken prior to that one.

A useful feature offered by the graphical interface is that the user can scale the simulation time in relation to the real time, that is, they can accelerate/decelerate the simulation as they wish. This is done using the upper slider in the window and it may be used, for instance, to quickly view the evolution of the sensed magnitude, which in the simulation takes place over hours or even days. The maximum scale factor is 50; a second in reality may correspond from $1/50 = 0.02$ seconds up to 50 seconds in simulation (by default it corresponds to 1 second). The different sections that make up the rest of the main window of the interface are described below.

5.1.1 Model of the Magnitude. This corresponds to the upper right part of Figure 5 and groups together the actions related to the magnitude under study. In this panel, the dimensions of the plane on which the model is created are defined. Since the model is generated from nine known initial points (see Section 4.6), before it can be created these points must be defined. There are four options for such a definition, namely, points with random values and random positions ('Irregular' + 'Generate Points'), points with random values and mesh positions ('3 × 3 Matrix' + 'Generate Points'), points with both the values and the

Table 1. Measured magnitudes for sampled points. Values used for initial points of the model are highlighted.

Point	Temp. (°C)	Hum. (%)
0	**23**	**41**
1	22	44
2	21	**45**
3	21	45
4	20	49
5	**20**	47
6	20	50
7	19	56
8	20	54
9	**20**	51
10	20	52
11	19	52
12	19	52
13	19	52
14	20	50
15	21	46
16	20	**47**
17	**21**	46
18	21	46
19	21	48
20	19	51
21	21	46
22	20	**53**
23	21	**45**
24	22	44
25	21	46
26	21	45
27	20	46
28	21	46
29	21	46
30	**20**	48
31	21	45
32	26	**49**
33	20	**49**
34	20	49
35	19	50
36	20	51
37	19	52
38	19	52
39	19	52
40	19	52
41	19	52
42	19	52
43	20	49
44	20	48
45	**20**	48
46	20	49
47	20	50
48	**20**	**50**
49	19	51
50	20	51
51	**20**	48
52	21	47
53	20	48
54	20	**47**
55	20	48
56	**21**	48

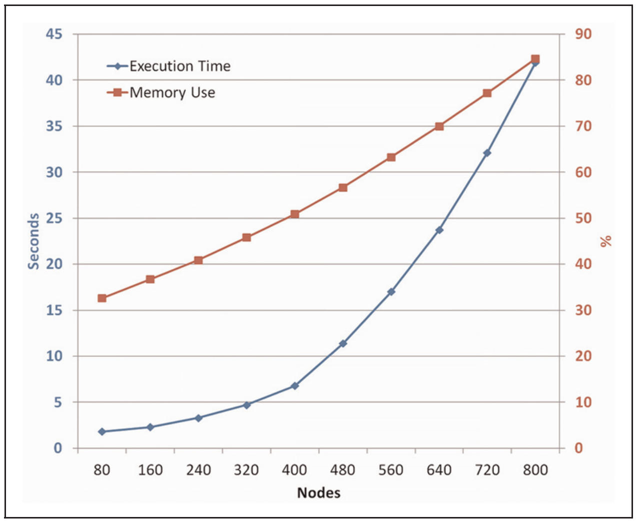
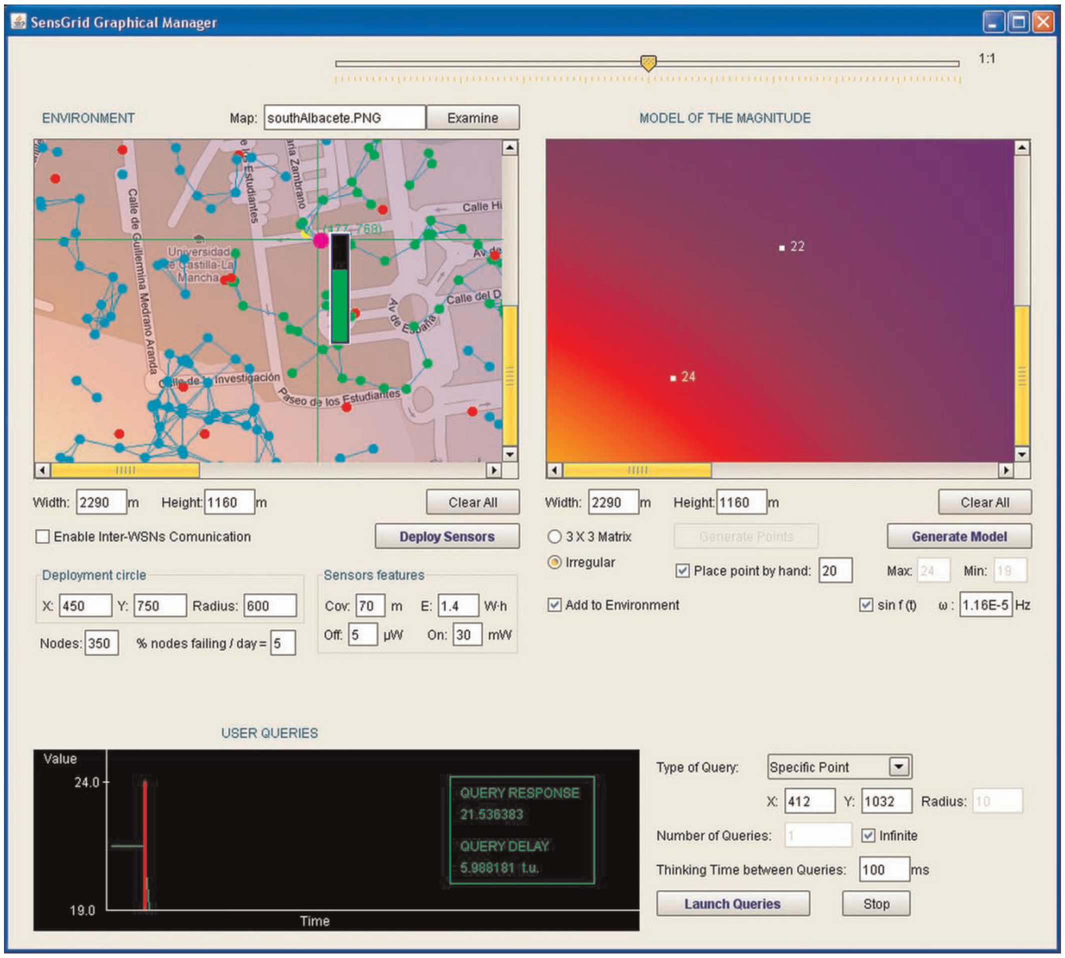


Figure 4. Simulator performance in relation to the number of simulated nodes.

positions specified by the user ('Irregular' + 'Place point by hand'), and points with values specified by the user and mesh positions ('3 × 3 Matrix' + 'Place point by hand'). The first, third, and fourth options are only available to this interface; in the command line interface the points are always set to random values with mesh positions. Regardless of whether their values are random or entered by the user, it is recommended that the positions of the points form a mesh of 3 × 3, because in this way the bivariate polynomial interpolation's results are smoother and therefore more realistic. Via this panel the minimum and maximum values of the model are also established, the temporal evolution (sinusoidal function or normal distribution) is defined, and, where applicable, the value in hertz of the sinusoidal function's frequency ω is established. In Figure 5 the same model that we use for the temperature in Section 4.6.1 is set up. So, the model shown in the figure is generated from those nine points highlighted in that column for temperature in Table 1; also, the temporal evolution of the shown model is given by a sinusoidal function with the ω value for the day/night cycle. But of course, a generated model will have no relevance if it is not added to the environment.

5.1.2 Environment. This section of the graphical interface corresponds to the upper left part of Figure 5 and groups together the actions related to the deployment of WSNs in the environment. In this panel, the dimensions are defined for the environment in which one or more WSNs are deployed. Setting up these dimensions will automatically alter the dimensions specified for the plane on which the model is created, in order to match the two tuples. Likewise, modifying the dimensions for this plane will


SensGrid Graphical Manager
1:1
Calle de Guillermina Medrano Aranda
Universidad de Castilla-La Mancha
Calle de la Investigación
Paseo de los Estudiantes
Av. de España
Calle del D
Calle Hi
(412, 768)
22
24

ENVIRONMENT

Map: southAlbacete.PNG Examine

Width: 2290 m Height: 1160 m Clear All

☐ Enable Inter-WSNs Comunication Deploy Sensors

Deployment circle: X: 450 Y: 750 Radius: 600

Nodes: 350 % nodes failing / day = 5

Sensors features: Cov: 70 m E: 1.4 W·h Off: 5 µW On: 30 mW

MODEL OF THE MAGNITUDE

Width: 2290 m Height: 1160 m Clear All

○ 3 X 3 Matrix Generate Points Generate Model

◉ Irregular ☑ Place point by hand: 20 Max: 24 Min: 19

☑ Add to Environment ☑ sin f(t) ω : 1.16E-5 Hz

USER QUERIES

Value 24.0 19.0 Time

QUERY RESPONSE 21.536383

QUERY DELAY 5.988181 t.u.

Type of Query: Specific Point

X: 412 Y: 1032 Radius: 10

Number of Queries: 1 ☑ Infinite

Thinking Time between Queries: 100 ms

Launch Queries Stop

Figure 5. SensGrid graphical interface.

automatically alter the dimensions of the environment. Via the environment panel it is possible to define the coverage range of the nodes in a WSN, the initial battery level of nodes (Watt-hour), the hardware energy consumption (both in active mode and sleep mode), the number of nodes and the percentage of them that are broken within a time interval (a day), the position in the environment of the center of the WSN (i.e. the base station, yellow node in Figure 5), and the radius of the circular area of deployment. To facilitate simulation viewing for the user, there exists the option of loading a real map (an image in JPEG, BMP, etc.) to perform the environment background role. As many WSNs may be deployed as deemed appropriate. In Figure 5 an environment is set up whose parameters correspond to the real environment in which we gather samples for Section 4.6.1. The possible nodes whose position is outside the rectangle of the environment will not be shown or be taken into account in the simulation. In this figure, each sensor node in the deployed WSN has 70 meters of coverage, an initial battery level of 1.4 Wh (AAA batteries) and a consumption of 5 µW when the transceiver is in sleep mode and 30 mW when it is active (these are reasonable values for typical motes). Nodes whose battery is depleted are not included in the 5% of nodes that fail in a day. The current state of battery power for any sensor node can be observed by moving the mouse over its position.

5.1.3 User queries. This section corresponds to the bottom of Figure 5 and groups together the actions related to user queries performed on the simulated grid. Via this panel the parameters of a query are specified, that is, the type and geographical region to which it refers. Also in this panel it is possible to specify the number of queries of the same type to be launched and the thinking time between them. In Figure 5 the temperature at the position associated to point 1 of Figure 3 (≈ (412, 1032) in environment) is queried every 100 milliseconds. Note that the difference between the real sampled value (see Table 1) and the query response is less than half of one degree.

5.2 Command line interface

In the command line interface it is necessary to enter all the parameters of the simulation in a sequential manner. The queries are performed by a SQL-like language whose syntax is

```
SELECT <queryType> FROM <magnitudeName>
WHERE x=<queryX> y=<queryY>
  [r=<queryRadius>]
```

where the tag *queryType* may be 'point', 'avg', 'max', or 'min'. For instance, the equivalent to launching the queries shown in the graphical interface screenshot is as follows:

```
SELECT avg FROM temperature WHERE x=446.0
 y=440.0 r=100
```

Note that, in contrast with the graphical interface, it is not possible to program a number of queries with thinking times between them in the command line interface.

6 Example experiment

As an example of the possibilities that SensGrid may provide to researchers, we have performed the following experiment. Its purpose is to study the influence of the density of the WSN that is deployed in an environment and the coverage range of the sensor nodes composing that WSN on the performance of a WSG system. Simulation runs were carried out by using the batch execution mode of SensGrid, that is, by using a configuration file instead of an interactive input.

6.1 Modeled environment

The simulated scenario is a metropolitan area inside the city of Albacete (Spain) of 100 hectares (1000 $\times$ 1000 meters). The modeled magnitude is the temperature. We assume that it can vary spatially by up to 5 °C (due to the sun/shade difference) and we establish a maximum value of 16.5° C (value registered in Albacete on 27 March), so we set up its range as $v_{\max} = 16.5\,^\circ\mathrm{C}$ and $v_{\min} = v_{\max} - r = 16.5 - 5 = 11.5\,^\circ\mathrm{C}$. The nine initial points for the magnitude model are randomly generated in that range. To build the model we considered a real-time variation of temperature, that is, a sinusoidal evolution with frequency $\omega = 1/(\text{seconds per day}) = 1/(60 \times 60 \times 24) = 1.16 \cdot 10^{-5}$, which represents the day/night cycle.

6.2 Metrics

The metrics used to evaluate performance are the response time t_r (according to the simulator clock) for a user's query to the system and the accuracy of the response provided by it. To measure the accuracy we use the absolute error e_a in relation to the real value (i.e. the modeled one) for the geographical region to which the query refers. Thus we have

$$metrics \begin{cases} e_a = & |v\prime - v| \\ t_r = & |t_c\prime - t_c| \end{cases}$$

where $v\prime$ is the value returned to the user, v is the modeled value for the geographical region of the query, t_c is the time at which the user launches the query, and $t_c{}'$ is the time at which the response is received. For queries about the media in a region we have calculated the value v using the expression (see Section 4.6)

$$v = \frac{1}{r_c^2 \pi} \sum_{x=x_c-r}^{x_c+r} \sum_{y=y_c-r}^{y_c+r} \left(\chi(\Phi(x,y), t_c) \Leftrightarrow x^2 + y^2 \le r_c^2 \right) \quad (3)$$

where x_c and y_c are the coordinates x and y, respectively, of the region indicated by the query and r_c is the radius of that region. Note that, in the summation, variables x and y can only take discrete values, so the calculated value for v does not exactly correspond with the average of the modeled values for the magnitude in the region. It also happens that this average should be obtained after calculating the closed integral of the function χ in a circular area. Despite not obtaining an accurate value, expression (3) is valid for our experiment, as is explained below. Suppose that the region whose mean is being calculated is not circular, but square with sides equal to $2r_c$. Then the expression to calculate the exact value of the average would be

$$\begin{aligned} v_{real} &= \frac{1}{Area} \oint \chi(\Phi(x,y), t_c)\, dx\, dy \\ &= \frac{1}{4r_c^2} \int_{y_c-r_c}^{y_c+r_c} \left(\int_{x_c-r_c}^{x_c+r_c} \chi(\Phi(x,y), t_c) dx \right) dy \end{aligned}$$

That is, the result of integrating all the possible values of the magnitude in the region divided by the area of this region. The expression that calculates an average value for this hypothetical square region using only discrete coordinates would be

$$v_{discr} = \frac{1}{4r_c^2} \sum_{x=x_c-r}^{x_c+r} \sum_{y=y_c-r}^{y_c+r} \chi(\Phi(x,y), t_c)$$

We have compared the respective values v_{real} and v_{discr} for a total of 10 different square regions of 600×600, considering the discrete values of x relative to integers in the interval $[x_c - 300, x_c + 300]$ and the discrete values of y relative to integers in the interval $[y_c - 300, y_c + 300]$. Results show that the difference between v_{real} and v_{discr} can be neglected. Generalizing from the above, we have assumed that in a circular region the difference can also be neglected and hence the expression (3) serves our purpose for the experiment. For queries concerning the value of the magnitude at a point, however, the value v is calculated as $v = \chi(\Phi(x_c, y_c), t_c)$.

6.3 Simulation methodology

Every simulation launched for the experiment consists of two queries separated by an interval of 12 simulated hours (i.e. thinking time = 43,200,000 ms), so each simulation lasts this interval plus the response time for the two user queries. The input parameters varied for each simulation correspond to the density of sensors in the environment and the coverage range of the signal emitted by them. The way to vary the density of sensors was to keep the size of the deployment area constant and to vary the number of sensors deployed. The configurations considered in the experiment correspond to the elements of $C \times D$, where C is the set of coverages 10, 30, 50, 70 and 90 (values in meters), and D is the set of densities, ranging from 1.4147 nodes/ha to 14.1471 nodes/ha with intervals of 1.4147 nodes/ha between them. In order to avoid topology dependence, for every configuration 100 simulations were executed; the values for the metrics of a configuration correspond, respectively, to the average of its 100 e_a and the average of its 100 t_r. In those simulations where the value returned by the GeoIS to the user process was an empty list of sensors (see Section 4.2), that is, in cases where there was no sensor communicating with the base station, it was considered that the error of the query is maximum and equal to the static model range. Owing to the huge number of simulations (100×10 densities $\times 5$ coverages $\times 2$ types $= 10000$ runs), it was decided to employ a dedicated cluster of 64 nodes with Intel Xeon 3 GHz clock frequency.

6.4 Analysis of results

In this section we analyze the performance of both punctual and area queries in the scenarios described in the previous section. Results obtained for punctual queries are shown in Figure 6. As is well known, deployments with low coverage require high densities in order to maintain good connectivity. With low network densities, the list handled by the GeoIS includes only a few nodes located in its surroundings, and far away from the requested location. As density increases, the GeoIS discovers better positioned nodes closer to the query point. Routes to these latter nodes require more hops, which has an impact on query delay.

In the plots, series 'cov 10 m' represents such a low coverage that the network is mostly disconnected. In this situation, the list returned by the GeoIS is empty in almost all cases and, therefore, the response to the query is practically instantaneous, and with the maximum error. In the 'cov 30 m' series, as density increases, the GeoIS has nodes that get gradually closer to the requested location.

The rest of the series exhibit the same behavior, but with greater intensity. Network densities greater or equal to 4 nodes/ha guarantee that the GeoIS has registered the closest nodes to the requested point. In this situation, we obtain the most accurate response, and the delay reaches an upper bound (approximately 2 s).

Results obtained for area queries are shown in Figure 7. Compared with the punctual queries, we can observe several differences, because we gather information from an unknown number of nodes (instead of only considering the five best ones). The first difference is that as network density increases, a greater number of nodes are involved in the query. Therefore, response time is longer, without presenting an upper bound. Another consequence is that, in this situation, densities greater than 8 nodes/ha practically do not contribute to improving the accuracy. A small error remains in the results due to the inherent nature of the process (the variation between the population average and sample mean).

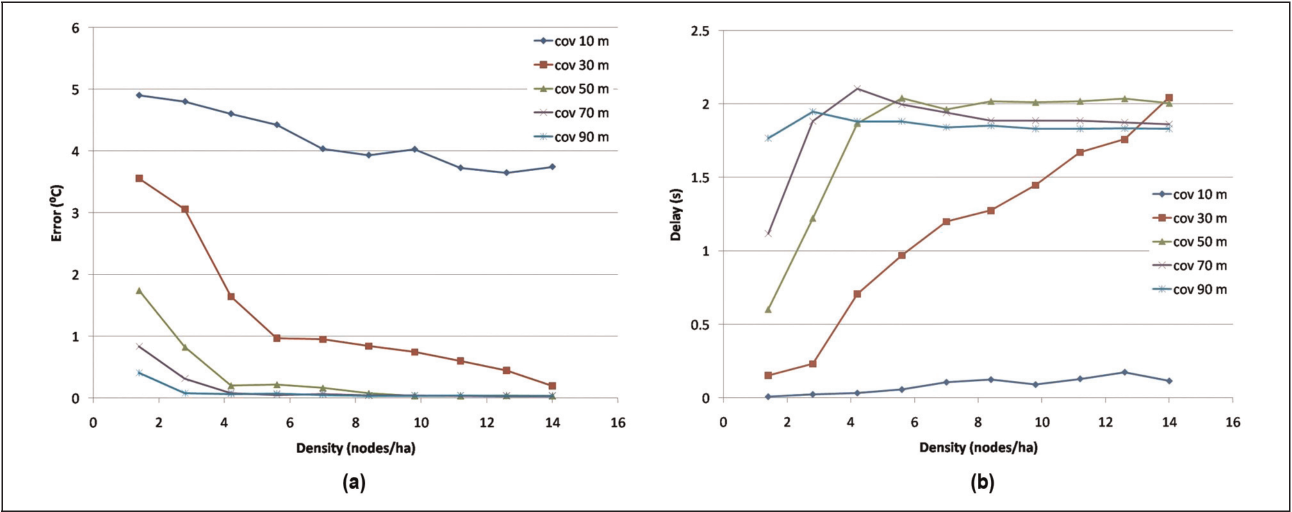


Figure 6. Experiment results for punctual queries. (a) Accuracy for punctual queries. (b) Response time for punctual queries.

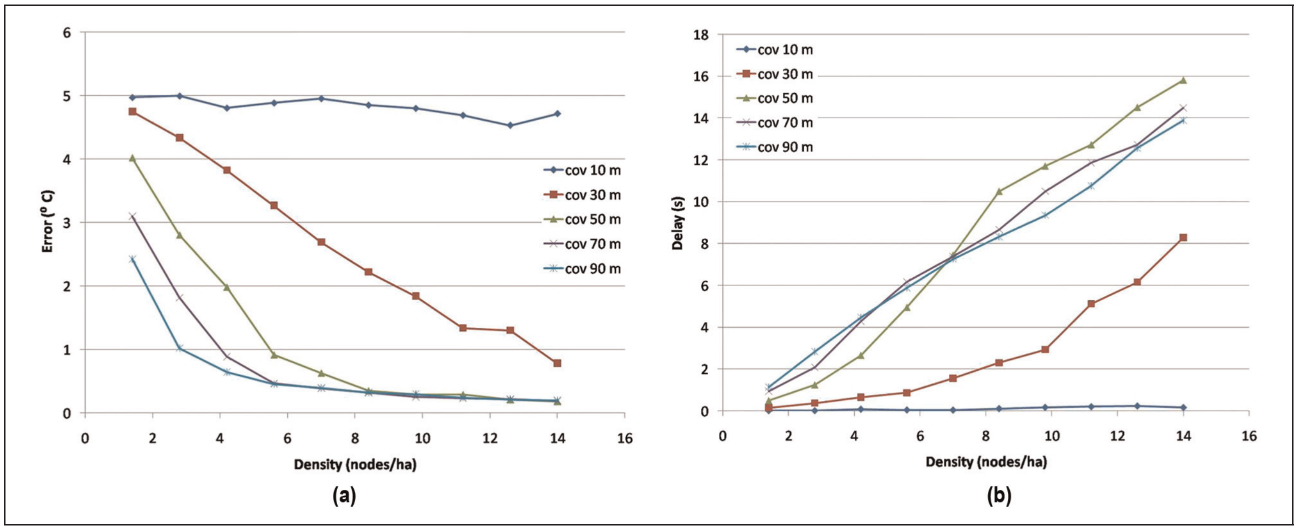


Figure 7. Experiment results for area queries. (a) Accuracy for area queries. (b) Response time for area queries.

There is an especially curious behavior, which can be observed in the figures. When the accuracy is approximately the same for different coverages the response time becomes a turning point, that is, the delay is greater for low coverages instead of for high coverages. The cause of such behavior is that the number of sensors polled is the same but the number of links to reach the base station is probably lower if the radius of coverage is higher.

The main idea lies in using an analysis similar to the one above to help draw conclusions regarding a possible real WSG deployment. Such an analysis could guide the choice of the radio coverage for the nodes or the optimal number of these to be deployed.

7 Conclusions and future work

Merging grid computing and sensor networks provides a wide range of interesting applications. Several efforts have been made toward the integration of these two technologies. All these efforts are related to the implementation of real systems. Since it is wise to test the behavior of such systems before they are implemented, we have proposed a simulation architecture that accomplishes this task by extending the GridSim grid simulation tool. In the modeling process for the system to be simulated, issues such as the routing scheme within the sensor networks, mutual exclusion in accessing wireless communications, and the sensed magnitude have been taken into account. By means of a sample experiment we have shown that our proposal may be very helpful in finding the optimal configuration for a real grid deployment that integrates sensor networks.

We plan as future work to integrate our SensGrid simulator inside a server. This will mean that users can perform simulations of grids that include sensor networks through a remote web browser. Therefore, possible updates to the simulator will be instantly applied without needing any action by the user. Likewise, we plan to improve the wireless communication model in order to reflect a more realistic signal pattern; this will be achieved by means of having the option of including obstacles, which would disturb ideal propagation. Future versions of SensGrid will also incorporate the WSG model that considers only one grid resource per sensor network; we expect to reduce the detected scalability problem by employing such an approach.

Funding

This work was supported by the Spanish Ministry of Education (CSD2006-00046); the Spanish Ministry of Science and Innovation (CSD2006-00046); the European Regional Development Funds (TIN2009-14475-C04); and the Regional Government of Castilla-La Mancha (PREG-07-25, PII1C09-0101-9476).

References

1. Baker M, Buyya R and Laforenza D. Grids and grid technologies for wide-area distributed computing. *Software Pract Ex* 2002; 32: 1437–1466.
2. The Cloud Computing and Distributed Systems (CLOUDS) Laboratory. GridSim homepage, http://www.gridbus.org/gridsim/ (Accessed on 22 September 2010).
3. Akyildiz IF, et al. A survey on sensor networks. *IEEE Commun Mag* 2002; 40: 102–114.
4. De Roure D. Improving flood warning times using pervasive and grid computing. *Ingenia Mag* 2005; 1: 48–51.
5. Sanabria J, et al. A sensor grid framework for acoustic surveillance applications. In: *proceedings of the fifth international joint conference on INC, IMS and IDC*, Seoul Korea, 2009, pp.31–38.
6. Rehman A, Shaikh ZA, Shaikh NA, et al. An integrated framework to develop context-aware sensor grid for agriculture. *Aust J Basic Appl Sci* 2010; 4: 922–931.

7. Pallikonda-Rajasekaran M, Radhakrishnan S and Subbaraj P. Sensor grid applications in patient monitoring. *Future Generat Comput Syst* 2010; 26: 569–575.
8. Gaynor M, et al. Integrating WSN with the grid. *IEEE Internet Comput Mag* 2004; 8: 32–39.
9. Tham CK and Buyya R. SensorGrid: Integrating sensor networks and grid computing. *CSI Commun* 2005; 29: 24–29.
10. Bramley R, et al. Instruments and sensors as network services: Making instruments first class members of the grid. Technical report, 588, Indiana University, Computer Science Department, 2003.
11. Hock BL, et al. Sensor grid: Integration of wireless sensor networks and the grid. In: *proceedings of the 30th IEEE conference on local computer networks (LCN'05)*, Sydney, Australia, 2005, pp.92–98.
12. Chu X. Open sensor web architecture: core services. Minor Project Thesis of Master of Information Technology, University of Melbourne, 2005.
13. Botts M, et al. Sensor web enablement: Overview and high level architecture. Technical report OGC 07–165, Open Geospatial Consortium, 2007.
14. Kobialka T, et al. Sensor web: Integration of sensor networks with web and cyber infrastructure. In: *Handbook of research on developments and trends in wireless sensor networks: From principle to practice.* IGI Global, Hershey, PA, USA, 2010, Chapter 20.
15. CrisisGrid homepage, http://www.crisisgrid.org/cgwiki/ (Accessed on 15 September 2010).
16. QuakeSim homepage, http://quakesim.jpl.nasa.gov (Accessed on 15 September 2010).
17. Avilés E and García JA. Providing service-oriented abstractions for the wireless sensor grid. In: *proceedings of the 2nd international conference on advances in grid and pervasive computing (GPC'07)* in LNCS 4459, Paris, France, 2007, pp.710–715.
18. Fox G, et al. A collaborative sensor grids framework. In: *proceedings of the international symposium on collaborative technologies and systems (CTS'08),* Irvine, USA, 2008, pp.29–38.
19. Stasch C, Walkowski AC and Jirka S. A geosensor network architecture for disaster management based on open standards. In: *digital earth summit on geoinformatics 2008: tools for climate change research*, Potsdam, Germany, 2008, pp.54–59.
20. Li X, et al. ASGrid: autonomic management of hybrid sensor grid systems and applications. *Int J Sensor Networks* 2009; 6: 234–250.
21. Su S and Tham CK. SensorGrid for real-time traffic management. In: *proceedings of the 3rd international conference on intelligent sensors, sensor networks and information processing (ISSNIP'07),* Melbourne, Australia, 2007, pp.443–448.
22. Kao FC, et al. The design of intelligent sensor network based on grid structure. In: *proceedings of the IEEE Asia-Pacific services computing conference (APSCC'08)*, Yilan, Taiwan, 2008, pp.623–630.
23. Tham CK. Sensor-grid computing and sensorgrid architecture for event detection, classification and decision-making. In: *Sensor networks and configuration: fundamentals, standards, platforms, and applications*. Springer, Heidelberg, Germany, 2007, Chapter 5.
24. Ahuja SP and Myers JR. A survey on wireless grid computing. *J Supercomput* 2006; 37: 3–21.
25. Gang L, et al. A survey on wireless grids and clouds. In: *proceedings of the eighth international conference on grid and cooperative computing*, Lanzhou, China, 2009, pp.261–267.
26. Sulistio A, et al. A toolkit for modelling and simulating data grids: An extension to GridSim. *Concurrency Comput Pract Ex* 2008; 20: 1591–1609.
27. Sulistio A and Buyya R. A grid simulation infrastructure supporting advance reservation. In: *proceedings of the 16th international conference on parallel and distributed computing and systems (PDCS'04)*, Cambridge, USA, 2004, pp.1–7.
28. Sulistio A, Yeo CS and Buyya R. Visual modeler for grid modeling and simulation (GridSim) toolkit. In: *proceedings of the 3rd international conference on computational science (ICCS'03)* in LNCS 2659, Melbourne, Australia, 2003, pp.1123–1132.
29. Kurowski K, et al. Grid scheduling simulations with GSSIM. In: *proceedings of the 13th international conference on parallel and distributed systems (ICPADS'07),* Hsinchu, Taiwan, 2007, pp.1–8.
30. Klusáč ek D, Matyska L and Rudová H. Alea - Grid scheduling simulation environment. In: *proceedings of the 7th international conference on parallel processing and applied mathematics (PPAM'07)*, Gdan′sk, Poland, 2007, pp. 1029–1038.
31. Klusáček D and Rudová H. Alea 2 - Job scheduling simulator. In: *proceedings of the 3rd international conference on simulation tools and techniques (SIMUTools 2010)*, Málaga, Spain, 2010.
32. Caminero A, et al. Extending GridSim with an architecture for failure detection. In: *proceedings of the 13th international conference on parallel and distributed systems (ICPADS'07)*, Hsinchu, Taiwan, 2007, pp.1–8.
33. Caminero A, et al. Simulation of buffer management policies in networks for grids. In: *proceedings of the 41st annual simulation symposium (ANSS'08)*, Ottawa, Canada, 2008, pp.1–8.
34. Dias de Assuncao M and Buyya R. An evaluation of communication demand of auction protocols in grid environments. In: *proceedings of the 3rd international workshop on grid economics and business (GECON'06)*, Singapore, 2006, pp.24–33.
35. Messina F, et al. A QoS-aware architecture for multimedia content provisioning in a grid environment. In: *proceedings of the 7th workshop from objects to agents (WOA'06)*, Catania, Italy, 2006, pp.60–65.
36. Sulistio A, et al. On incorporating differentiated levels of network service into GridSim. *Future Generat Comput Syst* 2007; 23: 606–615.
37. SimJava homepage, http://www.dcs.ed.ac.uk/home/hase/simjava (Accessed on 22 September 2010).
38. Dijkstra EW. A note on two problems in connexion with graphs. *Numer Math* 1959; 1: 269–271.
39. Birkhoff G. The algebra of multivariate interpolation. In: *Constructive approaches to mathematical models (proceedings of a conference in honor of R. J. Duffin)*, New York, 1979, pp.345–363.

40. Fasshauer G. *Meshfree approximation methods with MATLAB*. River Edge, NJ: World Scientific Publishing Co., Inc., 2007.
41. Caliari M, De Marchi S and Vianello M. Bivariate polynomial interpolation on the square at new nodal sets. *Appl Math Comput* 2005; 165: 261–274.
42. Kalman D. The generalized Vandermonde Matrix. *Math Mag* 1984; 57: 15–24.
43. Goodale CL, Aber JD and Ollinge SV. Mapping monthly precipitation, temperature, and solar radiation for Ireland with polynomial regression and a digital elevation model. *Clim Res* 1998; 10: 35–49.
44. Depoorter W, et al. Scalability of grid simulators: an evaluation. In: *proceedings of the 14th international euro-par conference on parallel processing (Euro-Par'08)*, Las Palmas de Gran Canaria, Spain, 2008, pp.544–553. Springer.

Author biographies

Raúl Moreno received his BSc degree in Computer Science in 2009 and his MSc degree in Computer Science in 2010, both from the University of Castilla-La Mancha. He is currently a PhD student in the Computing Systems Department of that university. His research interests include modeling in WSNs and routing for vehicular ad hoc networks.

Antonio Robles-Gómez received his MSc degree in Computer Science in 2004 and his PhD in Computer Science in 2008, both from the University of Castilla-La Mancha. He is an Assistant Professor at the Control and Communication Systems Department at the Spanish University for Distance Education, UNED. He teaches graduate and postgraduate courses related to the network interconnection and security domains. His research interests include QoS support in distributed systems and development of infrastructures for e-learning. He has co-authored more than 25 publications in international journals and conferences on these topics. He is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Aurelio Bermúdez received his PhD degree in Computer Science in 2004. He is an associate professor in computer architecture at the Computing Systems Department of the Universidad de Castilla-La Mancha (UCLM). His research interests include modeling, routing, and fault tolerance in interconnection networks, and localization and collaborative processing algorithms for WSNs. He has co-authored more than 30 publications in these areas. He is a member of the IEEE.

Rafael Casado received his BSc degree in Computer Science from the UCLM in 1993, his MS degree in Computer Science from the University of Murcia in 1995, and his PhD degree in Computer Science from the UCLM in 2001. In 1998, he joined the Department of Computer Engineering at the UCLM and is an associate professor at this department. His research interests include routing and reconfiguration algorithms for high-speed networks and multicomputers, and intelligent collaborative processing in WSNs. He has participated in more than 30 research projects at national and regional level, conducting several of them. Currently, he is leading a regional project focusing on the use of WSNs to monitor wildfires. He has co-authored more than 30 publications in these areas. He has served as a member of the program committee and reviewer in several conferences and journals, including some of the most prestigious in these areas. He is a member of the IEEE.