CHALMERS

Monitoring Sensor Nodes Using 3D MAP and Distributed Group Communication File Transfer Applications

Master of Science Thesis in the Networks and Distributed Program

IMAD RIZK

Department of Computer Science and Engineering
Division of networks and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010
Report No. xxxx

Abstract

The purpose of this master thesis is to describe a software suite project that allows remote visualization, monitoring and data acquisition from sensor nodes that could be scattered over a vast geographical area. Also, it describes a rather a distributed algorithm which allows users to join groups and share the same set of files within a group with other member. Thus, the concept of group communication is extended to develop a distributed group communication file transfer application. Consequently, the main objective behind this work is to allow individuals to monitor sensor nodes from any place in the world using the comfort of their web browsers, and to download data files from those nodes without the burden of physical presence near them, as well as the development of a reliable distributed group communication file transfer application or protocol that allows users within a group to share same set of data files related sensor nodes with other group memebers.

Acknowledgement

Table of Contents

Table of Contents

List of Figures

List of Abbreviations

API	Application Programming Interface
DGCFTA	Distributed Group Communication File Transfer Application
DGCFT	Distributed Group Communication File Transfer
FE	Front End
LAN	Local Area Network
TCP	Transmission Control Protocol
XDAS	XML Data Acquisition Server

Chapter 1 Introduction

1.1 Introduction to Sensor Nodes

Advances in communication, electronics and system integration have enabled the creation of small sensor nodes that can be clustered together in a wired or wireless fashion to work on a specific task. Wireless sensor nodes, particularly, can be spatially distributed to collectively monitor the environmental and physical changes related to a specific geographical area (1) (2). Typically, sensor nodes contain a set of small embedded sensors that sample the surrounding environment, actuators that manipulate other components when changes are detected, and communication modules which allow the nodes to communicate and relay data to each other.

The research around sensor nodes have been active recently, and their applications range from large scale monitoring such as habitat monitoring (3) (4), and border surveillance (5) to small scale monitoring and control such as building automation (6), and smart environments (7). Other important fields of research include but are not limited to networking (8), health monitoring (9) (10), and even embedding nodes in textile (11).

1.2 Sensor Nodes Operation

Sensor nodes can work on a specific task either individually or collaboratively. It is no surprise that the second mode of operation is more attractive and challenging. In this mode, sensor nodes participate together in a distributed processing task by possibly forming a wireless ad-hoc network, and routing sampled date from the environment between each other. While relaying and routing data between nodes, there could be one or more gateway sensor nodes along the path that aggregate data from multiple nodes and sink this data to a base station for further processing. Typically, gateway nodes have more processing power and higher communication stack levels than other nodes, and can thus communicate with more intelligent devices, such as a PC.

1.3 Sensor Nodes Localization

Geographical awareness can sometimes means the difference between life and death especially in disaster events and thus it is an essential factor in our daily life. Hence, sensor nodes deposited to monitor an environment might need to have location awareness services built into them. There are several approaches to embed location intelligence into sensor nodes as discussed in (12). One approach to localization is to embed a Global Positioning System (GPS) receiver in each sensor nose. This approach, of course, can lead to increase in both cost and power consumption, which are a critical factor in sensor node because they are usually designed to be cheap and to operate for a long period of time. Another method for localization

is to let some nodes run an algorithm that calculates their position with respect to other nodes, called anchor or beacon nodes. Anchor nodes know their position beforehand by either plugging in the geographical coordinates in them at programming time or by equipping the nodes with GPS receivers.

1.4 Project Description and Objectives

After enabling localization service in sensor node, the challenge is to correlate this service with a graphical interface that allows its user to view the geographical location of nodes in a comprehensible manner. The purpose of this master thesis project is to describe the development of a set of applications that allow remote visualization, monitoring and data acquisition from sensor nodes that are scattered over a vast geographical area. Also, the concept of group communication is extended to develop a distributed group communication file transfer application, which allows users to join groups and share the same set of files that belong to that group with other users within it.

The main objective behind this work is to allow individuals to monitor sensor nodes from any place in the world using the comfort of their web browsers, and to download data from those nodes without the burden of physical presence near them, as well as the development of a reliable distributed group communication file transfer application or protocol that allows users within a group to share same set of data files downloaded from or related to available sensor nodes.

Chapter 2 Motivation and Requirements

2.1 Introduction

In this chapter the aim behind this project is presented. First, the problem statements, and the motivations behind solving this problem are presented. Then, the project's requirements will be explained and discussed.

2.2 Problem Statement

Monitoring a large number of sensor nodes can be both difficult and pain stacking process especially if these nodes are scattered over a large geographical area. Apart from deploying sensor nodes at geographically known spots, too often the aim is to monitor a large geographical area where wireless sensor nodes are deployed by throwing them from a plane at an altitude, so the location of these nodes is not exactly known. And even if these nodes send geographical data that corresponding to their location, without the proper way to display these data the user will be oblivious to their location. Also, sometimes the data gathered from the sensor nodes can be critical and directly related to their geographical location. Consequently, it pays to have a system that can accurately to some extend and according to the currently available technology to pinpoint the position of these nodes in a map structure in case any professional, operational or humanitarian intervention is required. Furthermore, there is always a need to access data gathered from sensor node remotely from any place in the world.

2.3 Motivation

One of the driving forces for implementing or finding a solution to this problem is to allow operators and scientist to geographically locate and monitor sensor nodes from a map application using the comfort of their computers. This will help to reduce the financial burdens that could mount on the shoulders of system operators due to them monotonous and periodic data collection from sensor or gateway nodes. The expenses in such a system can increase exponentially with the number of sensor nodes deployed, the geographical divergence in which they are operating, the aggregation schema used to group nodes together, and the periodicity required for collecting data from these nodes.

Another aspect that motivates working on this project and that should be stressed on is the time factor. Most working individuals are bound to a tight schedule, which will hinder their capacity to periodically go to sometimes far locations where the sensor nodes are present to gather information from them. Even if the system aggregates data from all nodes available into one gateway node, this would also consume round trip time. Thus, most involved people, including

me, would be keen about the ability to monitor sensor nodes from home using the comfort of their chairs.

Finally, group working and monitoring often require the involved members to have the same perspective and the same set of files available. Consequently, it would be a waste of time and bandwidth to download the same set of files again by all group members, especially if one of the group members already has those files. Thus, having a distributed group communication file transfer application would both save time and network capacity because all members of the same group would share the same files downloaded by other member belonging to the same group. Now, integrating this application with sensor nodes monitoring, allows the creating of specific specialized groups that could monitor and download data from a set of nodes without exhausting the system.

2.4 Requirements

In this project, a software system or package is needed that is capable of interacting with wireless sensor nodes and end user. This system should act as a mediator between user clients and sensor nodes and should facilitate the interaction between these two components to be as smooth as possible. Literally speaking, the System requirements are as follows.

- A mechanism is needed to gather necessary Localization and functional data from sensor nodes that might be clustered and programmed to work on specific task, or even to work individually.
- Data gathered from sensor nodes should be identified by the date on which it has been collected.
- Localization or geographical data gathered from sensor nodes should be plotted in a Map that shows the location of these nodes. Also, functional data that represent the sensing measurements taken by sensor nodes at specific instants of time should be available to users for view and download.
- Web access should be the main method for visualizing sensor nodes geographical locations and for downloading data corresponding to these nodes.
- Any sensor node added to the system should be available for the user to view in a short period of time.
- Users should be allowed to join groups and share downloaded files that correspond to sensor nodes.

Chapter 3 Design Overview and Specifications

3.1 Introduction

In this chapter the assumptions that should hold before designing and implementing the system are defined, the solution and method adopted for solving this task, as well as an overview of the whole design concept is given, and the specs of the proposed solutions are presented.

3.2 Assumptions

The following conditions and assumptions must hold prior to the design and usage of the sensor nodes visualizing and monitoring system developed.

- Each sensor node is required to have a unique Id that differentiates it from other nodes in the system.
- Each sensor node knows it location and can provide it along with data gathered in a predefined XML format. The localization schema used by sensor nodes is beyond the scope of this thesis work, and is not discussed further.

3.3 Solution and Methods

For solving the problem of remote monitoring and data gathering from sensor nodes a front end server abstraction should be built to accept connection from sensor nodes, gather the necessary data, and store it in a suitable storage format (database) for future retrieval via http, or ftp requests by remote clients. Also, replica managers should be available that connects to the front end server periodically and polls it for the latest data it has acquired from sensor nodes. The front end server and replica managers should store data in suitable format that will be used by a web application, which has the task of updating a map with markers that represent the geographical location of each sensor node. Also, the web application should allow the user to download measurement or sensing data from each sensor node on the map. The web application should reflect the latest changes in the front end server database by polling the server periodically for the latest data it has without reloading the page and disturbing the user. Finally, an algorithm is going to be developed and implemented to allow reliable group communication for sharing downloaded files between group members in a reliable manner.

In designing phase of this project the following programming and scripting languages, and technologies will be used: Java, C#, JavaScript, html, Ajax, and ASP.NET.

- Java programming language will be used to build a server that accepts connections from sensor nodes, a replica manager that replicates the server's database, and the distributed group communication file transfer application.
- JavaScript, html, Ajax, C# and ASP.NET will be used to design the web application or the home page that client's browsers can load to view sensor nodes and download stored data.

3.4 Design overview

As I always say "what words cannot say figure might clearly display". For this reason 4 figures are presented below that give a bird's eye view over the working environment and a rather general idea of different solution components involved in this task.

The first figure(3.1), represent one of the solution components which is an xml data acquisition server (XDAS) that listens for connection from sensor nodes, possibly gateway nodes, and gathers data from them in xml format. Also the figure shows a geographical area over which the sensor nodes are dispersed. The area over which sensor nodes are deposited is not important for this thesis work, and this project starts by the assumption that sensor nodes are already there and are sending data to the server as shown by stars labeled with one. The server gathers the xml files from sensor node and then processes this data in a manner that make it available for future retrieval by connecting clients.

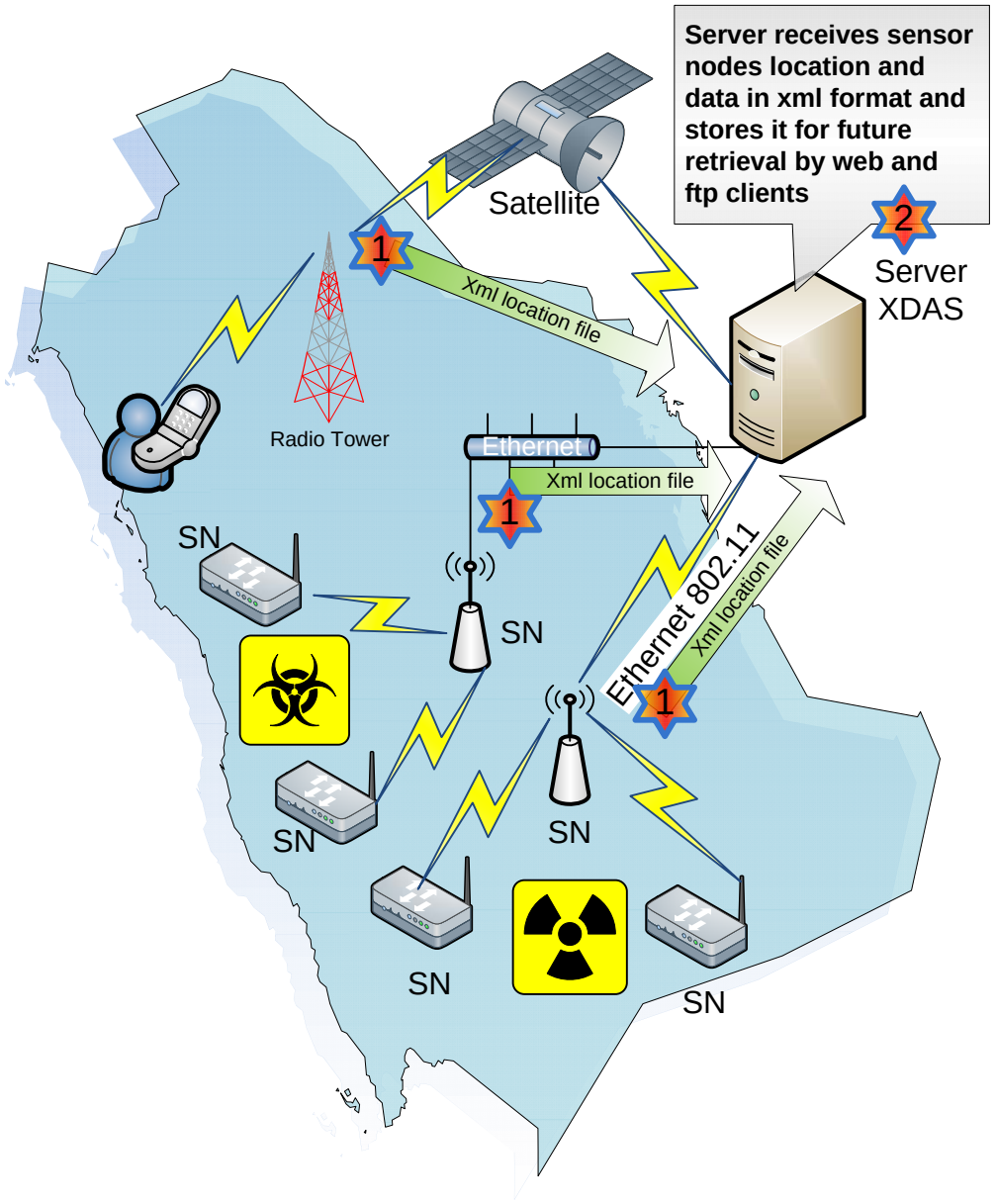

Figure 3.1: Server receiving xml data from sensor nodes

In figure 3.2, and after the server receives and processes the data from sensor node, it makes it available to web clients, replica managers, and ftp clients. The description of each client role as well as the replica managers will be given in the subsequent chapters.

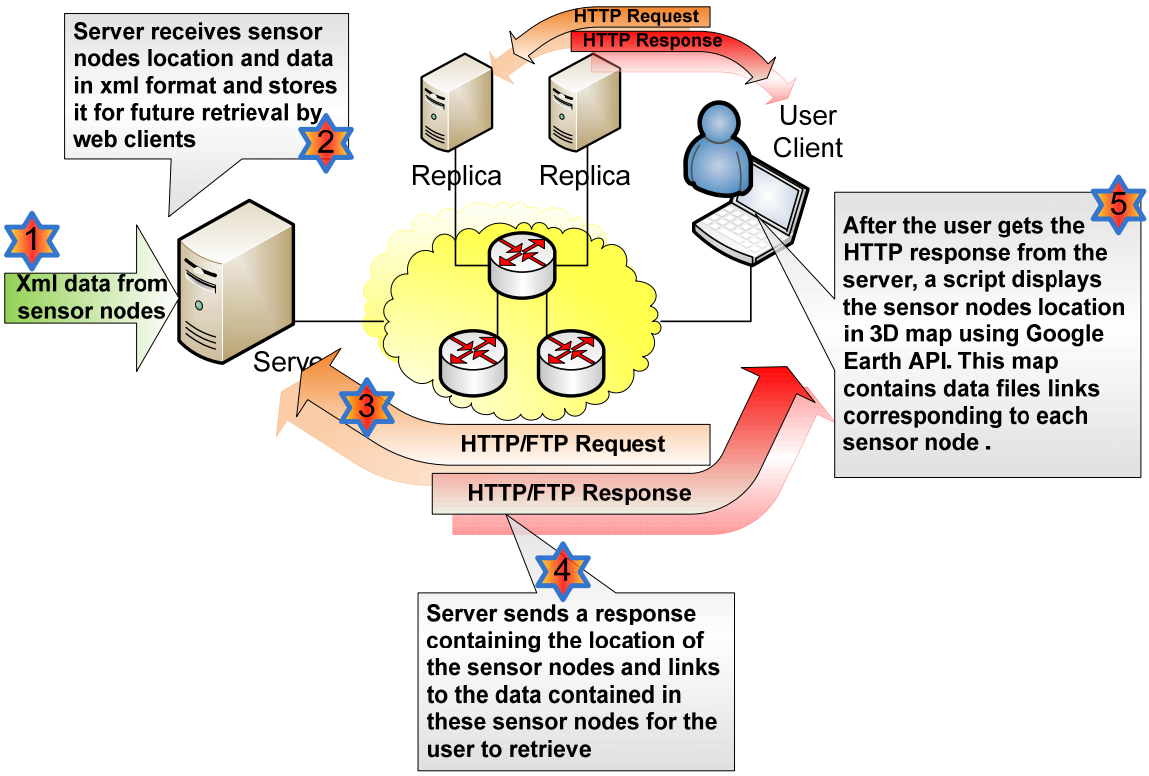

Figure 3.2: Web clients interaction

The third figure (3.3) is the general picture that combines the former two figures. The only thing that is not clearly mentioned in the previous figures is the distributed group communication file transfer protocol, which will be illustrated in fourth figure and subsequent chapters.

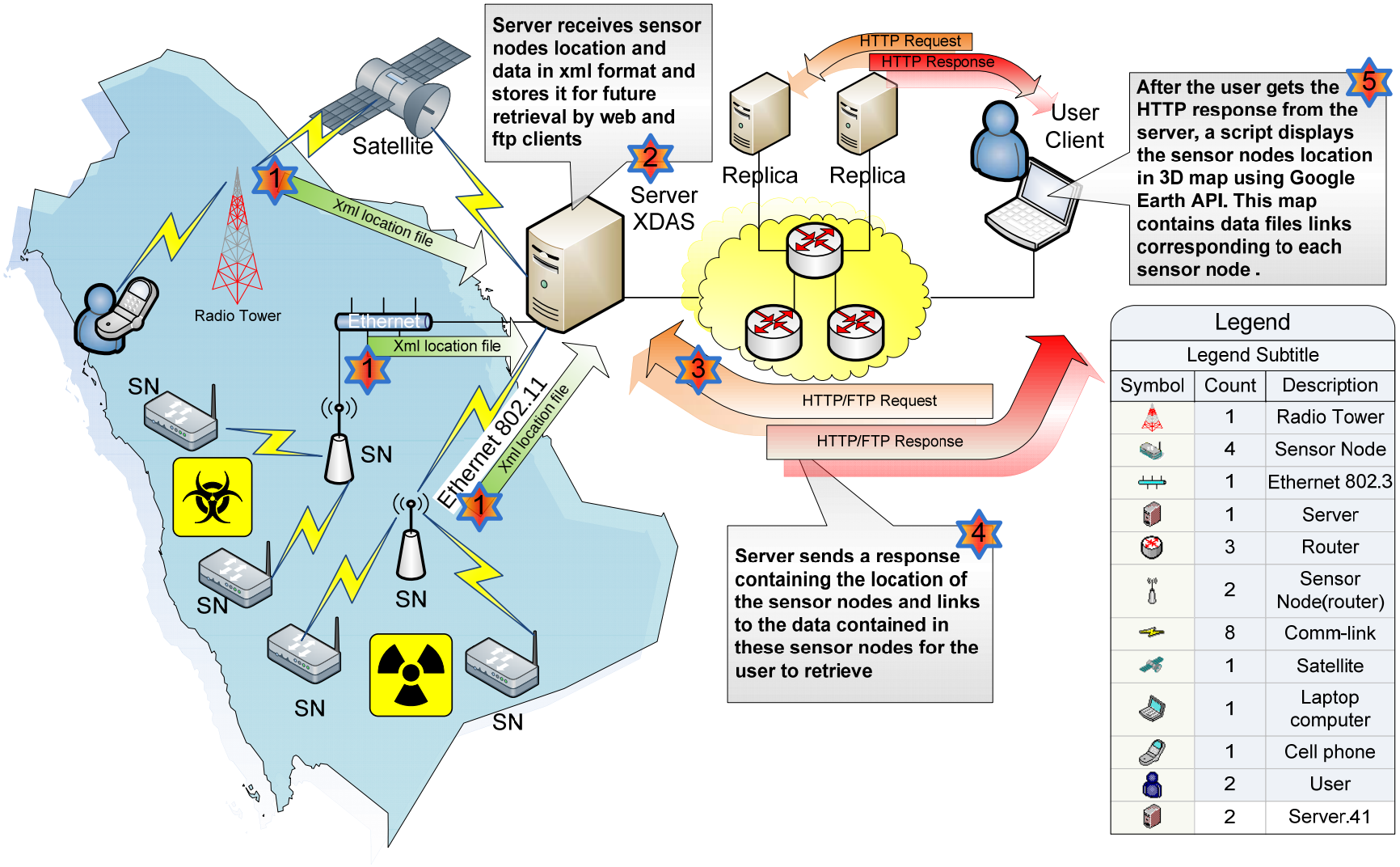

Figure 3.3: General picture

The fourth figure (3.4) shows the distributed group communication file transfer mechanism. Shortly, users join a specific group, download sensor nodes related files from the server and share these files with processes or other members of the same group.

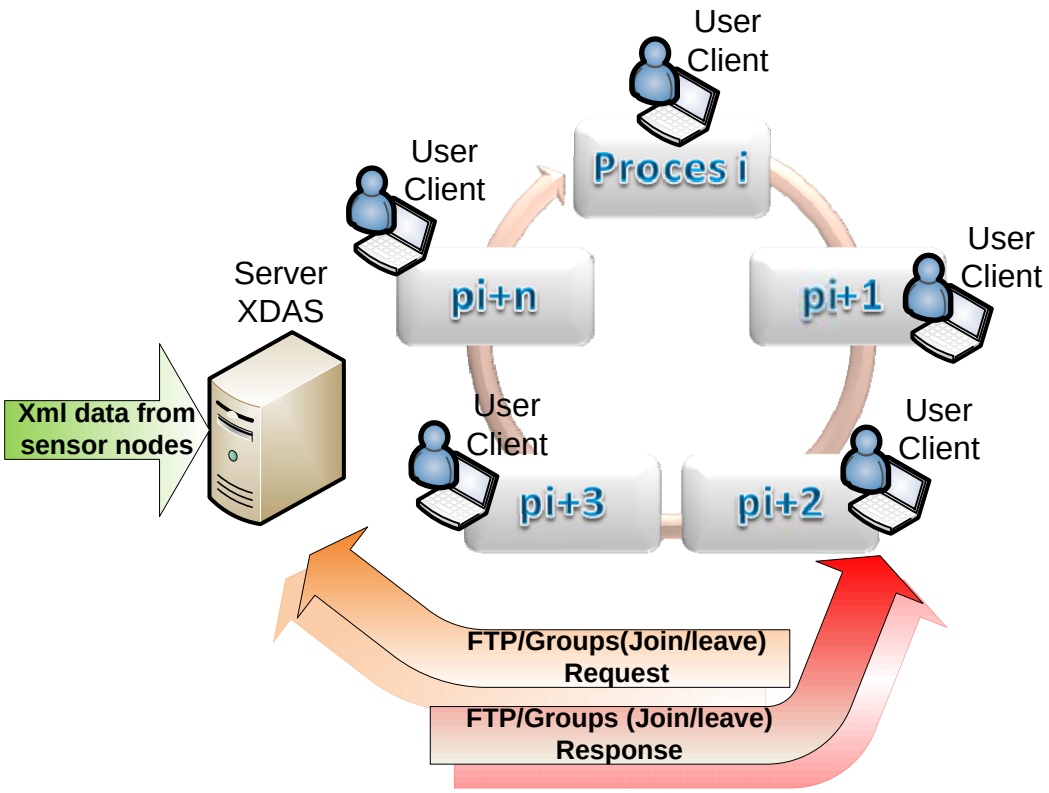

Figure 3.4: Distributed group communication file transfer mechanism

3.5 Design Specifications

The specification of the system developed to monitor sensor nodes are given in the list below:

- A server listens continuously for connections from sensor nodes and gathers data from them in XML format, tags the data with a timestamp that identifies the collection date and stores it in a node's specific file.
- Each sensor node has a specific file associate with it that contains gathered data from that node. The syntax of the file name is SensorXXX.sn where XXX represents the unique id of the sensor node.
- Availability is taken into account when designing the system and some of the data that was previously accessible by web clients will continue to be available indirectly through replica managers in case the system's main machine has failed.
- Any sensor node added to the network will be available for display in map within a short period of time that is less than 10 seconds.

- Nodes geographical locations as well as data corresponding to these nodes are available for the users to visualize and download respectively using 3D Google Earth API through web clients or browsers.
- Download of corresponding sensor nodes data files (SensorXXX.sn) is allowed through a separate distributed group communication file transfer client.
- Using a distributed file transfer application, a user is allowed to join one group at a time, and to share downloaded files with group members.
- Groups are can scale by allowing user to join or leave the group without disturbing the functionality of the group.

Chapter 4 XML Data Acquisition Server

4.1 Introduction

This chapter describes the functionality of the xml data acquisition server, which is one of the software components developed specifically for this project. This server listens for connections from sensor nodes, gathers data, and stores it for future retrieval by web clients. In Addition, the server accepts connections from replica clients that replicate the server's database contents on another machine across the web.

4.2 XML Data Acquisition Server

4.2.1 Sensor Nodes Data Acquisition

A server listens continuously for TCP connections from gateway nodes or any sensor nodes that are equipped with TCP/IP stacks, and follow the assumptions mentioned previously. Sensor nodes send data to the server in XML format as shown in the figure below.

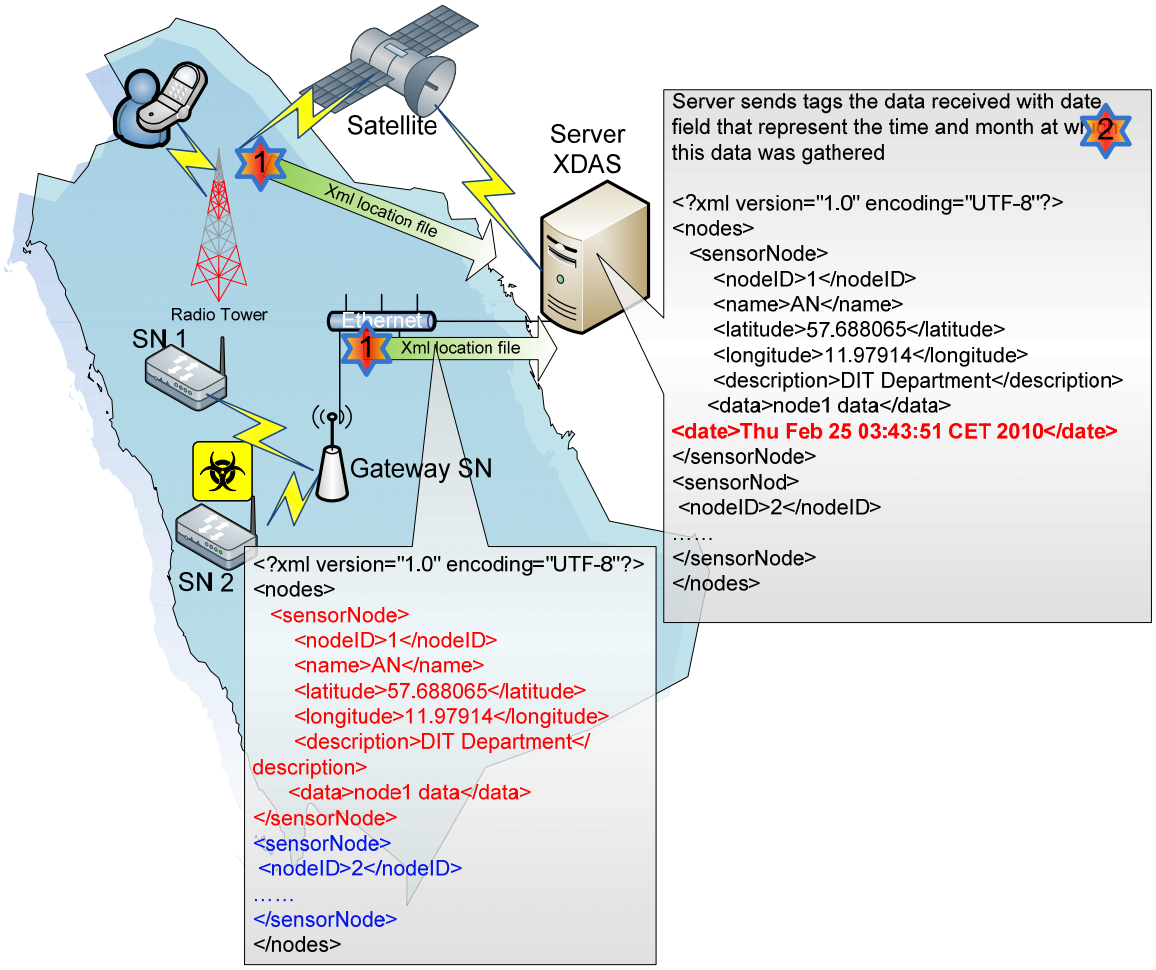

Figure 4.1: Sensor nodes sending xml data to server

In the figure 4.1, a gateway node is aggregating or collecting data from sensor nodes sn1, and sn2 respectively. The gateway node formats that data in a predefined XML schema and then sends this data to the server which tags it with a date field corresponding to the time and month at which this data was received. The XML schema used has the following tags:

- **<?xml version="1.0" encoding="UTF-8"?>**: represent the current xml version and the encoding used
- **<nodes></nodes>**: these are the root tags which encapsulate data from a group of sensor nodes. Each time a sensor node connects to the server it must send data that begins and ends with this tag as show in the figure above.
- **<sensorNode></sensorNode>**: these tags represent a particular sensor node and thus encapsulate data corresponding to that node. In the figure above the gateway sensor node, Gateway SN, is collecting data from sensor node one (sn1), and two (sn2) into two corresponding pairs of sensorNode tags colored red and blue respectively.

- **<nodeID></nodeID>**: These tags hold the unique ID of a sensor node. As has been mentioned earlier, each sensor node should have unique id for the system to in function properly in a deterministic manner.
- **<latitude></latitude>**: encapsulates the latitude of the node's geographical location.
- **<longitude></longitude>**: encapsulate the geographical longitude of a node.
- **<description></description>**: encapsulates a text describing the purpose of this sensor node. For example, one could say that this sensor node is used to monitor the border against intruders.
- **<data></data>**: encapsulates the sensed data that this node has sampled from the surrounding environment or specific application.
- **<date></date>** : these tags are only added by the server when it receives data from a sensor nodes. It encapsulates the date at which this data was collected.

Data transmitted between sensor nodes and the server is formatted in XML schema as shown earlier. The Extensible Markup Language (XML), which is a simple text-based format for representing structured information, was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use (13). Some of the properties of XML are as listed below (14):

- XML is a markup language much like HTML: HTML was designed with the focus on how data should look and display, however, XML was designed to focus on the data itself and its transportation. Thus, XML was designed to carry data, not to display data.
- XML tags are not predefined: A user needs to define his/her own tags.
- XML is designed to be self-descriptive
- XML is a W3C Recommendation
- The syntax rules of XML are stricter than HTML: All elements or tags must be closed or marked as empty to be processed correctly by XML tools.

If this short introduction is not sufficient, and because this paper is not and XML tutorial, the reader is advised to refer to (13) and (14) for further information.

4.2.1.1 Data processing

Once the server receives and tags the XML data from sensor nodes, it stores this data in several files as show in the figure below.

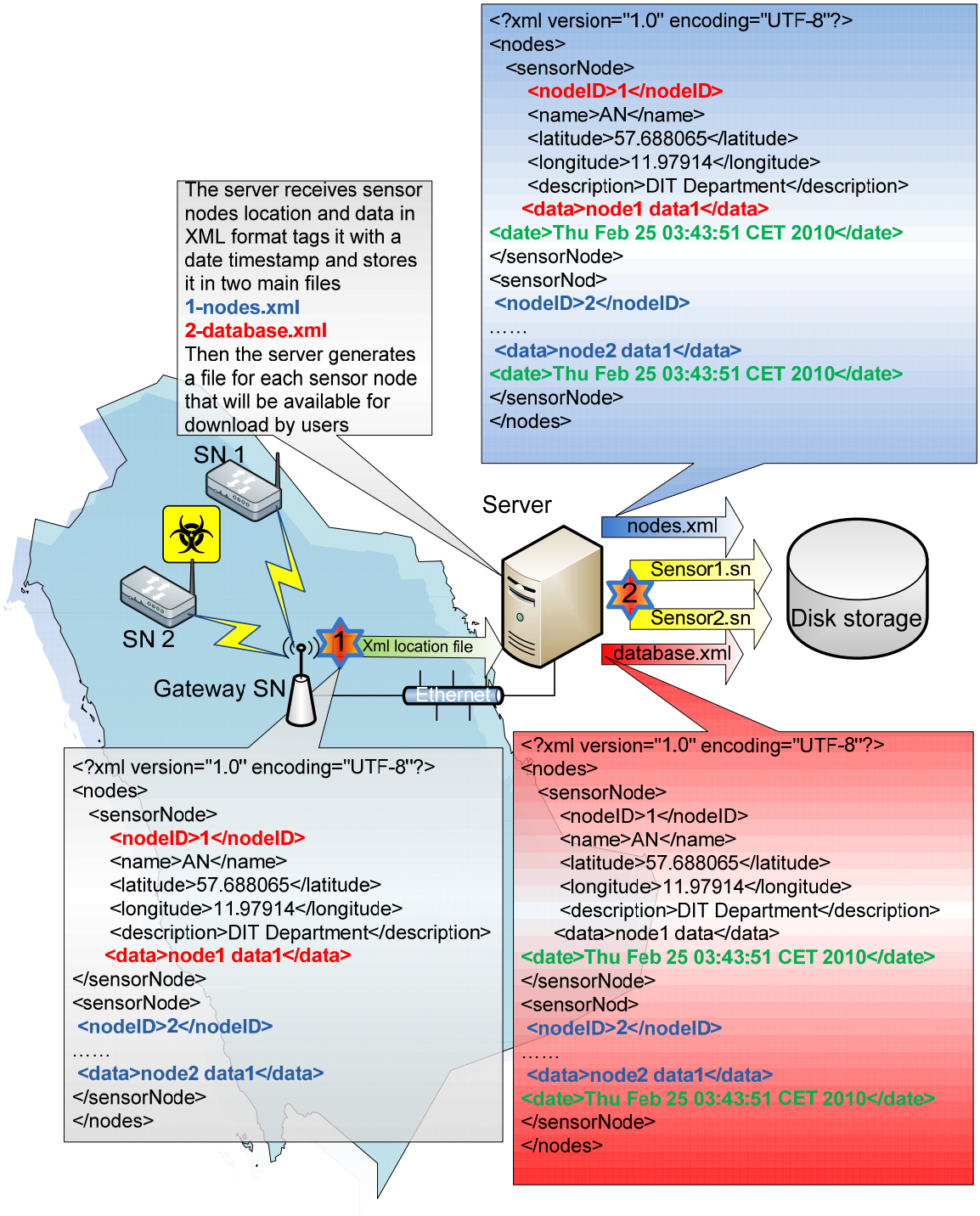

Figure 4.2: Server stores data to a database

Figure 4.2 shows that after the server receives sensor nodes location and data in XML format and after it tags that data with a date field, it stores it in a file hierarchy as described below:

- **Nodes.xml**: Contains the latest copy of XML data received by the server, which corresponds to each node in the system. Thus each XML sensorNode tag in this file represents a unique sensor node that has a unique ID and contains latest data generated by that node. This file is used by a web server as will be described later.
- **Database.xml**: Contains a copy of all the XML data received by the server up to this point from all sensor nodes in the system. Thus a single sensor node might have several corresponding <sensorNode> tags related to it as can be seen in figure 4.2. This file is will be used by replica clients as will be described later.
- **Sensor<nodeID>.sn**: For each sensor node represented by sensorNode tags that has connected to the server using a unique id, a file is created that corresponds to that node. This file stores all the data, in text format, sent by the node up to this point.

Figure 4.3 is a comparison between the content of nodes.xml versus database.xml, the assumption in this figure is that sensor node 1 connects to the server and sends data for the first time then node1 & 2 connects to the Server and send their data. As can be seen, the database.xml contains all the data that has been received from node 1 and 2, however, nodes.xml contains only the chronologically latest value sent by node1 and node 2.

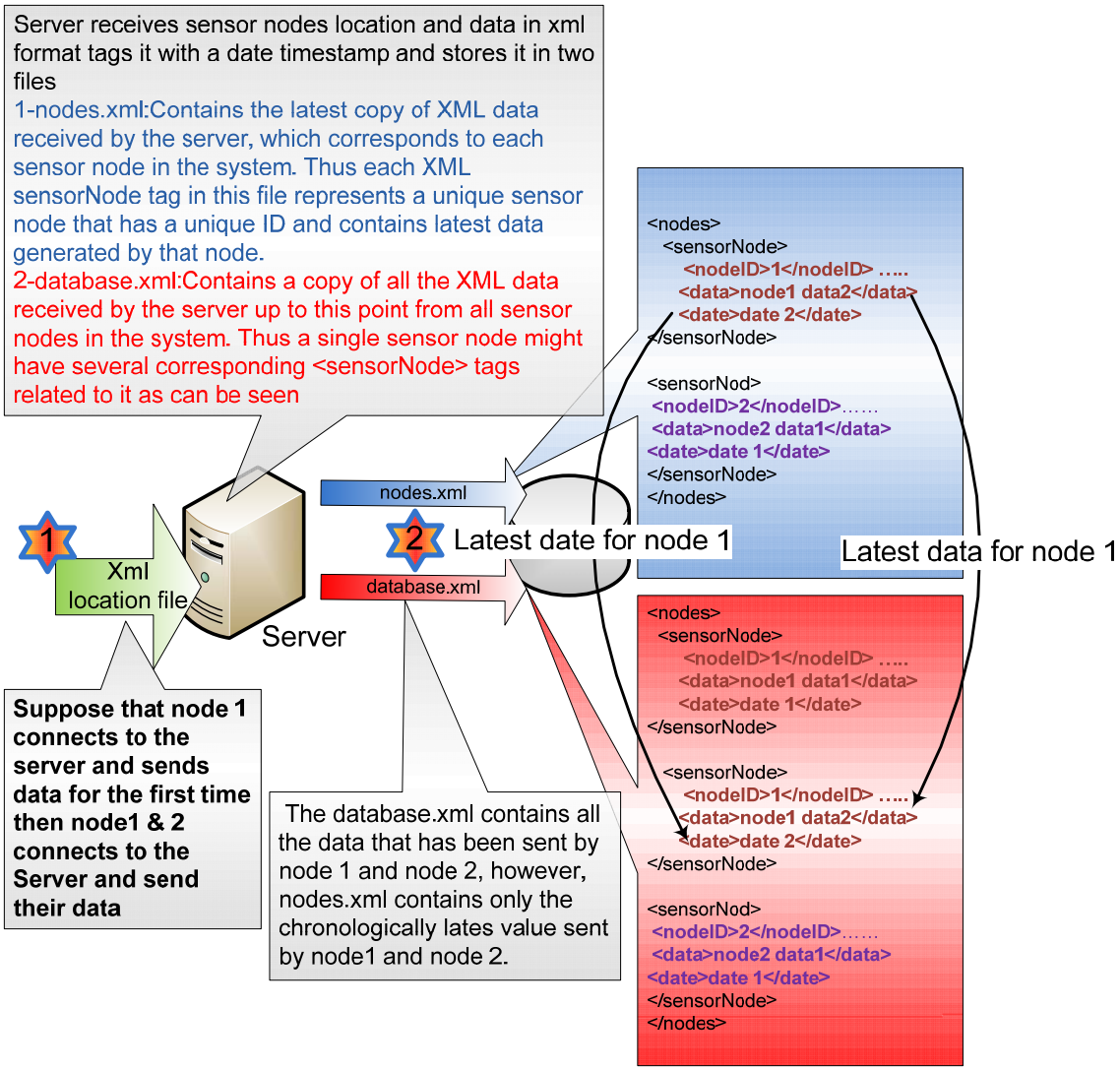

Figure 4.3: database files comparison

4.2.2 Replica Server

The Server also listen for connections from replica clients and once it receives and XML request from those it sends the latest version of the database file stored on the server machine that represent data received from sensor nodes. The functionality of the replica client as well as a complete description of the connection process is described in the subsequent replica client section.

4.2.3 File Transfer Server

Another functionality embedded in the Server is the multiple files transfer mechanism. When working in this mode the server can send a list of files available for download as well as files

requested for download by users. This concept will be further explored in the distributed group communication file transfer application (DGCFTA) chapter.

4.2.4 Groups Server

The last function that this server handles is establishing groups where users can request to create a group, join a group, and leave a group. This functionality will be thoroughly described in the DGCFTA chapter.

Chapter 5 Replica Clients

5.1 Introduction

In this chapter, another software component of the project called replica clients will be investigated. Shortly, the principle behind replica clients is to increase the availability of the system by allowing data present on xml data acquisition server (XDAS) to be replicated on another machine somewhere else across the web.

5.2 Purpose behind Replica Clients

To increase the availability of the system in case the main server that receives connections from web client has failed, a set of alternative web servers should be available to handle connections instead and to allow users to access at least an older version of the data that was previously available before the main server has failed. These set of web servers should have replica clients that connects periodically to the main server, request updates, and replicates data on the servers in a form presentable to the web clients.

5.3 Operation of Replica Clients

Replica clients connect to the XDAS periodically by sending <replica></replica> tags as can be seen in figure 5.1. The reasons behind using XML tags when requesting updates are:

- **Design preference**: the main server is designed to interpret XML request only.
- **Future Security Implementation**: Security such as authentication and identity check of the replica clients that have permission to replicate data can be implemented by adding the necessary security tags inside the replica root tags. In this project, however, security and its implication, which could be severe, are not considered and thus are not discussed further.

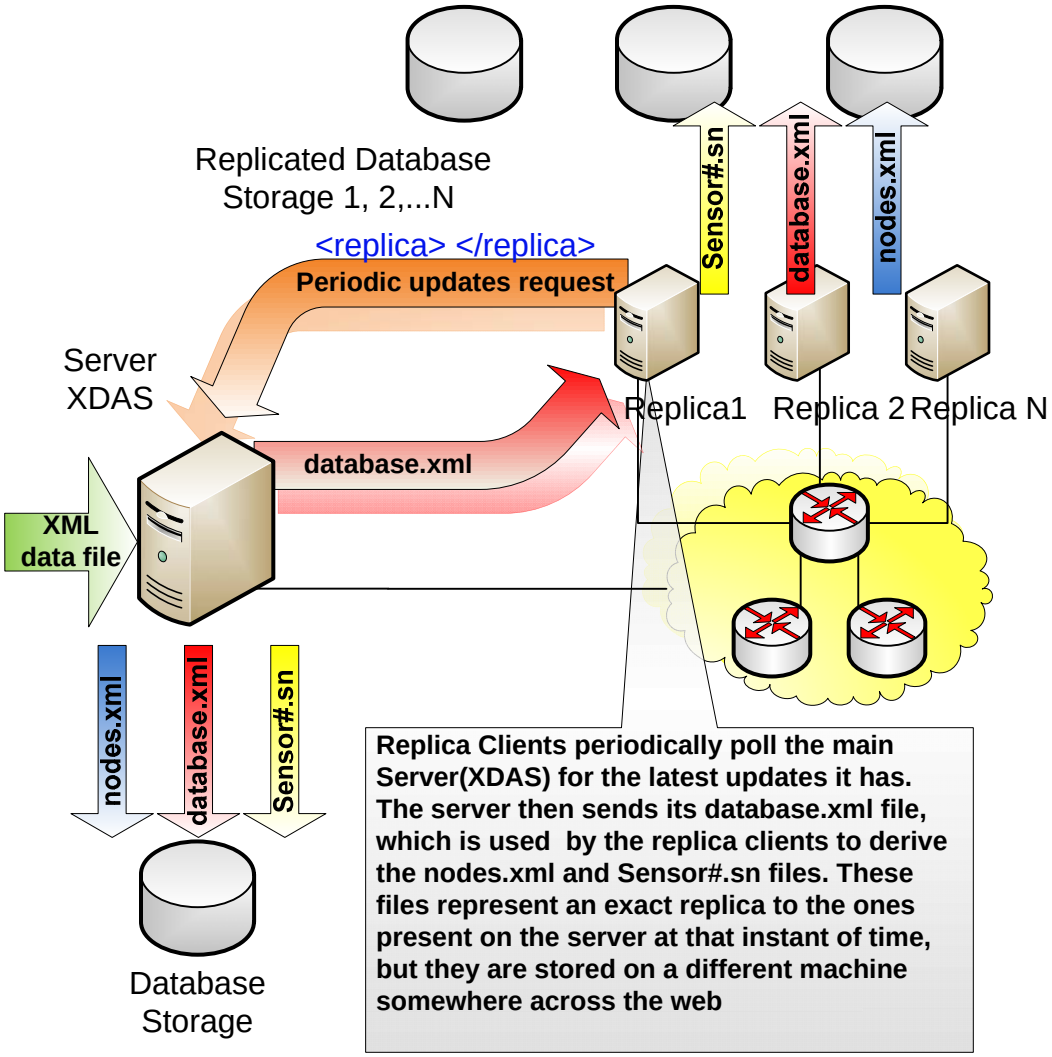

Figure 5.1: Replica clients requesting updates

5.4 Replica managers vs. Replica clients

Replica client are used to increasing the system availability in case the XDAS machine failed. Thus replica clients' machines along with the XDAS's machine are intended to run as a distributed web servers machines as well can be seen in the figure 5.2.

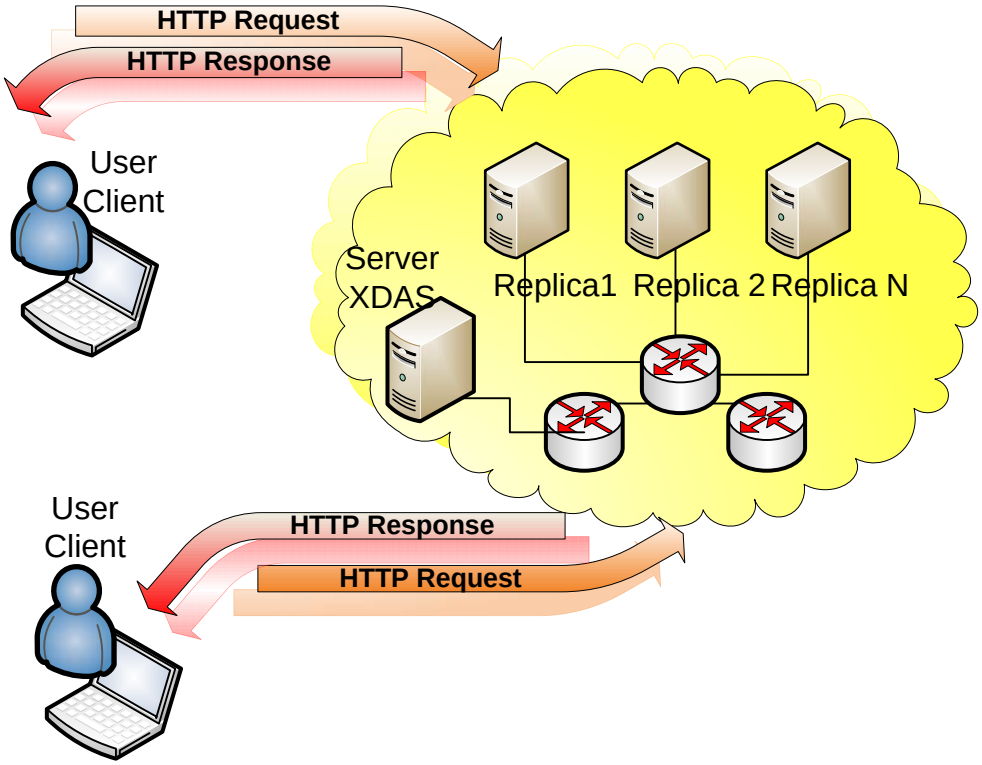

Figure 5.2: distributed web server

In fully fledged replica managers as opposed to limited replica clients develop in this project, there should be a front end (FE) that receives connections from clients, sensor nodes in this case, in FIFO manner and then forward these request to replica manager(s) according to a specific design model or algorithm. Referring to (15) there are two general models of replication:

- Passive (primary backup) replication: In this model, the FE communicate with only the primary replica manager which takes the request, executes it, delivers the service to the FE, and send copies of the updated data to all the secondary replica managers that exist if the request was an update request. One of the secondary managers is promoted as primary replica manager if the primary failed.
- Active replication: In this model, replica managers are organized in a group and are considered to be as a state machine, which means that there state is deterministic and depends on the set of operations that they have been executed. The FE sends or multicasts reliably ordered request to all replica managers in a group, then all managers within that group process the request identically but independently and send their replay to FE, which selects one reply and delivers it to the user. This model achieves consistency among all the replica managers because of the assumed reliable and totally ordered multicast and because replica managers are considered as state machine, which

mean that they have a deterministic output when executing the same set of instructions or requests.

Of course, there can be other models of replication such as gossip architecture, but discussion will be limited to the formerly mentioned models. Furthermore, request by users to the FE can be either a query or an update. A query is simply a read request or operation, and update is a state modification request without any read. In this thesis, sensor nodes can only execute update request directed only to the xml database acquisition server (XDAS), which communicates with other replica clients.

Due to the fact that replica clients idea was an addition, not a requirement, and due to the fact that a group communication algorithm will be implemented in chapter 5, which could be used as a replication algorithm, the design of replica clients is made as simple as possible and do not strictly follow any of the formerly mentioned models for replication. Thus there is no true Front End that get request from sensor nodes and forward it to replica clients. However, the XDAS can be still thought as an intelligent front end that accepts data from sensor nodes, processes it on the same machine, and make it available for replication on other machines through replica clients.

The other question is then from which machine does the user gets the http response? The answer to this question is that replicated data at several servers should be accessed by binding all the web servers IP addresses to the DNS (Domain Name System) of the project site or home page. Doing so will serve two purposes:

- Firstly: The workload between all the servers will be shared, since each DNS lookup of the site result in the return of one of the several IP addresses mapped to the domain name. The randomization process of selecting IP addresses to return depends on the design of the DNS
- Secondly: Availability is achieved by allowing the user to access other available web servers if some of the replica clients' machines failed. The detection of the failure of a server in this case should be implemented at the DNS level, which should return IP addresses for site domain name lookups that corresponding active servers only.

Chapter 6 3D Web Application

6.1 Introduction

In this chapter, the 3D web application that allows the user to access sensor node information at the web server is going to be described. This application maps a sensor node location to a map geographical location, and allows the user to access sensor node related files at the server.

6.2 Web Server

After the XML data acquisition server receives connection from sensor nodes, it stores their corresponding location and data in a web directory for future retrieval by web clients. Of course, there should be a web server running on the XDAS's machine that exposes the contents of the web directory to public clients.

6.3 Web Access

From the comfort of his/her browser, a user can access and retrieve data related to sensor nodes from either the XDAS machine's web server or the replica machine's web servers as shown in figure xxx.

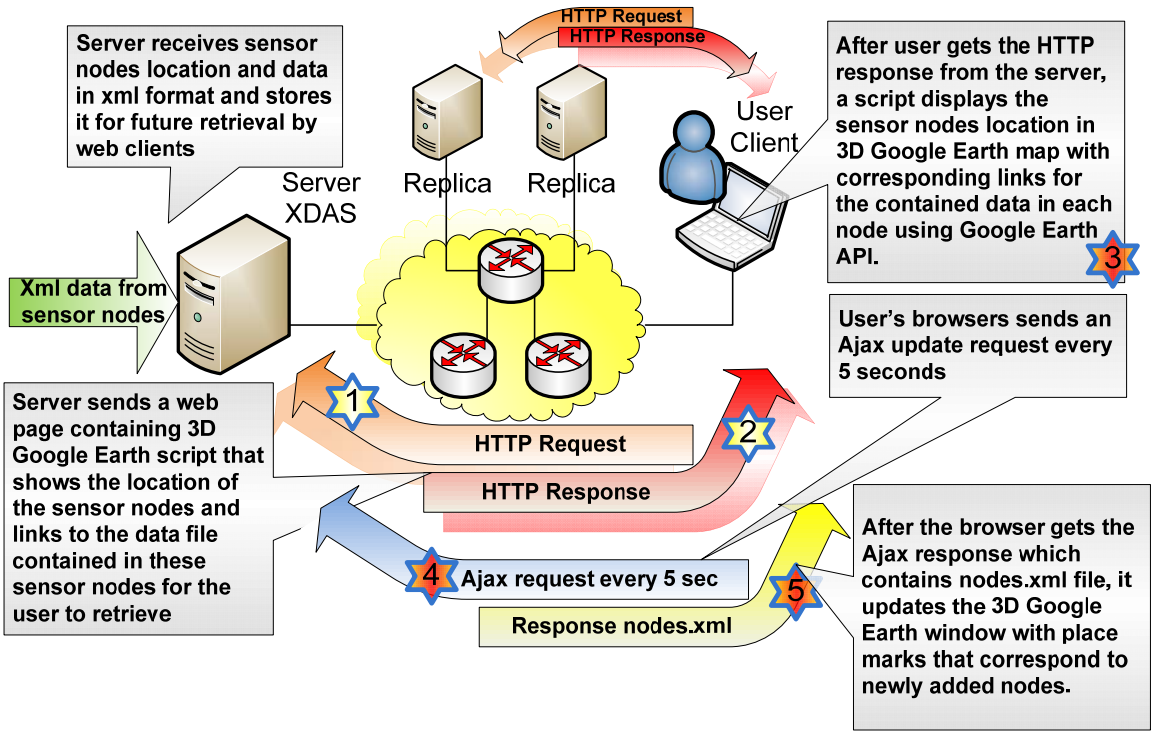

Figure 6.1: Web access

According to the figure above the following sequence of events happen when a users enters the web address or URL of the XDAS machine into his/her web browser:

1. The browser sends an HTTP request containing the URL the 3D web application page application located on the server.
2. After the server receives the HTTP request it sends an HTTP response that contains the request web page.
3. After the user's browser gets the HTTP response from the targeted web server, a script in the responses issues an Ajax (Asynchronous java script and XML) request to the server and uses the data in the response to display the sensor nodes location in 3D map using Google Earth API technology.
4. The 3D Web Application running on the client's machine keeps the updating itself automatically every 5 seconds by sending Ajax update request to the server. Ajax is used instead of HTML requests in order not to disturb the user by reloading page every 5 seconds. Thus the user will not feel any discrepancy from page reloading during his/her browsing experience. This periodic request is made to insure that any sensor node added to the system or the server side will be available to web clients to view within a short period of time that is less than 10 seconds.
5. After the user's browser gets the Ajax response from the server which only contains nodes.xml file, it updates the 3D Google earth map with place marks that correspond to any newly added nodes to the system.

6.4 3D Web Interface Interpretation

The 3D web interface is an instance of and has the same functionality as Google Earth (16) (17). Consequently, it is very easy to use the interface especially if the user is already familiar with using Google Earth. Thus, this section will only describe how sensor node geographical information is mapped into 3D web application.

In the 3D web application each sensor node is mapped to a place mark that corresponds to its geographical location. This concept of mapping is illustrated in figure 6.2. In addition to viewing the geographical location of a sensor node, the 3D web application shows in a popup window downloadable data files corresponding to a particular sensor node. These node's specific files are stored at the server and are derived from database.xml. Also these files are stored using names that follow the syntax of Sensor#.sn, where # represents the unique sensor node id.

According to the figure 6.2, once XDAS receives connection from sensor node 64 and 1, it stores their corresponding geographical location in nodes.xml and their sensed data in Sensor64.sn and Sensor1.sn respectively. A 3D web application running by a web client then maps the location of each sensor node to a geographical place mark, and for this purpose the information found in nodes.xml file is used. This file is updated every 5 seconds as described earlier using an Ajax request.

In each place mark, there is an embedded html popup window that is invoked by clicking on a specific place mark. This window contains the following information respectively.

- Node id: Represents the unique id of a node.
- Update date: Is a node's chronological time stamp that specifies when a node data file has recently been updated at the server.
- Data file: is a hyperlink that correspond to each node's data file stored at the server. For example, if the figure blow the 3D web application shows the html popup window for node 1, where Data hyperlink link points to the url of Sensor1.sn (node's 1 data file) at the server.

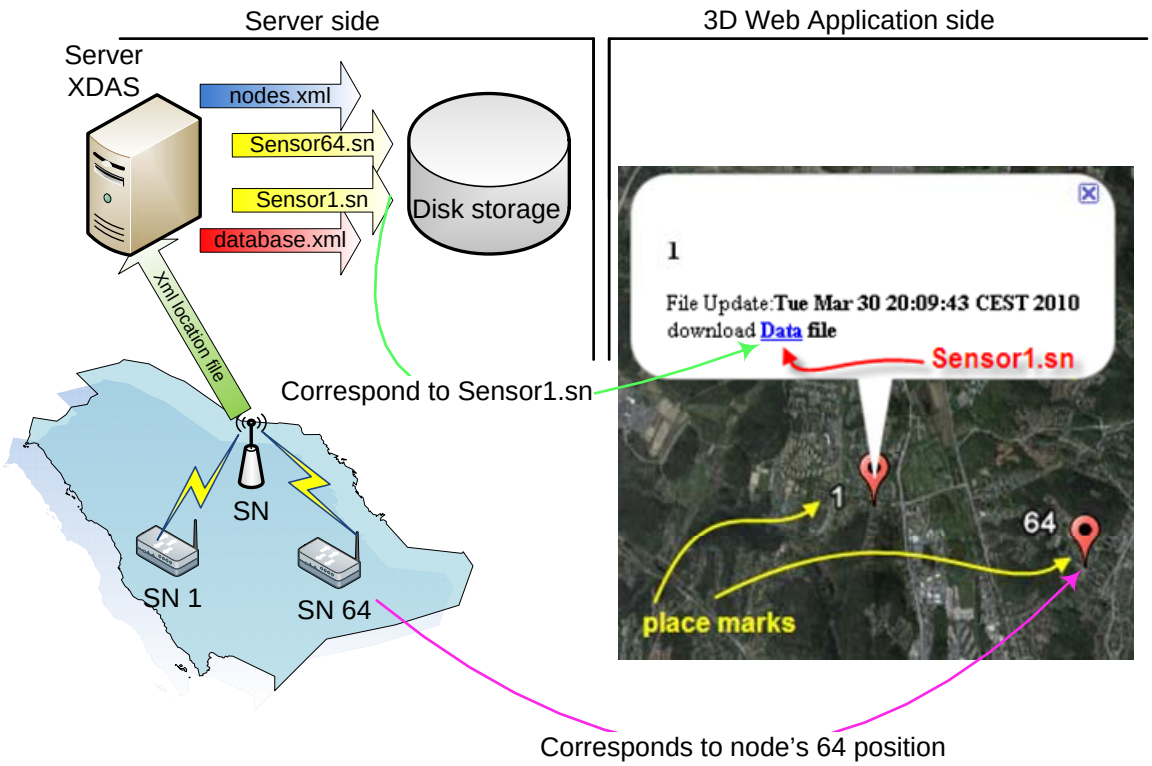

Figure 6.2: Mapping between placemarks and sensor nodes Locations

Chapter 7 DGCFTA

7.1 Introduction

In this chapter the system model, specifications, and the algorithm behind the Distributed Group Communication File Transfer Application (DGCFTA) will be described thoroughly. Also, the message communication header(s) that are exchanged between processes is going to be unveiled and described along with the implementation and guarantees. Finally, a detailed scenario of executing the algorithm over three processes is going to be illustrated.

7.2 System model and Specifications

The system model of the distributed group communication file transfer application developed in this thesis work has the following specification.

- A group of processes in the system communicates reliably with each other over one-to-one channels.
- Processes communicate in a group by building a directed ring network overlay graph where each process communicates with exactly 2 processes and messages are sent in clockwise fashion within the group as show in figure 7.1.
- Multicast of message m to all group members is achieved through a reliable unicast schema where each process forwards m in a clockwise fashion to its left neighbor process in the topology. For example, in figure 7.1, if pi+3 wants to send a message to pi+1, then pi+3 sends the message to pi+2, which forwards it to pi+1 and so on.
- A process that sends a message is identified by unique identifier contained in the message.
- A group is scalable which means that any processes is allowed to join or leave a group
- A group is closed which means that only processes within the group can send messages directly to the group. Processes inside a group, however, can communicate with outside processes, download data from them, and make this data available to all group members indirectly.
- There could be any number of separate groups in the system.
- A process is allowed to join one group at a time.

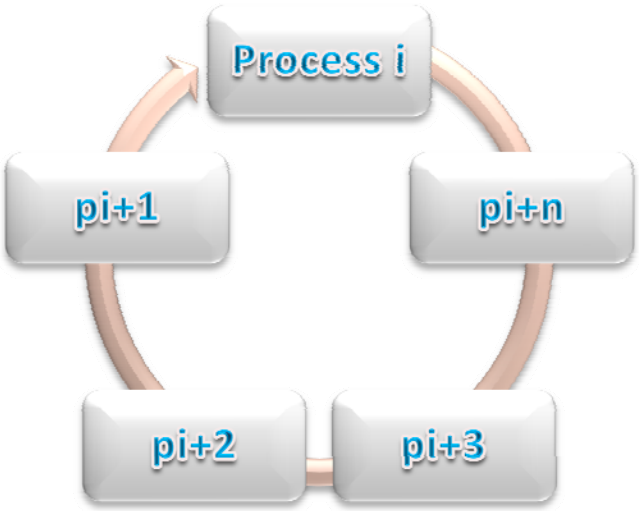

Figure 7.1: Directed ring network overlay

7.3 Message Communication Header

Processes communicate with each other by exchanging messages. Each message sent and received should have a predefined header as shown in table 1.

Table 7.1: Message communication header

Message Communication Header			
SenderID	ForwarderID	MessageText	FilesMap
GroupMapper	ProcParam	Status	

The description of each field contained in the message communication header is given in the list below.

- **SenderID**: Is the unique ID of the process that has initially sent the message communication header.
- **ForwarderID**: Is the unique ID of the process that has forwarded the message communication header. Usually the sender id is different than the forwarder id except in the case where the process sending the message is the original initiator of that message.
- **MessageText**: A variable String of characters that contains private messages between group members.
- **FilesMap**: Is an object that maps between a file name and its length in bytes. The purpose of this field is to tell the peer process about files that are available

for download and the length of each. The exact fields contained in this object can be seen in figure 7.2.

- **ProcParam**: Is an instance of a processParameter object which contains fields that identify a process parameters such as:
    - **Process Name**: Is the unique name or ID of the process and is assigned or obtained from the XML data acquisition server when the process joins a group.
    - **Host Name**: Is the name of the host machine in which the process is running.
    - **Host IP Address**: Is the IP address of the process's host machine
    - **Port Number**: is the port number of the process's server thread.
    - **Group ID**: is the name or ID of the group that the process belongs to.

- **GroupMapper**: Is an object that is used to manage group members. It is used to map a set of processes to a specific group as can be shown in figure 7.2. Also, it is used to manage the addition and removal of process to and from groups respectively. For example, if a process wants to join a group, an entry corresponding to that process's procParam object is added to the appropriate group in the groupMapper object. The pocParam object is added to a group that is specified by the group id field contained in that object.

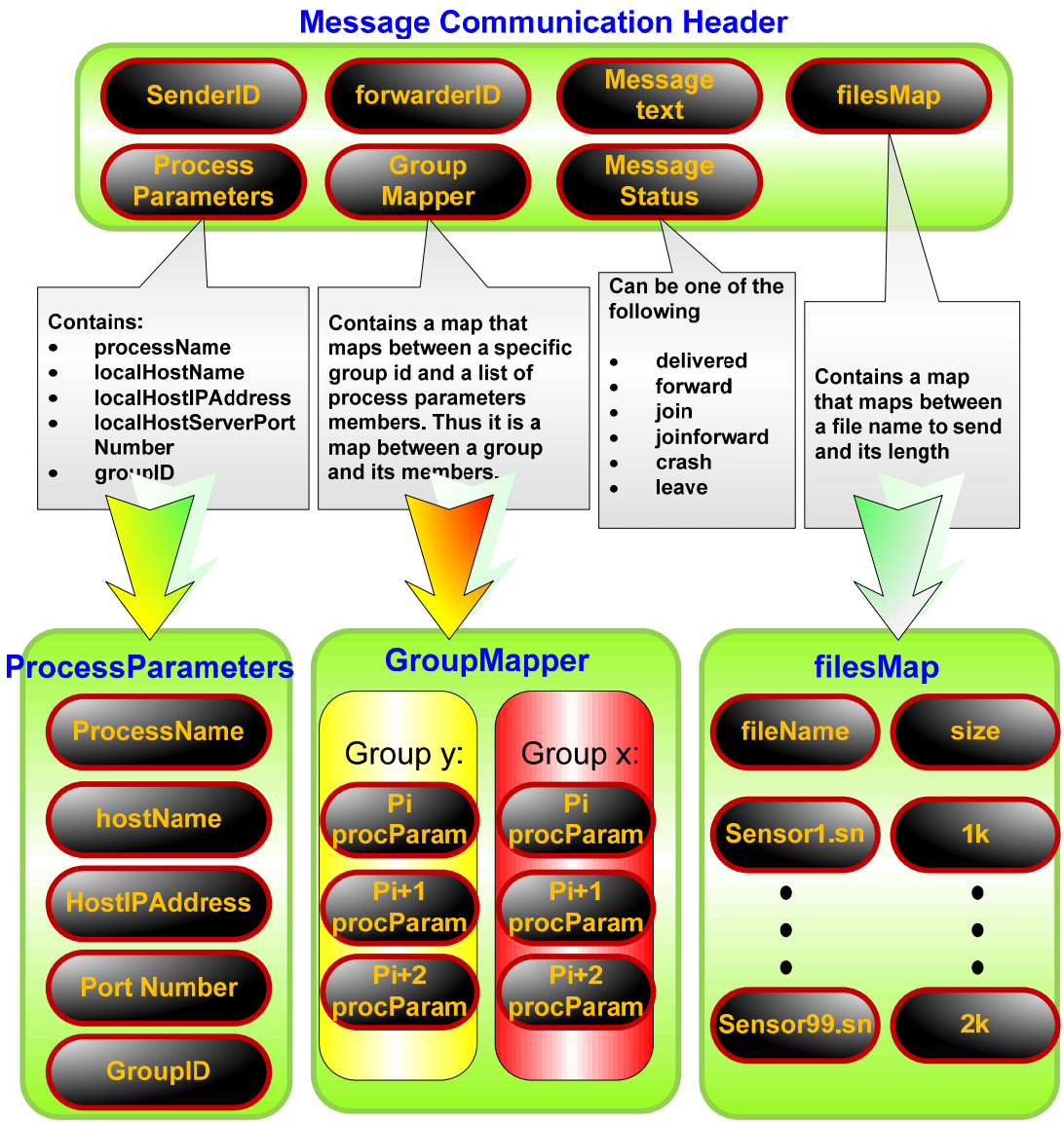

Figure 7.2: Communication headers

7.4 Distributed Ring Algorithm

In this section the algorithm behind the distributed group communication application is going to be investigated, but first the assumptions that should hold for the algorithm to function properly are given.

7.4.1 Assumptions

In describing the algorithm the following assumptions are made:

1. Processes in the same group have unique IDs.

2. Each message exchanged between group members should be marked with a status field that can take one of the following values:
    a. delivered
    b. forward
    c. join
    d. joinforward
    e. crash
    f. leave
3. All processes know about all group members and manage those members using a groupMapper object, which maps a member to a group.
4. Each process maintains a deliveredQueue and downloadedQueue objects. The first object maintains the set of files that were recently downloaded by the process, and the second object contains all the files that the process's group has.
5. The Algorithm uses a directed ring graph or network overlay as was shown in figure7.1, where messages sent in the group traverse the overlay in a clockwise fashion.
6. Each process has reliable TCP connections to its left and right peer process or neighbors that are before and after the process in the overlay ring.

7.4.2 The Algorithm

The general flowchart of the algorithm is given in figure 7.3. In this figure, process pi+1 is assumed to be running the algorithm, and according to the flow chart when pi+1 receives a message it checks the status of that message and handles it accordingly. The only missing parts in the flow chart are the handling mechanisms which are as described below:

- **Join Handling:** When process pi+1 receives a join message:
    1. It downloads the files in contained in the message from the sender, if any.
    2. It adds the downloaded files, if any, to the deliveredQueue and downloadedQueue objects.
    3. It adds the Sender Process to the group by adding its procParam instance to the groupMapper object. If pi+1 is the first process in the group it connects back to message sender to form a ring.
    4. It Sets the received message's status to "joinforward" and message's forwarder id to pi+1, Sends it to it left neighbor, probably pi, and then the clears or empties the deliveredQueue object.

- **Joinforward Handling:** Same as delivered Handling, except that the message received is joinforward instead of a join, and if the receiving process detects that it is the original sender of the message, then it drops the message instead of forwarding it to the neighboring process.

- **Delivered Handling:** When process pi+1 receives a delivered message, it:

1. It downloads the files in contained in the message from the sender.
2. It adds the downloaded files, if any, to the deliveredQueue and downloadedQueue objects.
3. It Sets the received message's status to "forward" and message's forwarder id to pi+1, Sends it to the process directly above it in the group, probably pi, and then it empties the deliveredQueue object.

- **Forward Handling:** Same as delivered Handling, except that the message received is forward instead of delivered, and if the receiving process detects that it is the original sender of the message, then it drops the message instead of forwarding it to the neighboring process.

- **Leave Handling:** When pi+1 receives a leave message:
    1. It downloads the files contained in the message from the sender, if any.
    2. It adds the downloaded files, if any, to the deliveredQueue, and downloadedQueue.
    3. It removes the message sender process from the group by removing its corresponding procParam instance from the groupMapper object. If the leaving process was the left peer process, pi+1 connects to it's a new active left peer derived from the groupMapper object after removing the leaving process from it.
    4. It Keeps the received message's status set to "leave", changes the forwarder id to pi+1, Sends the message to its new left peer and then clears the deliveredQueue.

- **Crash Handling:** When pi+1 receives a crash message:
    1. It downloads the files in contained in the message from the sender, if any.
    2. It adds the downloaded files, if any, to the deliveredQueue, and downloadedQueue.
    3. It removes the crashed process from the group by removing its corresponding entry from the groupMapper object. If the crashed process was the left peer process, pi+1 connects to a new active left peer derived from the groupMapper object after removing the crashed process from it.
    4. It Keeps the received message's status set to "crash", changes the forwarder id to pi+1, Sends the message to its new left peer and then clears the deliveredQueue. However, if pi+1 is the original sender of the message, it does not forward the message further and drops it instantly without any processing.

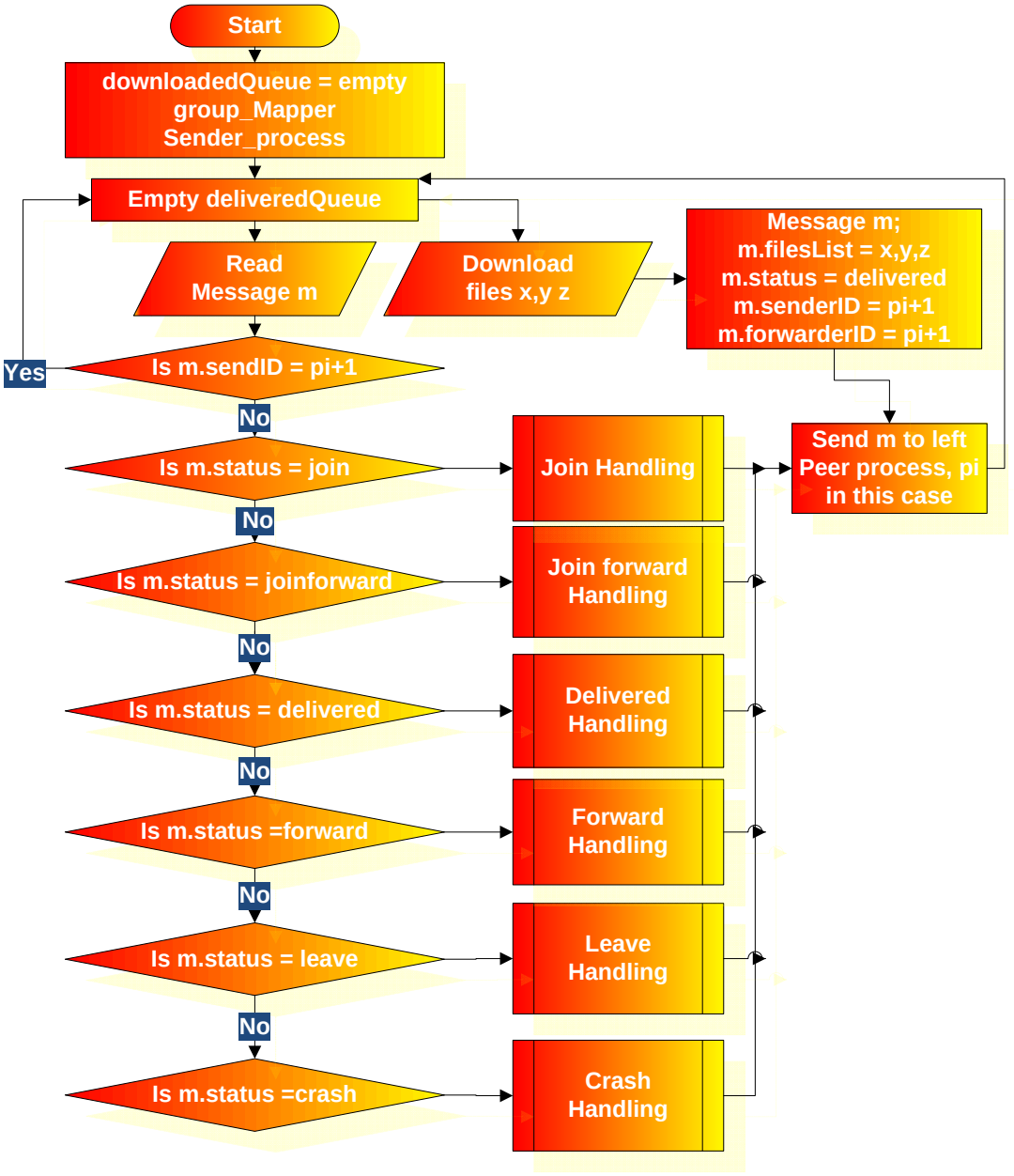

Figure 7.3: Flowchart of the group communication algorithm

7.5 Algorithm and System Implementation

In the distributed group communication file transfer application developed, the algorithm and system model were implemented using java as programming language. To allow communication between processes according to the discussed system model and the algorithm, each process has a client thread, a server threads, and a delivered queue attached to it. The communication is based on a client-server model, where each process uses its client thread to write to the server thread of the neighboring processes in a clockwise fashion. When the server thread of a process

receives data, it places that data in a delivered queue. Then, the client thread of that process reads the data from the delivered queue and forwards it to the next process's server thread in the ring topology in a clockwise manner as shown in figure 7.4.

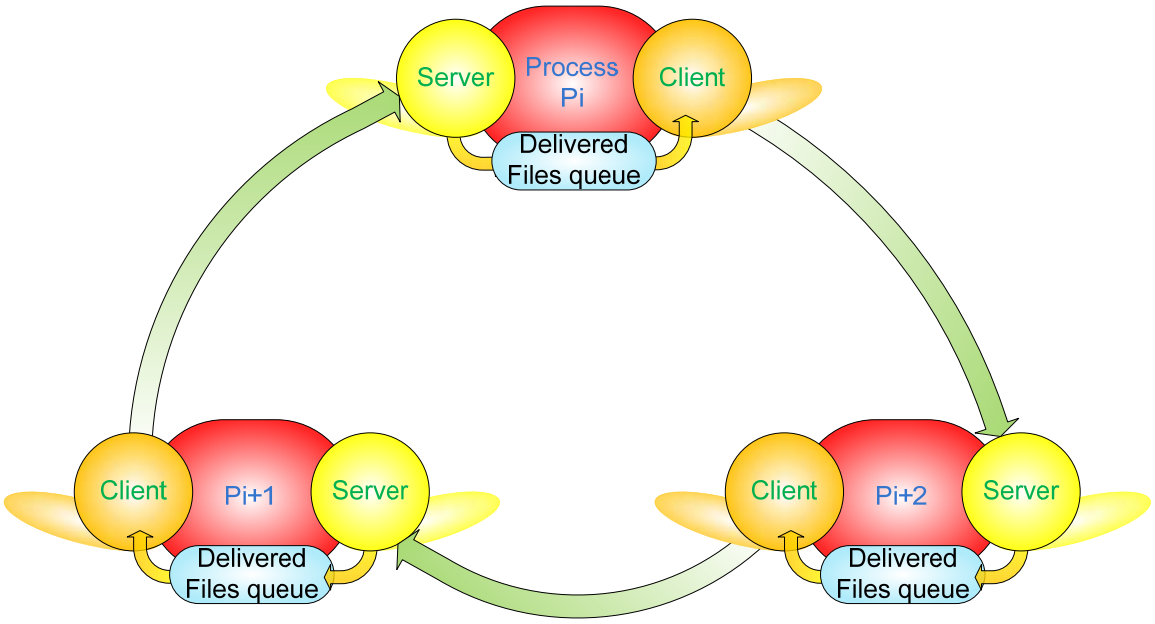

Figure 7.4: Processes in group communication

In the distributed group communication file transfer application, the properties of a reliable multicast (15) are achieved. The properties of a reliable multicast along with a discussion on how they are preserved are as given below:

- **Integrity:** For every message m, every correct process delivers m at most once only if it was multicasted by some process. This property is preserved by the use of TCP as a reliable transport protocol at the communication level. TCP, which stands for transmission control protocol, ensures reliable message delivery and integrity as for one-to-one communication by using checksums, ACK, Sequence numbers, and by rejecting duplicates. Thus, the message received is identical to the one sent, and no messages are delivered twice. Also, when sending a message, it will reach the recipient and it will reach it only once.
- **Validity:** if a correct process multicasts m, it will eventually deliver m. This is ensured in the application code by the fact that a process delivers a message before sending it to next process.

- **Agreement**: if a correct process in group p delivers a message m then all correct processes will eventually deliver m. All or nothing, and atomicity refers to agreement which means that even if the process sending message m crashes while m is in transit, then any correct process that delivers m must make sure that all correct processes in the group receive it. This property is preserved in the algorithm by the fact that each process that delivers a message m forwards it to the next process in the topology in a clockwise fashion. Also, if the sender crashes the algorithm recovers from this crash and a new ring network overlay is established between processes. This property is further described in the subsequent crash handling scenario section.

  The reset of the chapter is devoted to describe the implementation of the algorithm and the DGCFT application by investigating all the possible scenarios for 3 processes pi, pi+1, and pi+2 while executing the algorithm.

7.6 Group Join Scenario

The application allows a process to join an already existing group or to create its own group. In the first case the process or user's application sends an XML request (<groups></groups>) to the XDAS as shown in figure 7.5. In this figure, process pi sends a request to the server asking it about the available groups. When the server receives the request it sends a response that contains the available groups.

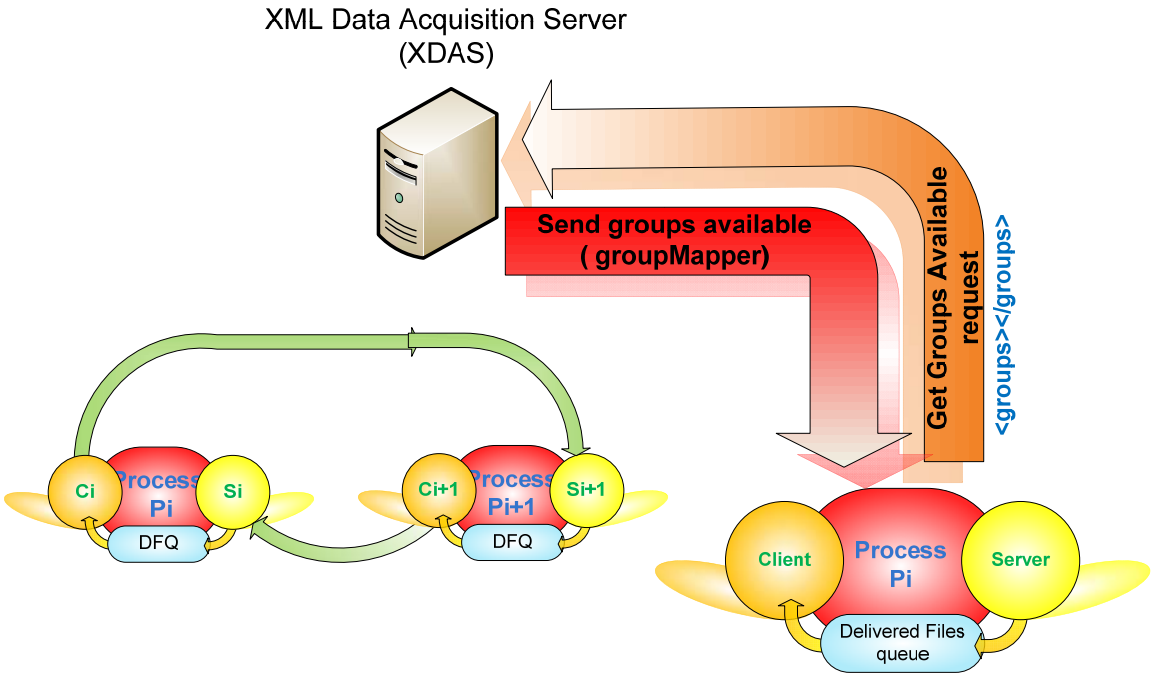

Figure 7.5: Available groups request

Once the use's application receives a response of the available groups, and if either the user chooses to join or create a group, then the application sends and XML join request that contains

a port number and a group id. The port number corresponds to the port of the process's server on which it will listens for connections from group members, and the group id is and identifier of the group to join or create. When the XDAS receives the request it instantiate a processParameter (procParam) object, and initialize it with fields corresponding to the requesting process parameters that are derived from the TCP connection and from the fields provided in the request. For example, the XDAS initializes the processParameter object with IP address and server port number of the requesting process's machine. Then the procParameter instance is mapped to a group in a groupMapper object residing at the XDAS. The groupMapper as well as the procParameter objects are then sent to the requesting process or application as shown in figure 7.6.

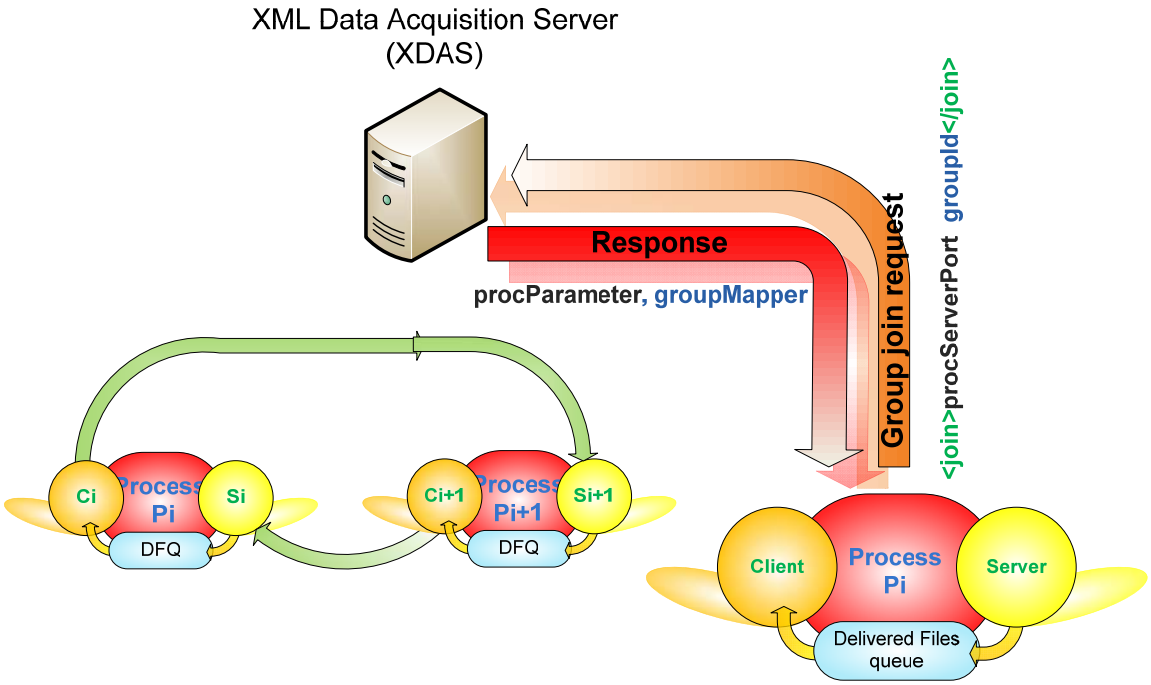

Figure 7.6: Group joing request and response

The above figure shows that process pi gets a processParameter and groupMapper objects from the XDAS, and then changes its id from pi to pi+2 as shown in figure 7.7. The first object obtained contains information related to the process such as process's unique id, which is pi+2 in this case, and process's server port. The second object contains information about the entire group members, such as their unique Ids, server ports, etc. As describe by figures 7.7 and 7.8, the steps involved in a group joining operation for 3 processes are as follows:

1. To join the group pi+2 connects it client thread to its left peer, pi+1, in a clock wise fashion and sends a join message which contains pi+2 processParameter object and might also contain a list of files names and lengths in bytes that correspond to the downloaded files by pi+2 before it joins the group.
2. When pi+1 receives the join message it:

a. Updates its groupMapper object with the new process by adding the processParameter in the message to it.
b. Downloads available files from pi+2, if any.
c. Checks to see if it is the head or first process in the group, which is not, and then set the message status as “joinforward” and the forwarderID to pi+1, and sends it to pi.

3. When pi receives the joinforward message it:
   a. Updates its groupMapper object with the new process by adding the processParameter in the message to it.
   b. Downloads available files from pi+1, if any.
   c. Checks to see if it is the head or first process in the group, which is. Since pi is the head process it has to establish connection with the new peer, pi+2, in a clockwise ring fashion. Consequently process pi terminates its client connection ci to process’s pi+1 server (si+1), and redirects it to connect to pi+2 or the new peer to establish a ring. Pi then sends a new join message to pi+2 that contains the list of files this group has up to this moment, and it forwards the joinforward message received from pi+1 to pi+2 by setting the message status as “joinforward” and the forwarderID to pi.

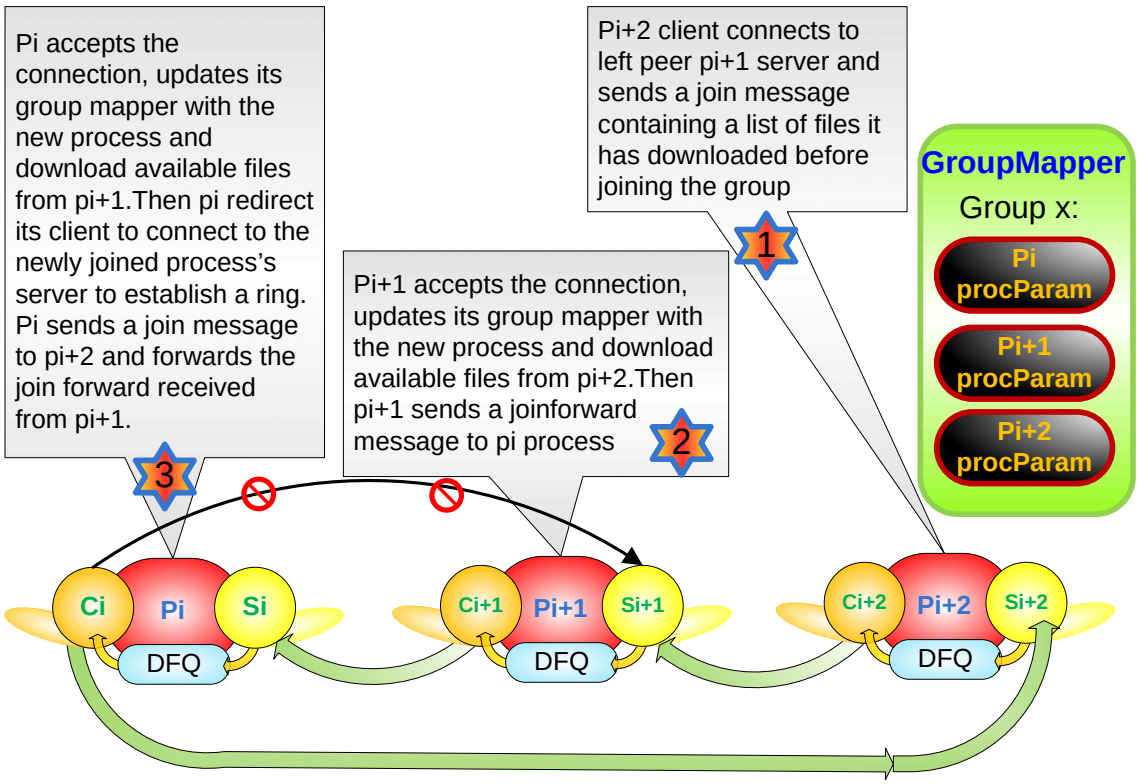

Figure 7.7: Group joining scenario

4. When pi+2 receives the join message from pi it:

a. Tries to updates its groupMapper object with pi, but it discovers that no update is needed since pi already exist in the groupMapper object obtained from the XDAS.
b. Downloads all group available file from pi.
c. Checks to see if it is the head or first process in the group, which is not. Then it sets the message status as “joinforward” and the forwarderID to pi+2, and sends it to pi+1. At this point pi+2 also receives a “joinforward” message from pi which correspond to the join message that pi+2 has previously sent. Pi+2 checks the senderID field of the received message and determines that it is the original sender of the message. Consequently, pi+2 drops the second message and prevents it from circulating in the network overlay. By receiving its own sent message back pi+2 can be sure that all other processes have received its join message.

5. When pi+1 receives the “joinforward” message from pi+2 sent by pi it:
   a. Tries to updates its groupMapper object with pi, but it discovers that no update is needed since pi already exist in the groupMapper.
   b. Downloads available files from pi+2, if any.
   c. Checks to see if it is the head or first process in the group, which is not. Then it sets the message status as “joinforward” and the forwarderID to pi+1, and sends the message to pi.
6. When pi receives the “joinforward” message from pi+1 sent by pi:
   a. Checks the senderID of the message and since it matches its ID it drops the message because it is the original.

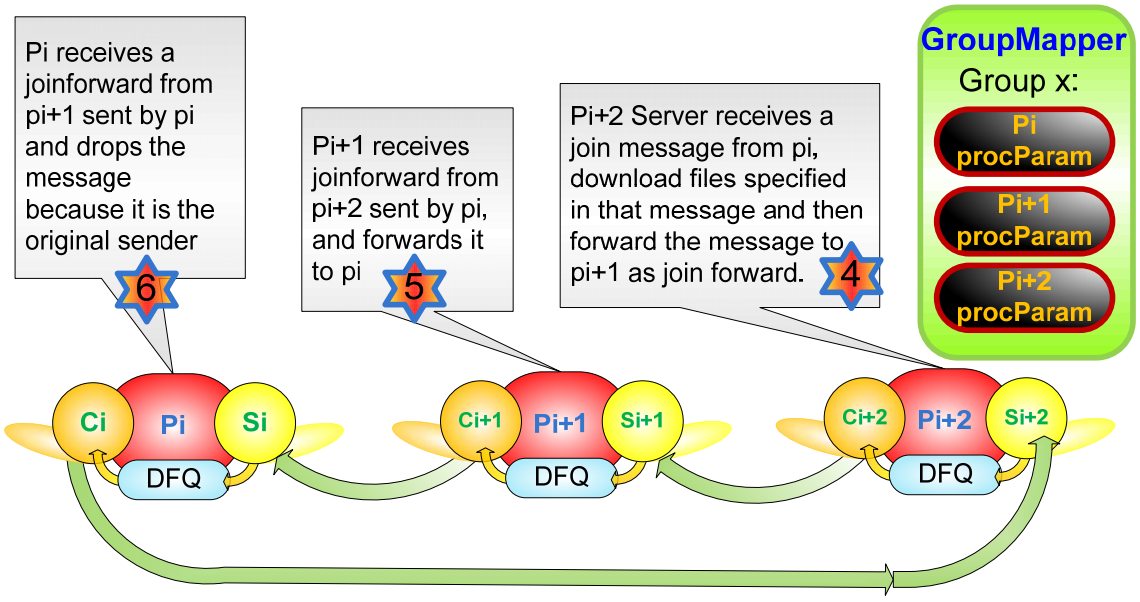

Figure 7.8: Group joining scenario

7.7 Group Leave Scenario

Figure 7.9 shows how a process leaves a group. In this figure, the assumption is that pi, pi+1, and pi+2 are already belonging to the same group and communication. If pi decides to leave the group the following events occur:

1. Pi sends a leave request to the XDAS and a massage with status “leave” to pi+2. When the XDAS receives the request it updates its copy of the groupMapper object by removing Pi from it and sending a confirmation message to pi.
2. When Pi+2 receive a message with status “leave” from pi. It first removes the sender from the groupMapper object. Then, it checks to see if the sender is its left peer or corresponding server to which this process’s client connects to, which is not. Finally it forwards the message to pi+1 by setting the forwarderID to pi+2 while leaving the status of the message set to “leave”.
3. When pi+1 receives the “leave” message it:
    a. Updates its groupMapper object by removing the sender (pi) from the group.
    b. It Checks to see if the sender is its left peer or corresponding server to which it’s connects. Since the sender is the corresponding server or the left peer process to which pi+1 client thread connects, then pi+1 terminates its client connection (Ci+1) to process’s pi server (Ci), and redirects it to connect to pi+2 server (Si+2) by sending a join request. (I must nod send the files again if there is a leave)

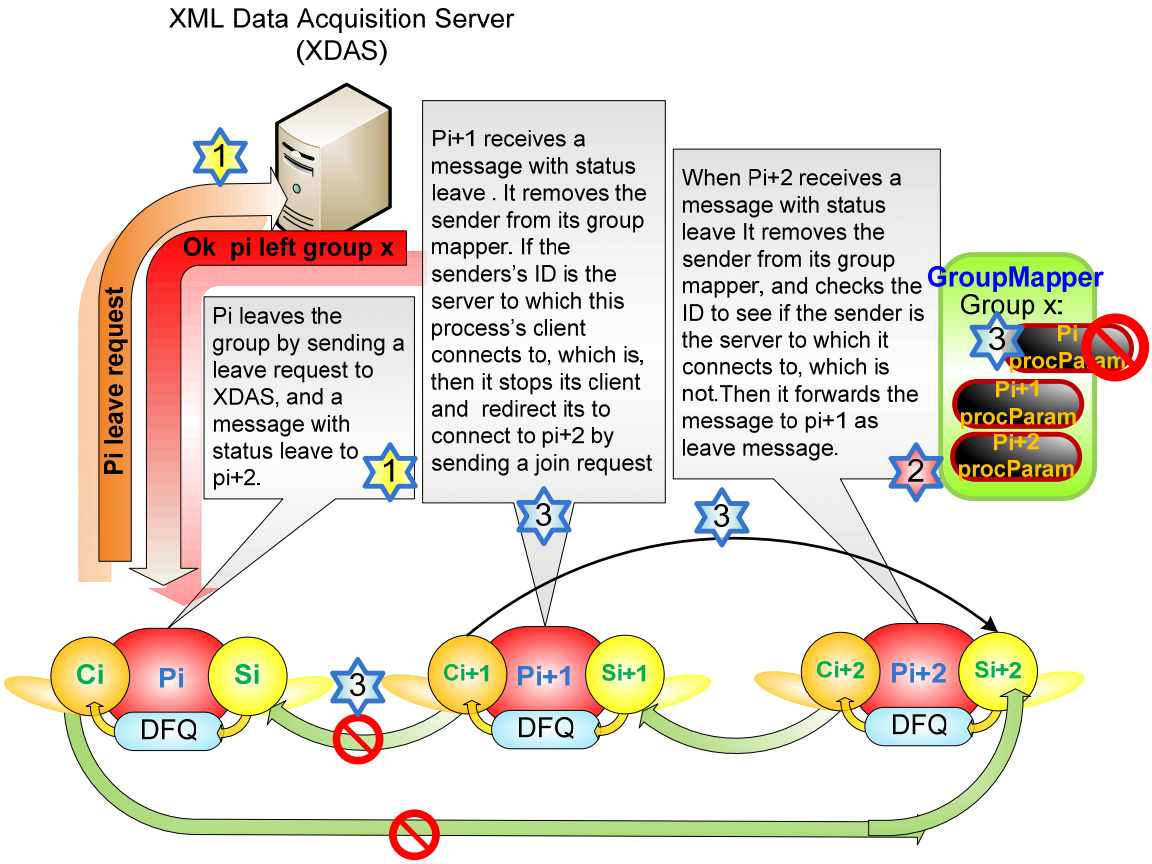

Figure 7.9: Group leaving scenario

7.8 Crash Handling Scenario

One of the properties of a reliable multicast, is agreement as mentioned previously. In figure 7.10, agreement means that if process pi+1 fail by crashing while sending or receiving a set of files, then any correct process that sent or received files from pi+1 should make them should make sure that all correct processes in the group receive a copy of these files.

If a process fails by crashing, the algorithm recovers nicely, and agreement between all processes is preserved. To demonstrate, figure 7.10 illustrates the following crash scenario:

1. pi+1 fails by crashing
2. pi detects the crash of this process and it:
    a. removes pi+1 from the group by removing the corresponding entry of this process from the groupMapper object.

b. sends a notification about the crashed process to the XDAS, which removes that process from the groupMapper instance that it maintains.
c. sends a crash message that contains the crashed process and delivered files to its left peer (pi+2).

3. When pi+2 receives a message with status crash it:
   a. removes the crashed process from it groupMapper object and thus from the group.
   b. Checks to see if the crashed process id corresponds to its left peer process or the corresponding server(pi+1) to which this process's(pi+2) client connects to. Since the id is for the left peer process, pi+2 terminates its client connection (Ci+2) to crashed process's server (Si+1), and redirects it to connect to the server of pi (Si) by sending a join request. (I must send the files again if there is a crash if pi+1 crashed while downloading messages from pi+2)

      In the figure below, agreement is achieved when the algorithm tries to recover from the crash because pi and pi+2 send the files they have delivered to each other after pi+1 crashes.

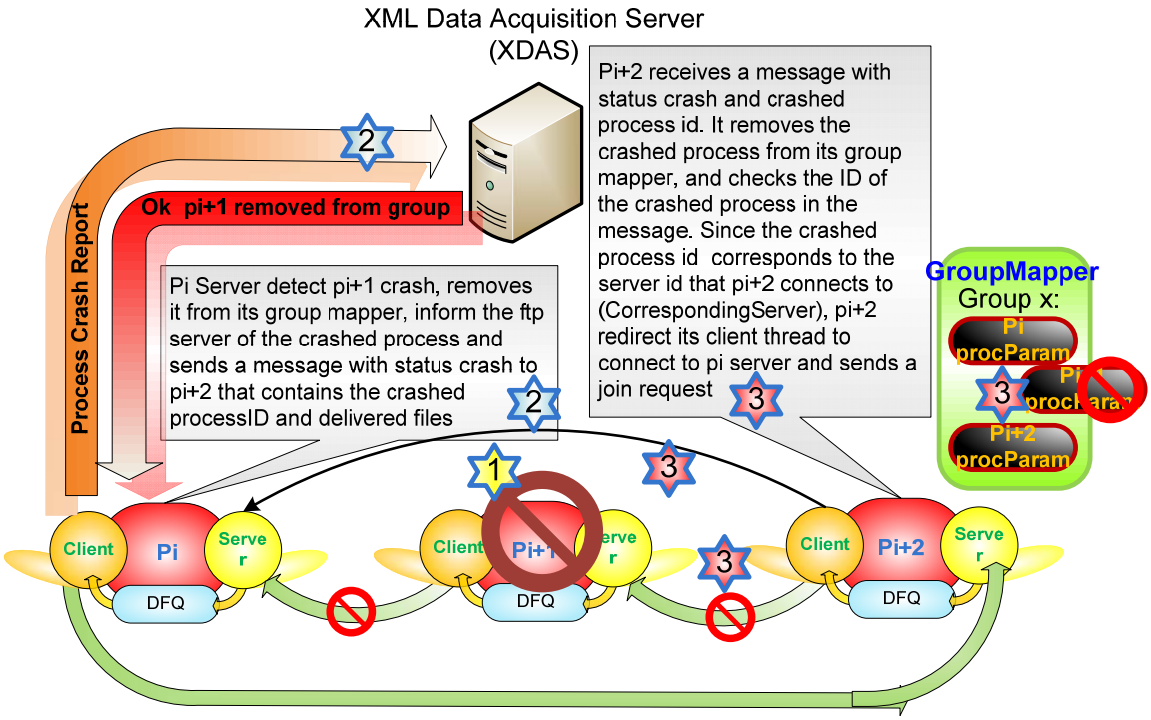

Figure 7.10: Crash handling

7.9 File Transfer

The DGCFT application allows the user to download a set of files from the XDAS through a graphical user interface as will be described in the user manual section. After the user selects the sets of file to download, the application process sends an XML download request to the XDAS as shown in figure 7.11, where process pi+2 sends "<file> file name </file>" request to the server. According to the figure when the server receives the request it responds back with the requested files and the following happens:

1. Process (Pi+2) downloads the files from the server, delivers them to the user's graphical interface, and sends a message m to process pi+1. This message has its status set to delivered, sender id and forwarder id set to pi+2, and its filesMap field set to contains a list of the recently downloaded files names along with their corresponding lengths.
2. When pi+1 receives m, it downloads the files specified in message from pi+2 and then set the message status to forward, the forwarder id to pi+1, and sends m to pi.
3. When pi receives m, it downloads the files specified in message from pi+1. Then it sets the message's forwarder id to pi, and sends it to pi+2.
4. When pi+2 receives the message again it drops it because the sender id of message is the same as pi+2, which means that it is the original sender, and the message has reached everybody.

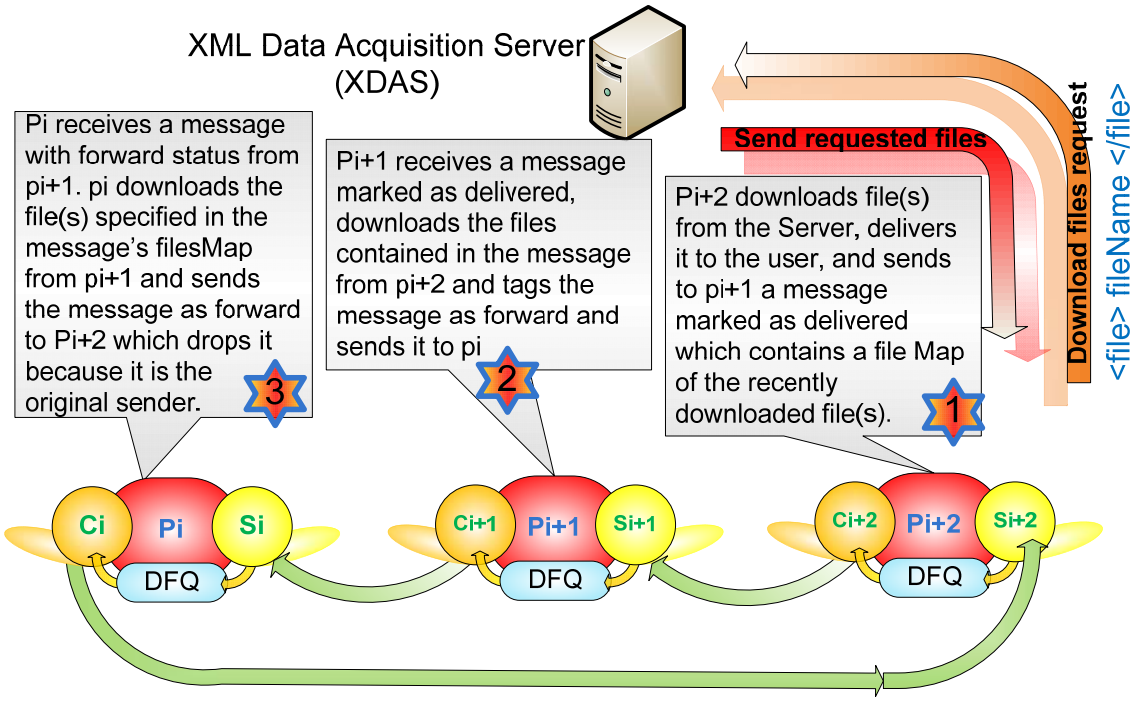

Figure 7.11: Group file download scenario

Chapter 8 Conclusion and Further Work

8.1 Discussion

The functionality of the system was tested on a single machine using a developed simulator to simulate 100 sensor node that connect to the server every 100ms respectively and the 3D web application made these nodes available to the user in less than 6 sec from the time they connected to the server. Also, the system scaled very well even after simulating 1000 node, knowing that in general the number of sensor nodes that will be deployed is expected to be less than that number. Nevertheless, the system will still adapt to a larger number of nodes, and will be only limited by the amount of disk storage and processing power available at the server side, as well as the capacity of Google Earth API.

On the other hand the distributed group communication file transfer application was tested by running six instances of the program on the same machine and establish one group that contain six member. Any group member that downloads a file from a sensor node makes it available to other group members using the previously described ring algorithm. The latency of sharing files between group members depends on the number of member in the group, and the physical location of group member. The more member in the same group, the bigger the ring network topology and the higher the expected delay. Also, the farthest the group member and smaller the bandwidth available the higher the file transfer latency. Furthermore, this application expected to decrease work load on the XDAS machine because all member of the same group have the same files downloaded by other members in that group and there is no need to burden the XDAS by download requests.

8.2 Conclusion

Monitoring a large number of sensor nodes can be both difficult and pain stacking process especially if these nodes are scattered over a large geographical area. Furthermore, it might be that the person that has deployed these nodes is not the same person observing their operation. Consequently, there should be a correlation between the location of these nodes and a geographical map which will give a better view that specifies where these nodes are exactly present over. This will allow the user to react accordingly when he/she obtains some critical sensing measurements from some nodes that require physical presence at these nodes. Moreover, there is always a need by operators, especially in these days, to monitor sensor nodes remotely from any place in the world using the comfort of their web browsers.

In this project a set of applications were developed to allow the visualization monitoring, and download of data files corresponding to sensor nodes from a 3D map using Google Earth API as

well as from a distributed group communication file transfer application, which allows groups creation and file sharing among group members.

The first application that was developed is a server that listens continuously for connections from sensor nodes, and gathers information related to their geographical location and measurement sensed from the surrounding environment. This server, also, listens for connections from replica clients and provides them with the necessary data related to their operation.

The second application that was developed allows the user to instantiate a set of replica clients, which connect periodically to the server discussed above and replicate the database content of that server on a different machine across the web. This approach increases the availability of the system by allowing the web clients to access data on the replica clients' machines even if the server's machine has failed.

The third application that was developed allows the user to visualize the sensor nodes as a place marks in a 3D Google earth map, and to download data related to these nodes from the map interface. This application is web base, which allows remote access to the user from any place in the world using their web browsers. Moreover, the installation requirements of this application on the client side are very minimal and very often the application will run by only accessing the home page of this project.

The fourth application that was developed is a distributed group communication file transfer application. This java application is intended to be used as an ftp client to download files as well as a group communication file transfer client. In ftp mode, the application allows its user to download a set of data files corresponding to each sensor nodes from the first application. In the group communication mode, the application allows a user to join a group and share date files corresponding to sensor nodes with other group members. This application is more suitable to be used on LAN by a set of users that are monitoring the same set of sensor nodes. So, if one user downloads data files corresponding to one node, all other users have these files available and they do not need to download it.

The final application that was developed is a simulator that allows the simulation of set of sensor nodes connecting to the server. These nodes are either simulated over a rectangular

geographical area in multiple node simulation modes, or only one sensor node is simulated over a specific area in a single node simulation mode.

8.3 Future work

While developing this project, there were a lot of nice ideas that could have been included, but were not due to time constraints.

- Embedding video streams acquired from sensor nodes into the 3D web application, which allows the user to play and watch video from a monitoring sensor node, for example. This embedded video stream could simply be a video file stored at the XDAS server, or it could even be acquired in almost real-time, where the server to which nodes sinks data acts as a relay agent between the web user and sensor nodes. This idea is, of course, requires deep knowledge in the underlying communication protocols, especially if real-time streaming is to be considered.
- Develop a 2D map application that has less bandwidth requirements than a 3D map, and thus much more suitable for memory limited devices such as mobile phones.
- Enable the user to control the operation of a sensor node from his/ her web browser by sending commands to desired sensor nodes. This functionality is expected to be application specific, unless some sort of standard is developed.
- Adding some functionality that tells whether a specific node is still alive and sensing data or it has seized operating. If a node is alive it might be given a certain color, and if it is dead it might be given a different color. This will visually signal to the user in order take some action to repair dead nodes, if any.
- Developing a true distributed replica managers system that allows sensor nodes to sink data to a front-end which then submit the data to the replica manager(s) in a predetermined fashion. The distributed group communication file transfer algorithm developed, could be used in implementing such system
- Enhance the proposed distributed group communication file transfer algorithm to allow users to join multiple groups at the same time instead of one group limit.

Bibliography

1. **P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler.** TinyOS: An Operating System. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

2. Wireless sensor network. *wikipedia.* [Online] Wikimedia Foundation, Inc. [Cited: 4 18, 2010.] http://en.wikipedia.org/wiki/Wireless_sensor_network.

3. **Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, John Anderson.** *Wireless Sensor Networks for Habitat Monitoring.* s.l. : (Intel Research, IRB-TR-02-006, June 10, 2002.) 2002 ACM International Workshop on Wireless Sensor Networks and Applications, Sept.2002.

4. **A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao.** *Habitat monitoring: Application driver for wireless communications technology.* In Proceedings of the Workshop on Data Communications in Latin America and the Caribbean : s.n., Apr. 2001.

5. **Denise Dudek, Christian Haas, Andreas Kuntz, Martina Zitterbart, Daniela Kr¨uger, Peter Rothenpieler, Dennis Pfisterer, Stefan Fischer.** *A Wireless Sensor Network For Border Surveillance.*

6. **E. Arens, C.C. Federspiel, D. Wang, and C. Huizenga.** How Ambient Intelligence will Improve Habitability and Energy Efficiency in Buildings. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

7. **G. Stromberg, T.F. Sturm, Y. Gsottberger, and X. Shi.** Low-Cost Wireless Control-Networks in Smart Environments. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

8. **W.J. Kaiser, G.J. Pottie, M. Srivastava, G.S. Sukhatme, J. Villasenor, and D. Estrin.** Networked Infomechanical Systems (NIMS) for Ambient Intelligence. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

9. **Haksoo Choi, Sukwon Choi, Hojung Cha.** *Structural health monitoring system based on strain gauge enabled wireless sensor nodes.* s.l. : This paper appears in: Networked Sensing Systems, 2008. INSS 2008. 5th International Conference on , 2008.

10. **T. Hui Teo, Gin Kooi Lim, Darwin Sutomo David, Kuo Hwi Tan, Pradeep Kumar Gopalakrishnan, Rajnder Singh.** *Ultra Low-Power Sensor Node for Wireless Health Monitoring System.* Institute of Microelectronics, Singapore.

11. **Jung, C. Lauterbach and S.** Integrated Microelectronics for Smart Textiles. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

12. **Rabaey, J. van Greunen and J.** Locationing and Timing Synchronization Services in Ambient Intelligence Networks. [book auth.] W.Weber J.M. Rabaey E. Aarts. *Ambient Intelligence.* Verlag Berlin Heidelberg : Springer, 2005.

13. XML ESSENTIALS. 2010. [Cited: 4 10, 2010.] http://www.w3.org/standards/xml/core.

14. XML Tutorial. 2010. [Cited: 4 10, 2010.] http://www.w3schools.com/xml/default.asp.

15. **George Coulouris, Jean Dollimore, Tim Kindberg.** *Distributed Systems: Concepts and Design (4th edition).* s.l. : Pearson Education Limited, 2005.

16. Google Earth. 2010. http://earth.google.com/.

17. Google Earth API. 2010. http://code.google.com/apis/earth/.

18. Google Earth API Developer's Guide. 2010 http://code.google.com/apis/earth/documentation/index.html.

19. Google Maps API. 2010. http://code.google.com/apis/maps/.

Appendix A User Manual

A.1 Introduction

This section is intended to get you started using the programs implemented for this master thesis. The programs suite is intended to visualize, monitor and download data from sensor nodes. The functionality of these programs can by divided into two logical section. The first section is the server side application that is necessary to interact and sink data from sensor nodes. The second section is the client side applications that are necessary to replicate, share, and present data to the users. In general, the following applications will be discussed:

- DataAcquistionServer
- SensorNodeSimulator
- ReplicaClient
- DGCFTA
- 3D Web Application

Where DataAcquisitionServer corresponds to the first logical section, and the rest belong to the second logical section.

A.2 Requirements

A.2.1 General Requirements

In order to run the software you need to have java runtime environment installed on your machine (jre). The software was developed using java development kit (jdk) version 6.1 (java 6 update 1). Thus, to simply run the applications provided you need to install jre6 or above. You can also install jdk6.1 or above, which includes jre, if you are interested in developing further java applications. If you have older versions of the jre the software might still run if the java virtual machine supports the current version of your operating system, however, I have only tested the software using jre6. At the time of writing, you can download the latest version from http://java.sun.com/javase/downloads/index.jsp

A.2.2 Server Side Requirements

In addition to the general requirements mentioned above, and if wed access is to be allowed to remote users using web browsers, then you need to have a web server running on the machine that runs either the DataAcquistionServer.jar or the ReplicaClient.jar applications. On the other hand, if access is only to be granted through DGCFTP program then there is no need for a web server, but the software was mainly designed to allow users to visualize nodes geographical location using Google earth API, so web access is highly recommended.

A.2.3 Client Side Requirements

There are two methods for users to monitor and acquire data from sensor nodes through DataAcqusition.jar application and web Servers. They can either use their web browsers, or the DGCFTA.jar application to gain access. In the former case, the only requirement is to have JavaScript enabled in the web browsers, and in the latter case users only need to have jre installed.

A.3 Data acquisition Server Application

A.3.1 How to Run

After downloading the necessary jre or jdk mentioned in the previous section, you can use the jar file (DataAcquisitionServer.jar) to run the server.

A.3.2 Functionality

This program, DataAcquisitionServer.jar, is the server that receives connections from sensor nodes, replica Clients, and DGCFTP clients. When the server receives data from sensor nodes it stores that data in a directory called “WebDirectory” that is used to serve replica Mangers, and DGCFTP clients. Also this directory is used as a web directory to provide sensor nodes’ data to web client via http requests. Thus, there should be a web (www) server running on the same machine that runs the DataAcquistionServer.jar application. This web server can be any commercial or free server available such as IIS, Apache,etc...

A.3.3 The Graphic user Interface (GUI)

The GUI of the DatabasAcquistionServer.jar application with the description of each section is depicted in the figure below.

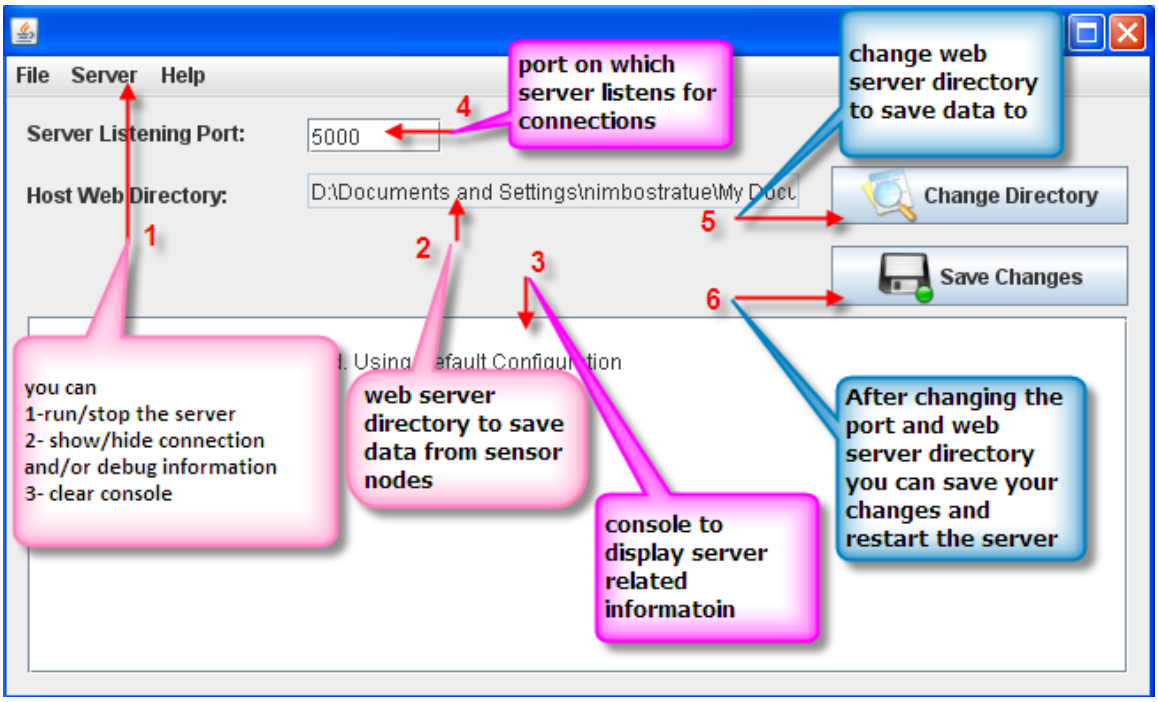

Figure A. 1: GUI of the DatabasAcquistionServer.jar application

Initially the server is not running, so to run it you can go to server>run as specified by (1) in the GUI. Once the server is running you can stop it by going to server>stop. Moreover, you can change the port (4) on which this server listen for connections from sensor nodes, replica managers and DGCFTP clients. Also you change the web server directory where the application stores sensor nodes received data. Once you make your choice and save your changes, you need to restart the server again for the changes to take effect.

A.4 Sensor Node Simulator Application

A.4.1 How to Run

After downloading the necessary jre or jdk mentioned in the requirements section, you can use the SensorNodeSimulator.jar jar file to run the program. Before running the simulator, make sure that the DataAcqusitionServer is running and listening on the default port 5000.

A.4.2 Functionality

As its name implies, the SensorNodeSimulator.jar program is used to simulate sensor nodes connecting to the DataAcquisitionServer.jar application. It can run in 2 modes:

1. Multiple Node Simulation: In this mode, multiple nodes connecting to the DataAcuqistionServer.jar application with specified delay can be simulated, respectively. The user can choose a geographical region over which the simulation should take place,

and the program uses this region to randomly simulate sensor nodes at random geographical locations within the region. Moreover, to facilitate entering the required bounds for the geographical region, the user can use the web application or home page of the project to find these bounds.

2. Single Node Simulation: In this mode, only a single sensor node is simulated. The user gives the exact location of this sensor node [latitude, longitude] and the program simulates connection from this node to the server. This mode is useful if you need to place a node at a specific location rather than random location. To find the exact location of a node, the user can refer to the 3D web application and double click anywhere in the map to generate the coordinates at the bottom of the home page.

A.4.3 The Graphic user Interface (GUI)

The GUI with description for the SensorNodeSimulator.jar is given in the figure below.

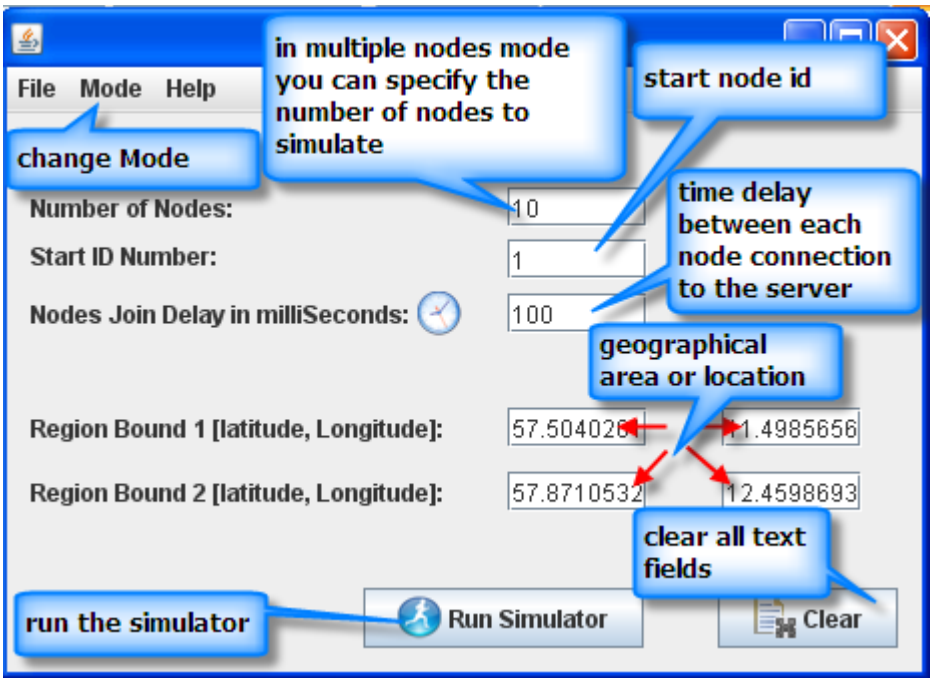

Figure A. 2: The GUI of SensorNodeSimulator.jar application

You can change the operation mode of the simulator by going to mode menu where you can choose between multiple or single node simulation modes. Respectively, you can use the short cut keys m and s to change toggle between the two modes.

In multiple nodes simulation mode you can find the bounds for the region that you want to simulate sensor nodes over by using the 3D web application provided with this project. Moreover, in single node simulation mode only two fields will be available for latitude and longitude and you can enter the values directly or you can again use the 3D web application to generate the values.

A.5 Replica Client Application

A.5.1 How to Run

After downloading the necessary jre or jdk mentioned in the requirements section, you can use the jar file, ReplicaClient.jar, to run the program. For successful replication, the data acquisition server should be running, otherwise, the program will keep trying to connect to the server and no data will be replication unless the server becomes alive.

A.5.2 Functionality

The ReplicaClient.jar program connects to the data acquisition server (DataAcquisitionServer.jar) periodically and replicates in a folder called "WebDirectoryReplicated" sensor node related database hierarchy available in the web directory on the server's machine. Consequently the machine running this replica client has all the data necessary to serve web clients in case the web server of the machine running DataAcquisitionServer.jar application fails. Of course, a web (www) server should be running on the machine that serves web client from the replication folder via http requests, i.e. the www server should expose the necessary replication directory data to public clients.

A.5.3 The Graphic user Interface (GUI)

The GUI of the program is displayed in the figure below.

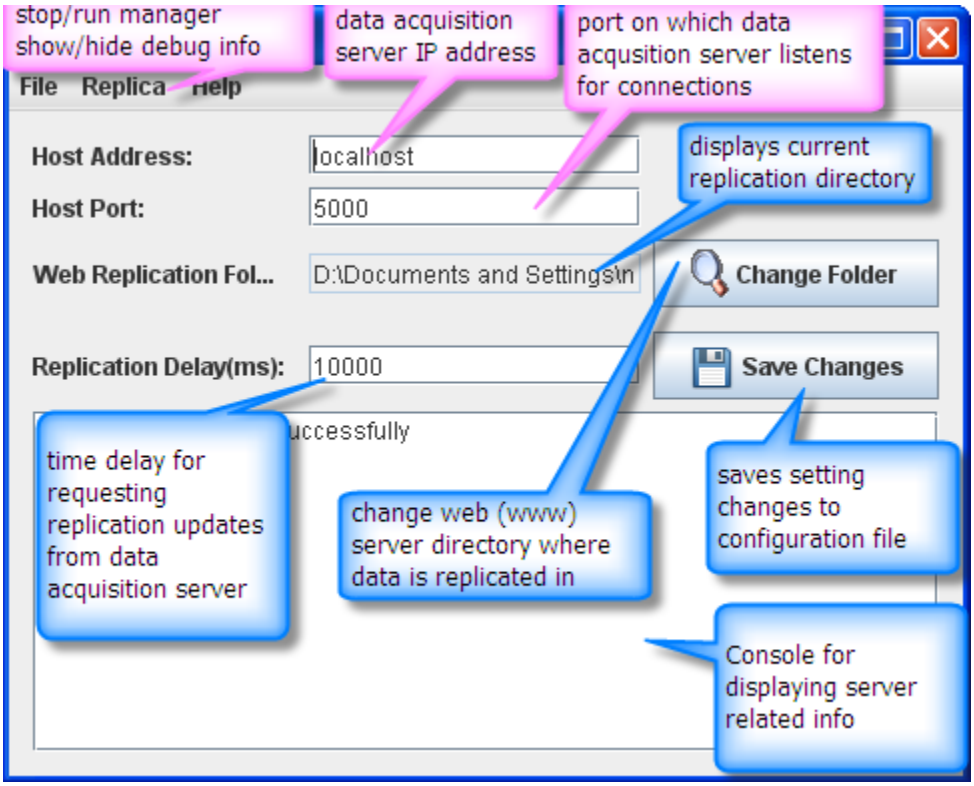

Figure A. 3: The GUI of ReplicaClient.jar application

To run the replica manager go to Replica>Run, and to stop it go to Replica>Stop. You can also change the IP address and port of the server (DataAcquisitionServer .jar) to which this manager connects, the web replication folder, and the replication delay time that specifies how often should this client application connects to the server for replication updates. Once you change the setting you can save them to disk and the new settings will be applied once the manager tries to connect to the server again or when a new instance of the replica manager program is created.

A.6 Distributed Group Communication File Transfer Application (DGCFTA)

A.6.1 How to Run

After downloading the necessary jre or jdk mentioned in the requirements section, you can use the jar file, DGCFTA.jar application, to run instances of the program.

A.6.2 Functionality

The program represents a process that encapsulates a distributed file transfer protocol mechanism. In its simplest form of functionality, the program process downloads available sensor nodes data files from the FTP server (DataAcquisitionServer.jar) and displays them to the user. Moreover, the program process can join a communication group and share downloaded files with other group member through a distributed group communication file transfer protocol. Whenever any process within the group downloads a file from the server, all other process will have copy of these download files, which means that all processes within a group will have the same set of files downloaded.

A.6.3 The Graphic user Interface (GUI)

The GUI of the main program is shown in the figure below, where the functionality of each GUI component is displayed in a corresponding descriptive callout.

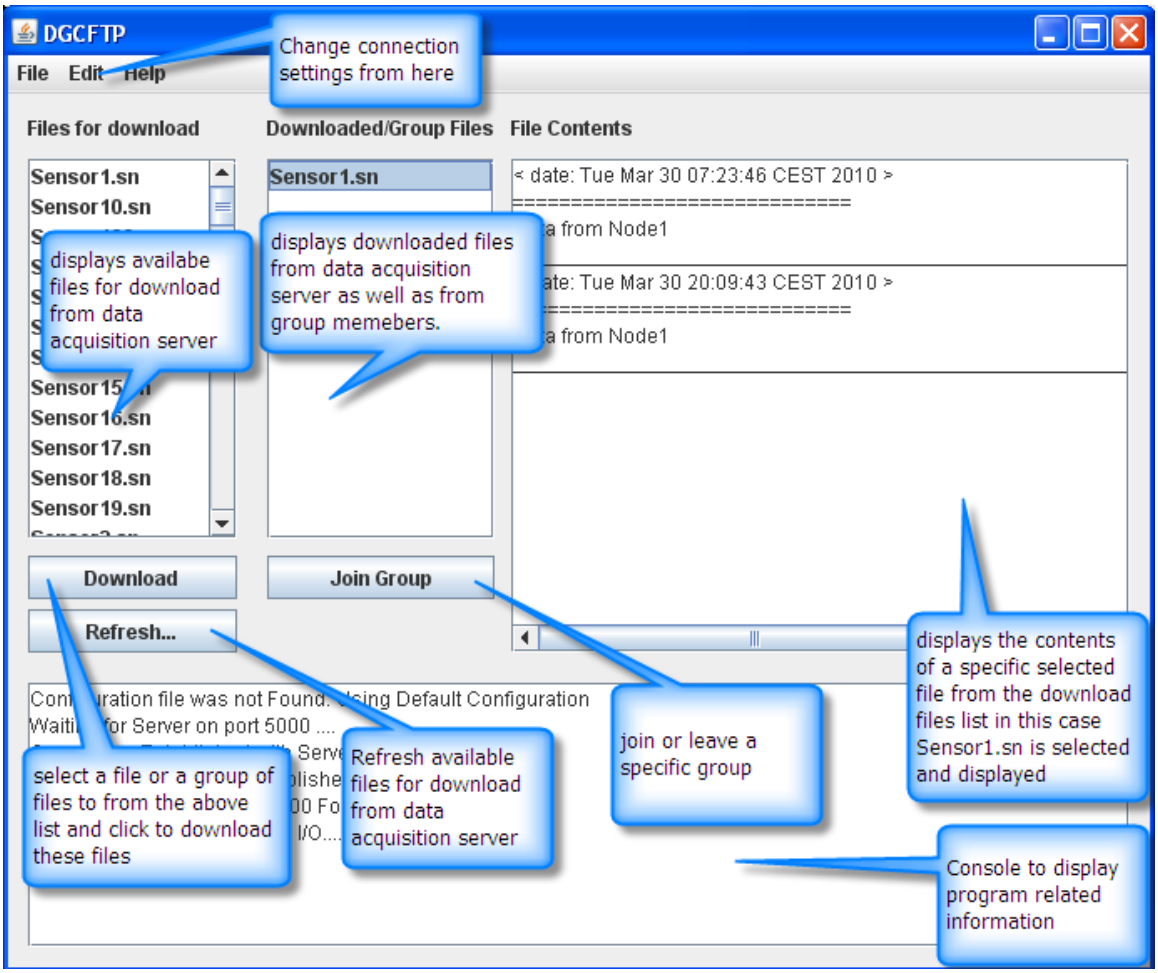

Figure A. 4: The GUI of DGCFTA.jar application

To change the settings of the program one can invoke the settings panel from Edit>settings. The functionality and description of this panel is given below.

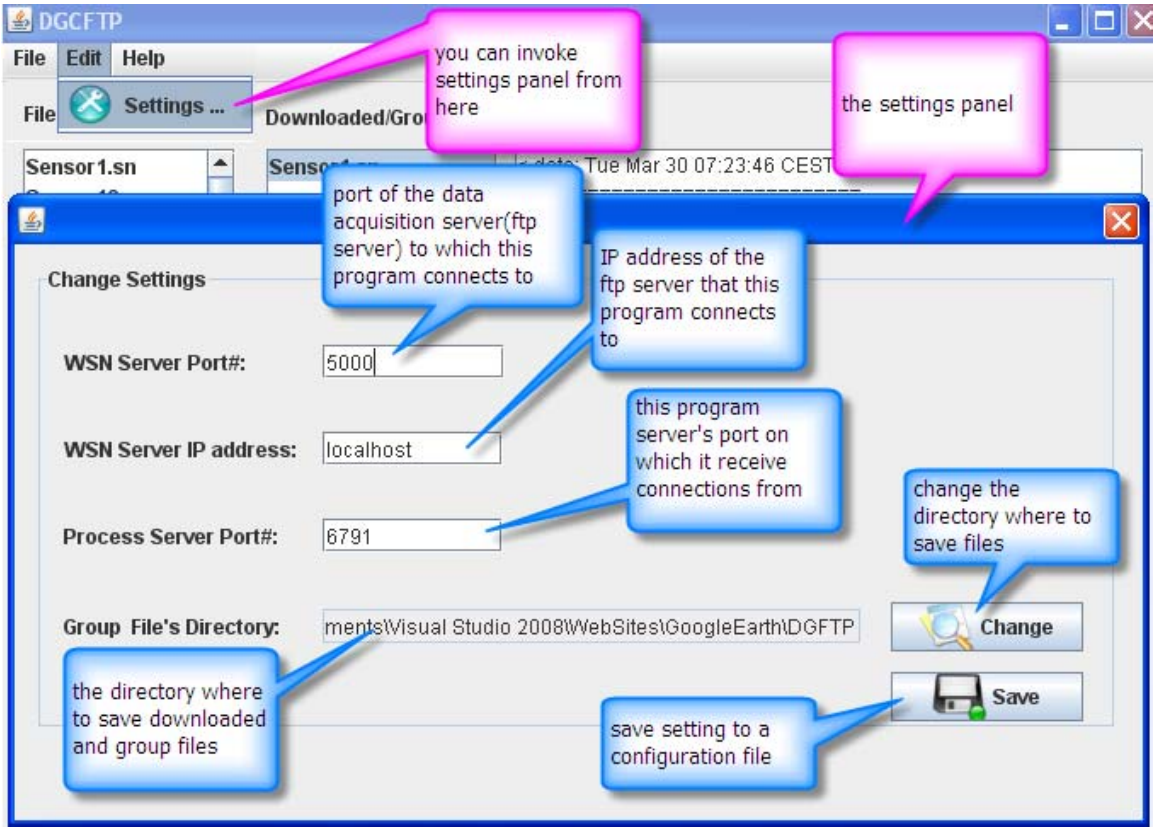

Figure A. 5: The GUI of the DGCFTA's settings panel

Also, to invoke the group join panel you can click on the join group button. The functionality and description of the panel is presented in the figure below

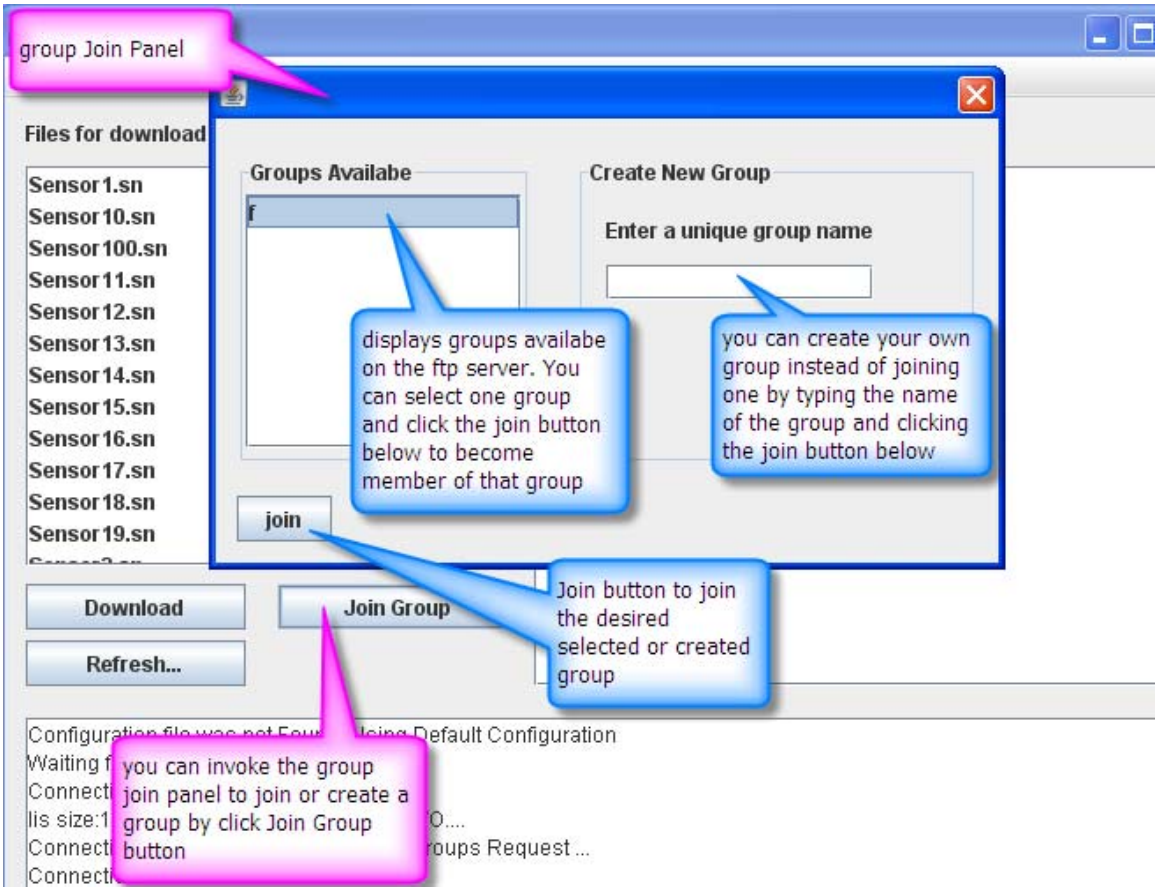

Figure A. 6: The GUI of the group join panel

A.7 3D Web Application

A.7.1 How to Run

A.7.1.1 Client Side Settings and Installation

This application uses Google Earth and Maps APIs, and requires Google Earth Plugin in to be available in the browser. According to (18) and at the time of writing of this document the Google Earth Plugin is currently available for the following platforms:

Microsoft Windows (2000, XP, and Vista)

- Google Chrome 1.0+
- Internet Explorer 6.0+
- Firefox 2.0+
- Flock 1.0+

Apple Mac OS X 10.4 and higher (Intel and PowerPC)

- Safari 3.1+
- Firefox 3.0+

When you run the application and if the Google Earth Plugin does not load then you need to install it by browsing to http://code.google.com/apis/earth . Once the plugin is installed, you may need to refresh the page before the plugin displays correctly. For more detailed information see requirements at (18) (19).

As soon as the plugin in is available you can simply browse to the project's home pages and start using the application.

A.7.1.2 Server Side Settings and installation

The machine that is hosting this project's home page should have a web server installed, and the output from DataAcquisitionServer.jar and ReplicClient.jar applications should be directed to point to the folder containing this projects home page. This means that the database.xml, nodes.xml, and Sensor#.sn files generated by these applications should reside in the same directory containing the web application of this project. By default these files are placed in the WebDirectory folder in case you are running the DataAcquistionServer.jar application and in WebDirectoryReplicated folder in case you are running ReplicaClient.jar application. Furthermore, there are two files included with this project that can be used as a web application. The first file is ThesisASP.aspx, which was written using ASP.Net and requires Microsoft IIs web server to be running. The second file, ThesisHTML.html, was designed in pure html, and can be run by any available web server. Both files refer to javaScript file called myscript.js that should be placed in the same directory where the files reside. The easiest option to get started is to is to deploy the second web application (ThesisHTML.html) first because it does not require the installation of Microsoft's IIS or any other server that can interpret ASP.NET, but one and still deploy both files as the project's main page. To get started quickly for testing purposes, put the ThesisHTML.html and myscript.js files in the WebDirectory generated by DataAcquisitionServer.jar, and then double click on the web page to open in the default browser.

A.7.2 Functionality

The 3D web application provided in this project allows the user to visualize place mark in Google Earth place marks with descriptions that represent the geographical locations of sensor nodes, and to download data files corresponding to each sensor node. Typically, these files contain information sensed by each node from its surround environment and are stored at the server. In addition to allowing the user to navigate to a specific location on earth and to add additional layers to the 3D earth window, the application provides the user with parameters that are used as an input for SensorNodeSimulator.jar application for simulating purpose.

A.7.3 The Graphic user Interface (GUI)

The GUI of the 3D Web application is shown in the figure below, where the functionality of each GUI component is displayed in a corresponding descriptive callout.

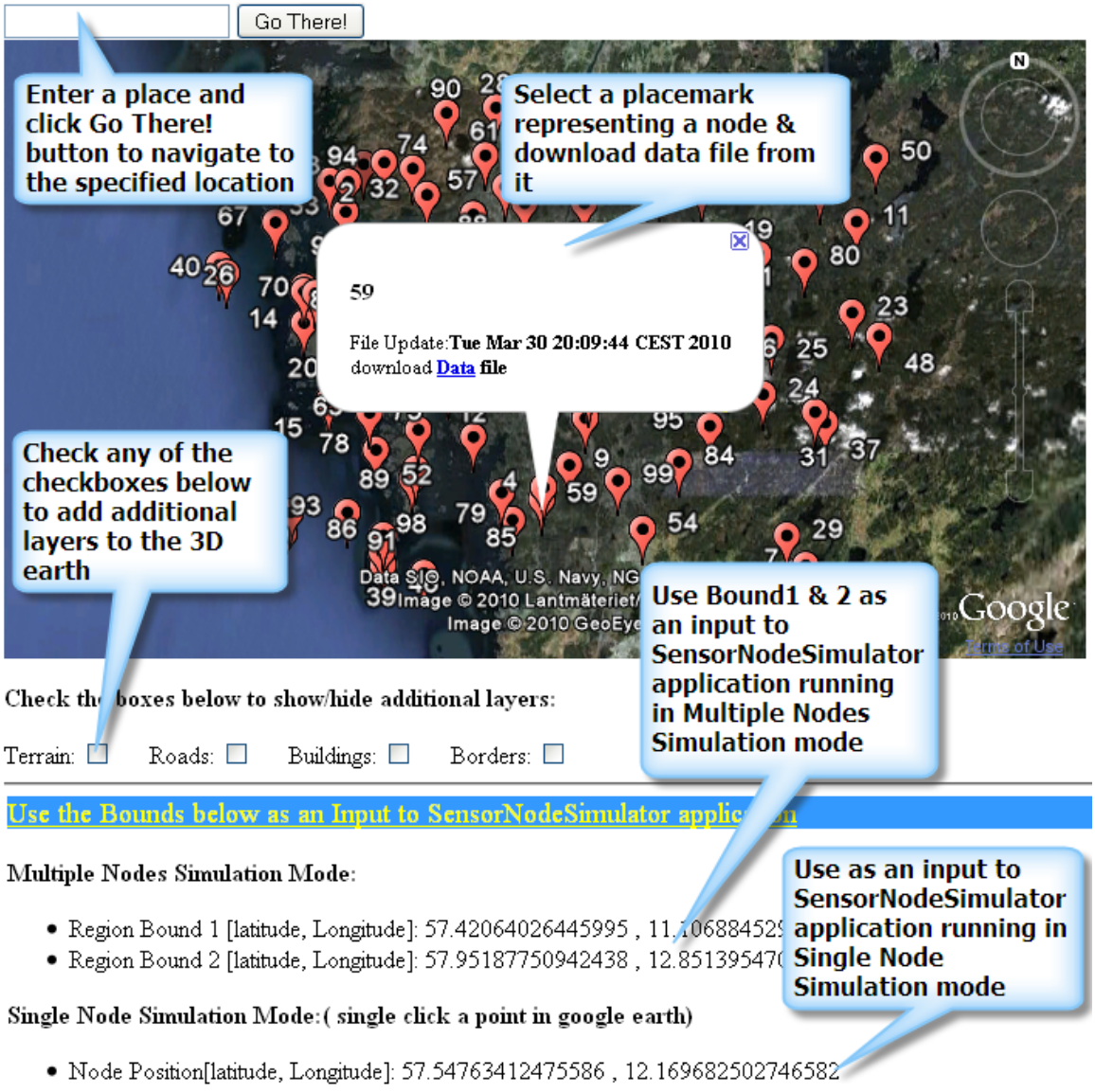

Figure A. 7: The 3D web application

In the figure above each sensor node is represented with a red place mark that holds a number, which represent the unique id of each sensor node. By clicking on a specific place mark a popup window appear that includes the unique id of the sensor node, a time stamp representing the date at which this place mark was updated with data corresponding to the sensor node, and a Data link, which allows the user to download data file corresponding to that node.

The user can also navigate to a specific location by entering a text corresponding to that location in the text box above the 3D earth window. The user then clicks the Go There button to direct

the 3D earth window to change its view to the new location. Additionally, layers can be added to the 3D window by checking the appropriate checkboxes below the 3D window. These layers give the users the ability to view terrain, street names, 3d buildings and borders between countries. Finally, the application generates some data that can be fed to the simulator application SensorNodeSimulator.jar. This data is found in the below the “Use the Bounds below as….” HTML section.