Università
della
Svizzera
italiana

Facoltà
di scienze
informatiche

Bachelor Project

Cyberspace

A Feasibility Study

Christian Ponti

supervised by
Prof. Dr. Michele Lanza

Abstract

A network system, such as the set of hosts of a university, is a complex system where each host provides services that other hosts use. During its evolution new hosts and services are added, increasing the complexity of the whole system.
We analyze networks and to accomplish this task we start to decompose the problem, characterizing the basic components, hosts, and extracting useful data. We explore hosts, nodes of a network, using the services that each node provides. Furthermore we represent services as units of a node, in such a way that it is possible to easily visualize the node with its components.
This feasibility study deals with the evolution of an application that, as final step, represents a network system, we call it the Cyberspace, and visualizes it.
The primary goal of the project is to build an application and to provide the user with a navigation system to navigate the acquired data. The application produces a new point of view from where the users have a better understanding of the Cyberspace and what it represents, the network system. Users have a different perception: they are wrapped around the scene and can interact.

Contents

Chapter 1

Introduction: from literature to the project

1.1 Origins of cyberspace

The word cyberspace appears first in literature in the early '80s. It was coined by William Gibson in his novel Neuromancer. The following is a portion of the novel, usually cited on the origin of the term:

Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts... A graphic representation of data abstracted from banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding ... [Gib84]

In 1992 the novelist Neal Stephenson coined another term, the Metaverse:

It is the Broadway, the Champs Elysees of the Metaverse. It is the brilliantly lit boulevard .... The dimensions of the Street are fixed by a protocol, hammered out by the computer graphics ninja overlords of the Association for Computing Machinery's Global Multimedia Group... Like any place in Reality, the Street is subject to development. Developers can build their own small streets feeding off of the main one. They can build buildings, parks, signs, as well as things that do not exist in Reality, such as vast hovering overhead light shows and special neighborhoods where the rules of three-dimensional spacetime are ignored. Put a sign or a building on the Street and the hundred million richest, hippest, best-connected people on the earth will see it every day of their lives. [Ste92]

Gibson and Stephenson abstract their reality building their cyberspace or metaverse. It is a projection of the reality that follows rules that authors see in the real world, enphasized depending on what the author highlight and show to the readers. They supply them a new point of view so that starting from there, readers can develop new

ideas and compare them with their every day life.
In the same way we abstract our reality, a networked system, and starting from our implementation let the user have a different point of view from which understand better the network.

1.2 Primary goal

The goal of the project is to build a new relationship between data acquisition and visualization, with an application that realizes it.
Data acquisition determines the quality of data, services on a given network, and data transport, the network itself.
Visualization shows data using a 3D model, evolving the old 2D model which does not allow an immersion in the visualized world. The analysis part is fundamental in order to understand problems related to visualization, like the more convenient metric and his representation in the 3D world, and related to networking and data acquiring, like the connection between participating peers and the way to send remotely complex objects.
The application is the concrete implementation of these concepts.
Not less important is the user perspective. To deal with this a client application will let the user navigate and change his point of view with respect to his needs.

1.3 Secondary goal

Since this is an open ended project, the secondary goal starts with the end of the implementation. The perspective will be different because a concrete implementation is something that allows additional considerations. Like an user sitting in front of a flat screen we start with new questions:

- Starting from these results, what can be optimized to build a fine grained application?
- What are the next metrics to add to the system to allow a better interaction?
- In which direction could go the evolution of the system among many possibles?

1.4 Structure of the document

The main document contains the following chapters:

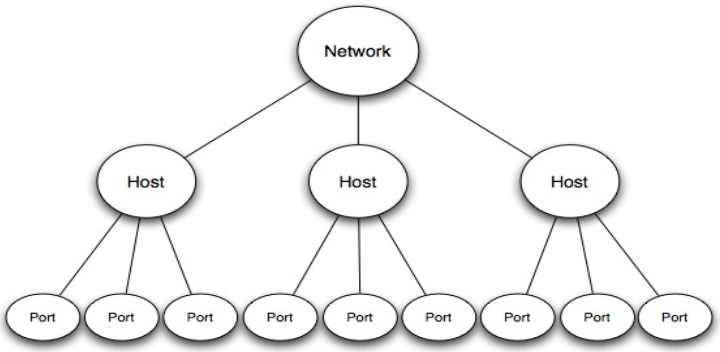
- In Chapter 2 we model the Cyberspace and we describe some problems related to networking and visualization.
- In Chapter 3 we first present the big picture and then we implement two scenarios, single node and many nodes, which lead to a first and final prototype.
- In Chapter 4 we discuss the conclusions of our project, and we introduce possible future evolutions.
- In Appendix A we show an instance of the system as UML sequence and the class hierarchy.

Chapter 2

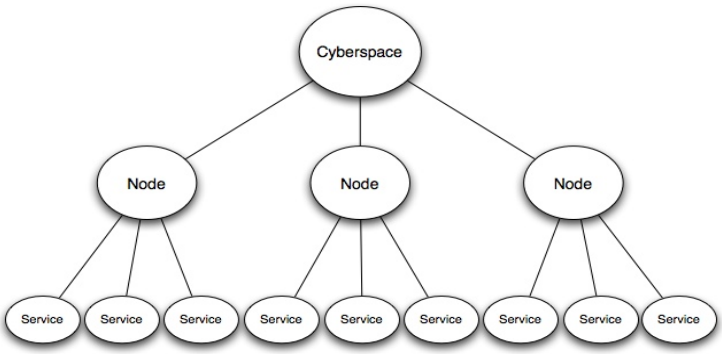
Conceiving the Cyberspace

2.1 Abstracting a network in the Cyberspace

The cyberspace is a 3D visualization that follows a tree structure where the cyberspace is the root, a node resides at the first level and each services at the second level (see Figure 2.1).



(a) Network model



(b) Cyberspace model

Figure 2.1: TCP/IP tree model VS Cyberspace tree model

To model the application we reverse the tree structure, starting from the leaves. A leaf is a service offered by a node, an host in the TCP/IP structure, by means of an open TCP/IP port. An active node has from one to many open ports and many nodes form the cyberspace.
The first prototype depicts this structure in a simple way. It is a port scanner that visualizes in a textual way nodes and open ports of a given network.
Our model evolves following the structure but modelling the acquired data and the visualization in different ways. First we intoduce the 3D visualization and experiment with it, than we model the application in such a way that follows the tree structure mentioned above: we let the node be responsible of the representation of its children, open ports, and establish relationships with other nodes exchanging data, allowing with these relationships the creation of root of the tree, the cyberspace.
The cyberspace is composed by two main parts:

- Computer networking
- Data visualization

Inside the first part we discuss about computer networking and data acquisition, while inside the second we describe the visualization of the acquired data.

2.2 Cyberspace = networking + visualization

Cyberspace is composed of a networking and a visualization part.

2.2.1 Computer networking

Computer networks abstracts the interaction between two or more participant who need to communicate. The way they communicate is modeled by the services offered and by the transport used to achieve the communication.
Cyberspace abstracts a network using the services offered by its participating nodes as data and its underlying network structure to distribute data among other nodes.

2.2.2 Data visualization

Visualization is a branch of science which goal is to allow for a better understanding and interpretation of data.
3D visualization is the projection of 3D objects on a flat surface, like a monitor. It allows to represent and model the reality as close as possible and provides a different point of view with respect to a textual representation.

2.3 A network is represented poorly

Every system needs a validation process in order to understand if its implementation reflects the requirements. A network can be validated by users of single hosts, but it is useful to have a global point of view. There is the need of an application that collects data going through every host of the network and reports the results. Such

data can be everything useful to validate the network, and depends on the purpose of the administrator. Open ports are good candidates because the service that stay behind an open port is part of the reason of the existence of a host. But other candidates can be useful too: the operating system, the topology of the network, with hierarchies between hosts, gateways, the amount of data processed by the network card, etc.
Applications usually represent networks of computers in a 2D textual way, allowing limited interaction between the user and the application and sometimes no interactions at all. These kind of applications solve the problem to have a basic representation and a global point of view, but nevertheless the textual representation is sometimes confusing and obligates the user to go through each line, losing the global point of view. This behavior depends on the amount of data acquired by the application: a small amount of data allows to mantain the global view but usually such situation does not need a complicated validation process. With the increase of data quantity, validation may become complex and requires a different point of view.

2.4 Local 3D visualization systems are static and slow

A local single machine 3D visualization system is usually static. The problem depends on the kind and amount of data we visualize and on the frequency with which data change. If there is a great amount of data the visualization produced is usually a shot, an instance that reflects the system at a given time but that can not catch easily the evolution of the system. The reason is the limited computation capability of the machine and the amount of time that the application employs to finish its task. This solution can be reasonable if data does not change frequently or does not change at all. However for dynamic systems of systems that must catch events when they happen the solution does not work.

2.5 Problem: distributed 3D visualization systems are complex

A distributed system allows hosts to share data and computational power. 3D visualization is expensive in term of computational power and CPU time and a distributed system can divide tasks in order to distribute the amount of work between the participating hosts.
Nevertheless a distributed system add levels of complexity to the application. Data is distributed as well and should be managed carefully in order to mantain its consistency.
This point is critical. In such a system the result, data visualization, is the sum of data supplied by the participating nodes, each one doing its task. The main problems are related to time and space. Data is acquired by each host at different time and processed for visualization. The concept of time in distributed systems is in some cases abstracted by the synchronicity of the processes: in other words, process A do its task at time 1 and when terminates sends a message, or data, to process B which registers the event as happened at time 2; process C blocks untill time 3, which means untill process B terminates and sends it a message. In addition time is related to space. Each node should manage the distance between peers and therefore the amount of time needed to

send remotely messages or data.
Some systems does not need a precise definition of time, although that data consistency should be assured. For example, the case where two nodes are in charge to scan a class C network and each is in charge to process half of the network: it is reasonable that the final visualization shows the results in order, from first address to the last one. However, depending on the mechanism, it is possible to have inconsistencies, because the correct order is not assured.
Errors and failures should be managed as well, in a way that does not affect the system and its purposes.

2.6 Related Work

This section provides some example of applications which use different approaches to solve a subset of the mentioned problems. Some explore only a problem and seems poorly related with our application, but ,although this, represent an important step during our implementation and let us improve our knowledge of the system.

2.6.1 Nmap

Nmap [1] is a free open source utility for network exploration or security auditing. It is designed to rapidly scan large networks as well as a single host. Nmap provides both a console and a graphical user interface and results are represented in a textual way.

2.6.2 CodeCity

CodeCity [WL07a][WL07b] [2] is a tool developed by Richard Wettel and Michele Lanza. The tool presents a 3D visualization approach in order to explore software systems written in object oriented languages. The city metaphor is used to represents the system, where each class is rendered as a building and city districts depict packages. The city can be traversed to give the user a sense of locality and a better understanding of the program (see Figure 2.2).

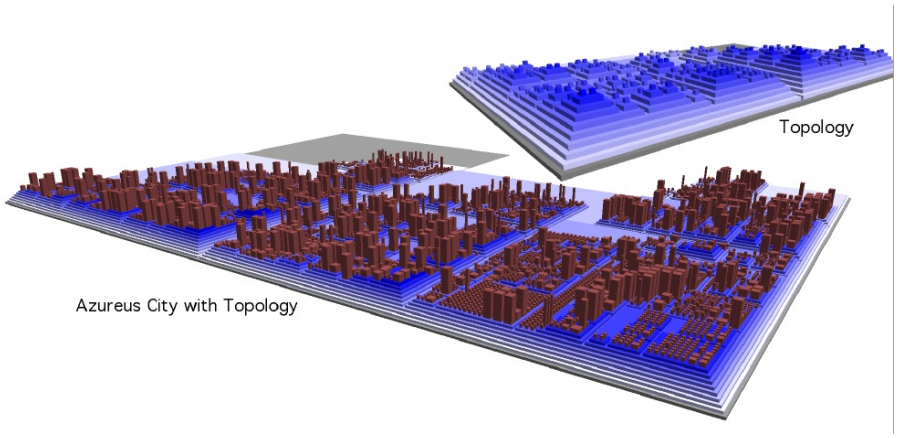


Figure 2.2: CodeCity

[1] http://insecure.org/nmap/
[2] http://www.inf.unisi.ch/faculty/lanza/Downloads/Wett07b.pdf

2.6.3 Metaverse applications

Metaverse applications [3] comes from a vision of Neal Stephenson in his novel Snow Crash. The term metaverse is used to describe the vision behind current work on fully immersive 3D virtual spaces.
There are many applications which started been developed following Stephenson's vision and exploring those 3D spaces using many angles. We explore briefly the followings:

- OpenCroquet
- Open Source Metaverse Project
- Second Life

2.6.4 Opencroquet

Opencroquet [4] is an open source software development environment for creating and deploying deeply collaborative multi-user online applications on multiple operating systems and devices. It features a peer-based network architecture that supports communication, collaboration, resource sharing, and synchronous computation between multiple users on multiple devices. Using Croquet, software developers can create and link highly collaborative cross-platform multi-user 2D and 3D applications and simulations - making possible the distributed deployment of very large scale, richly featured and interlinked virtual environments (see Figure 2.3).

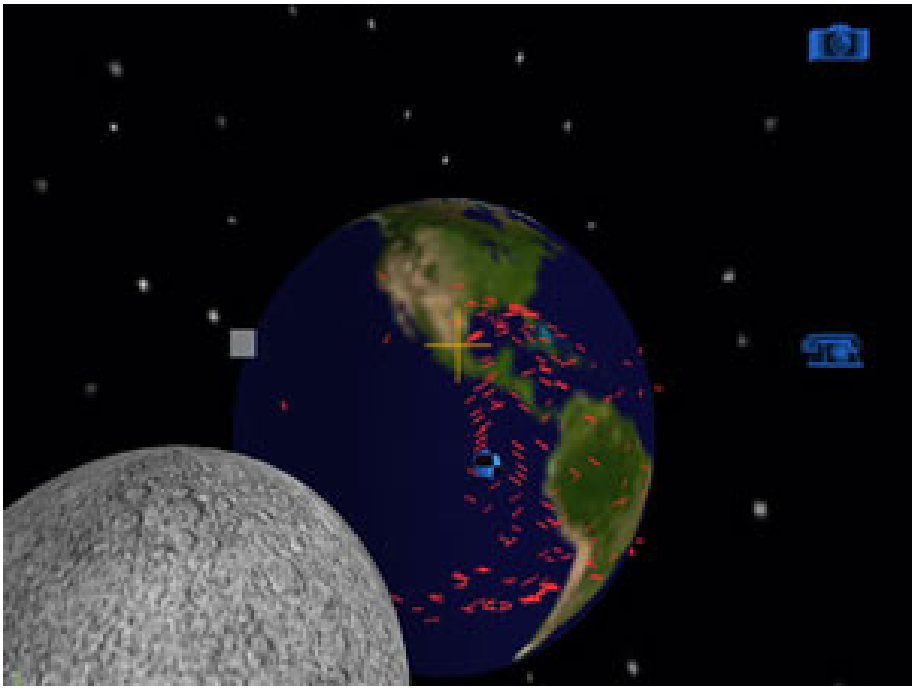


Figure 2.3: OpenCroquet: atmospheric simulation using a particle generator

[3] http://en.wikipedia.org/wiki/Metaverse
[4] http://www.opencroquet.org/index.php/Main_Page

2.6.5 Open Source Metaverse Project

The open source metaverse project [5] is a metaverse engine created because of the strong demand for virtual worlds that allows the customization by the player and creation of one's own world. The decision to build another metaverse engine, among the commercial existing versions, is the need of a flexible and scalable engine which is possible to customize in a way not possible with proprietary applications.
The application works similarly to a web server, the administrator can create his world and serve up data and contents of a 3D virtual world instead of web pages. The client can access the server to render the metaverse as seen by the player's avatar.

2.6.6 Second Life

Second life [6] is a commercial version of a metaverse engine developed by Linden Research Inc. It is an internet based virtual world that let the user interact using a client: the user creates an avatar and becomes a resident of the world. In second life world the user can interact with other users and explore the world. It is possible to participate to organized events in group or individually and create and trade items or services. Second life introduces the concept of virtual property and virtual money, the linden dollar, which is exchangeable for US dollars in a marketplace consisting of residents, Linden Lab and real life companies.

Figure 2.4: Second life

[5] http://metaverse.sourceforge.net/index.html
[6] http://www.secondlife.com/

Chapter 3

Implementing the Cyberspace

The solutions provided in this chapter reflect the evolutionary study and prototyping of the project. They follow the evolution of problems and solutions as they appeared during analysis or implementation of the system.
We start by showing the final result to give a first glance and then we come back to the beginning. We propose two scenarios and for each we illustrate the evolution. At each step we propose a solution to a given problem and its validation.

The following shows the structure of the chapter:

- The big picture: we describe the final solution with some screenshots to illustrate it.
- Development programming language: the section explains the reasons around the choice of Smalltalk.
- Single node development: we describe the first scenario wich ends with a 3D port scanner prototype.
- Many nodes: we describe the second scenario, built on top of the first, which ends with the final prototype of the project.
- The Client: describes some aspects of the user interaction.
- The GUI: illustrates the three GUI's of the application: main, configuration, visualization model.

3.1 The big picture

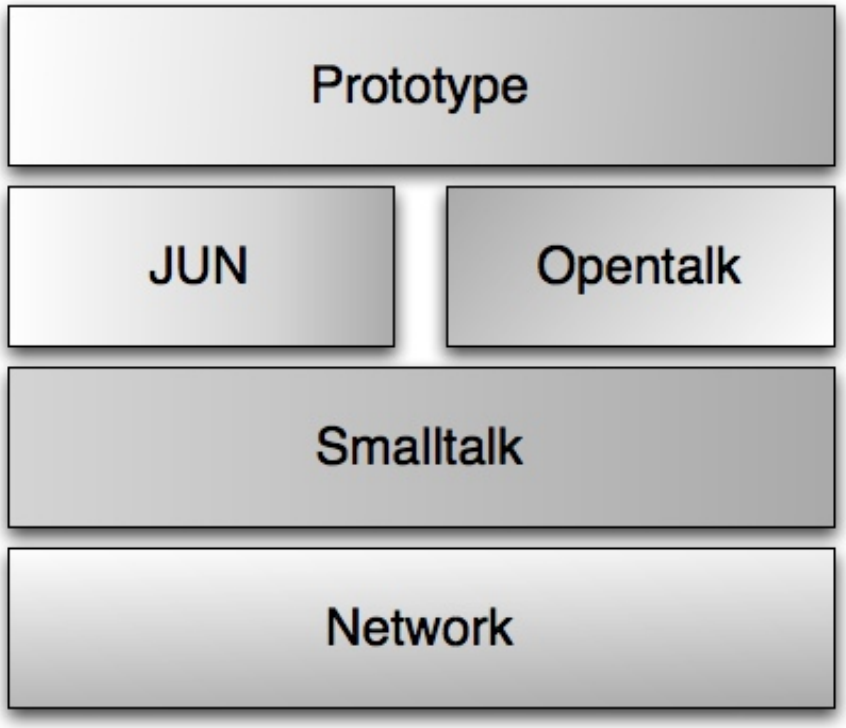


Figure 3.1: Cyberspace stack

We build our application as a prototype on top of the Cyberspace stack.
We define the Cyberspace stack (see Figure 3.1) with four components:

- *Network:* this component is offered by the underlying OS and basically allows all networking functionalities.
- *Smalltalk:* represents the Smalltalk Virtual Machine and the class library[1].
- *Jun:* it is a library supporting OpenGL binding[2].
- *Opentalk:* it is a library which allows the development of distributed applications.

Our prototype uses components and the offered functionalities to perform its task. The network is the means by which we scan a remote host and in the same time is the object of the scan. We retrieve information about a host connecting a Cyberspace node to a given set of TCP/IP ports and determining if such port is open or closed. Furthermore we use Smalltalk and its class library to manage data, and the Jun framework to model it in 3D objects and render the result on the screen. Finally we introduce Opentalk to bind together nodes and to allow each node to share the results among other participating nodes.

[1] http://www.cincomsmalltalk.com/userblogs/cincom/blogView
[2] http://www.sra.co.jp/people/aoki/Jun/Main_e.htm

The purpose of the prototype is to visualize in a 3D world the results of a port scan and share the results. At the end of the process each node visualizes the result of its scan and the result of the scan of each node of the Cyberspace. The visualization is a new point of view on the analyzed network.
To better interact with visualized data we provide a client with a simple navigation system. The user can navigate and be wrapped around the scene with the possibility to have a global point of view but also to go into detail (see Figure 3.2).

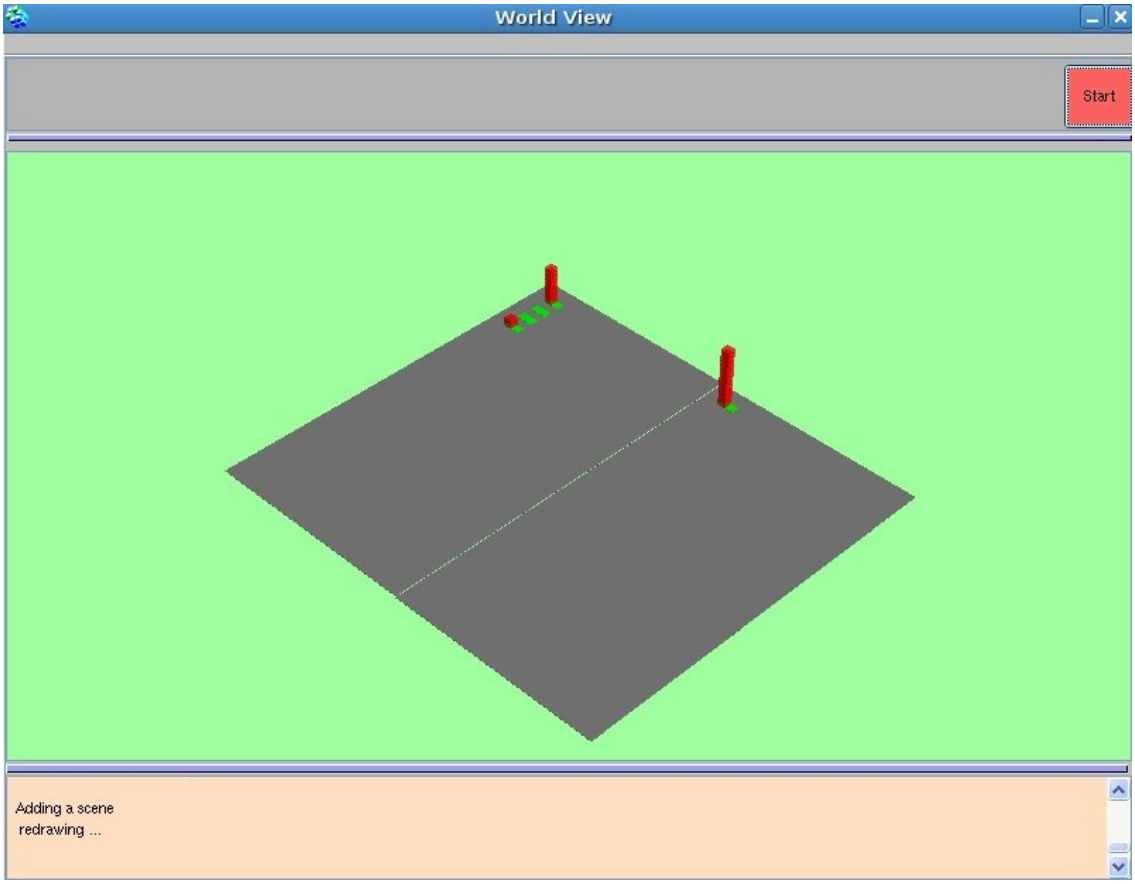


Figure 3.2: The main application

We develop our prototype following two scenarios:

- *Single node:* we start from a simple prototype and we add functionalities to obtain a 3D port scanner.

- *Many nodes:* we introduce Opentalk on top of the 3D scanner to build, over some steps, the final prototype.

Before the project discussion we introduce the language used to develop it.

3.2 Development programming language: Smalltalk

The developping language used for the project is VisualWorks Smalltalk.
There are several reasons to develop with Smalltalk. First at all we chose an object oriented (OO) language: we think that OO and the object structure is the best approach to describe reality, to model the world and its behavior. Smalltalk is considered a pure OO language with few primitives, built around a large library. The approach used by its developpers is to use a languange as natural as possible to assign classes and messages names, which makes it easy to understand and explore the class library.
Another important reason is to take the maximum profit from this project, not only in terms of a released application, but from a didactic point of view too. We learn in depth a new language and the better approach to do it is to start the development of an application: this allow us to use iteratively our understanding of the language to improve the application, and the improvement to gain more knowledge on the language.

3.3 Single node development

In this scenario a node scan a given network and represents the results on the screen (see Figure 3.3).

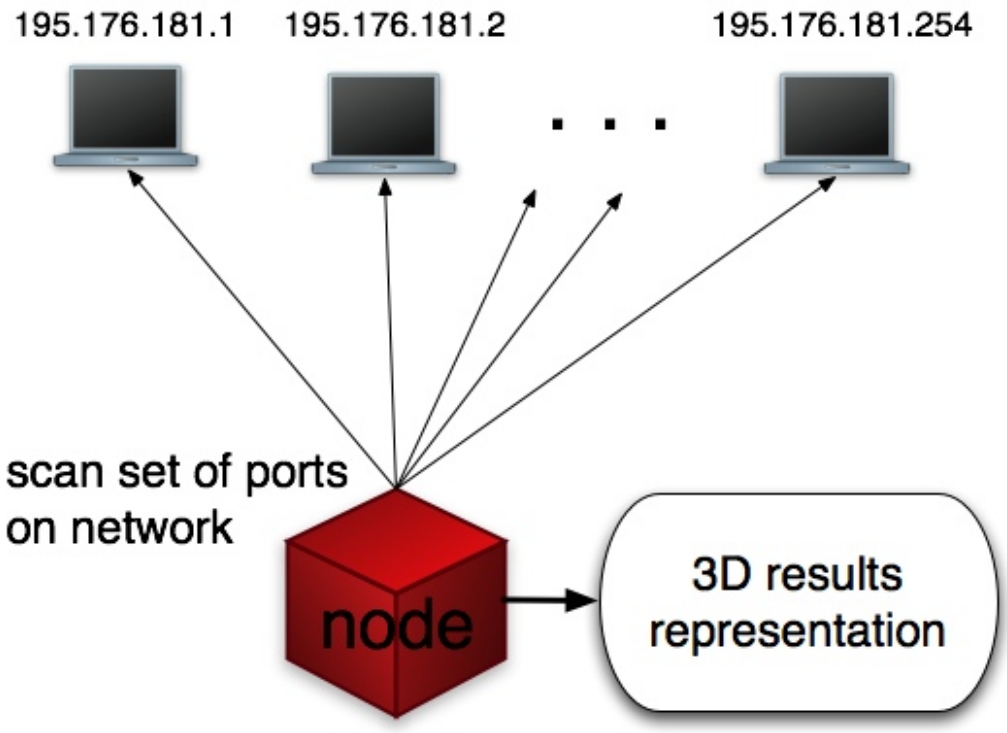


Figure 3.3: Single node scenario

We start to setup the scenario. Our task is to create objects to open a connection on a remote host and at a given TCP/IP port and to represent the result.

We start a procedure which leads to develop our task, discussing first a general issue which regards Operating System (OS) independency, then we divide our task, we develop the networking part and finally two ways to represent the results.

3.3.1 OS indepentent configuration

The first approach to the implementation of the system deals with differences between operating systems and the way the languages try to be as OS independent as possible. Our purpose is to automatically detect an IP Address from the underlying network card. There are two reasons:

- *simplicity:* if the node has only a network card it is simple to detect the IP address avoiding the use of external programs[3].
- *extension:* if a node has many network cards, for configuration purposes we are able to detect for each one the device name and its network address.

Unfortunately retrieving this information is not trivial in Smalltalk because there is not an OS independent way to do it.
This project is developed using Linux as OS and it seems that the only solution is to write C code to bind the operating system with Smalltalk using the DLCC class. This class binds the virtual machine with the underlying OS: for example it manages the connection between Mesa driver, which handles OpenGL on Linux systems, and the Smalltalk image.
Since this leads to an intensive effort, with no great gain for the project, we decide to postpone the probem to another release and to insert manually the IP address of the node.

3.3.2 A simple port scanner

The port scanner is the first simple but complete attempt to join networking with visualization. There are objects sent to a host, in this case the request to open a connection, and if an answer follows, the first data to represent, the port number and the address of the host. The scanner analyzes a class C network[4] and data are presented to the user in a textual way.
The first issue is the set of ports to scan and the derived slowness when the scanner analyzes the whole class C.
The amount of ports to scan depends on the number of addresses multiplied with the set of ports, and each connection requires time. Therefore the set of ports should be chosen carefully, taking into account the purpose of the scan[5]:

- *Complete scan:* if we perform a complete scan of the network, to verify possible security issues, we choose a large set of port (e.g. ports from 1 to 10000). We should take into account a long amount of time to perform the task.
- *Fast scan:* it is useful to scan the most known ports (e.g. ports in the rangle 1-1024) to discover services offered by a host or network.

[3] ifconfig, for unix based OS, ipaddr for windows
[4] IP addresses class with range 192.0.0.0-239.255.255.255, http://en.wikipedia.org/wiki/Classful_network
[5] http://www.iana.org/assignments/port-numbers

To deal with connection slowness, the solution is to add a timer after which the connection is closed. It should be carefully chosen, because if time is too short we may stop a connection before discovering that the port is open. We decide to insert a time out of 50 ms, value chosen experimenting and measuring the average response time with a ping request, which is about 27 ms. This is a reasonable approximation with an high speed connection.

3.3.3 Textual visualization

Data acquisition is not enough for our purpose. We represent data and we begin with a simple textual representation. As soon as we discover an open port we visualize the number on the screen and if it is a known port we visualize the name of the service.
This kind of representation can be useful, it is fast and does not require a lot of memory. However it has a great drawback: it is not possible to have a central point of view. The user is forced to scroll up and down to evaluate the results.
We need a better point of view, something that give us a complete view of the network. This leads us to explore another way to represents data, 3D visualization.

3.3.4 3D Visualization

This is the last step to implement the scenario. We decide, after some experiment with a textual visualization, to move in the domain of 3D visualization. We build a scene, a composition of 3D objects, and associate each object to a piece of data. The scene represents an image of the open ports of a network and determines a global point of view for the user.
The first issue regarding 3D visualization is the technology. The implementation of OpenGL in Smalltalk is provided by the Jun Framework.
The framework provides two main classes, used for this project:

- *JunOpenGLDisplayModel*: it has ApplicationModel has superclass and models application specific behavior, like UI builder, keyboard hooks, etc.
- *JunOpenGL3DCompoundObject*: it is a collection of 3D objects which can be passed to a DisplayModel to be rendered on the screen.

3.3.5 3D port scanner

We apply 3D visualization to the simple port scanner and we obtain a 3D port scanner. To accomplish this task we build a relationship between data and objects that represents it, we define metrics.

3.3.6 Metrics

The first chosen metric to visualize ports is a rectangular 3D object with high proportional to the port number (see Figure 3.4).

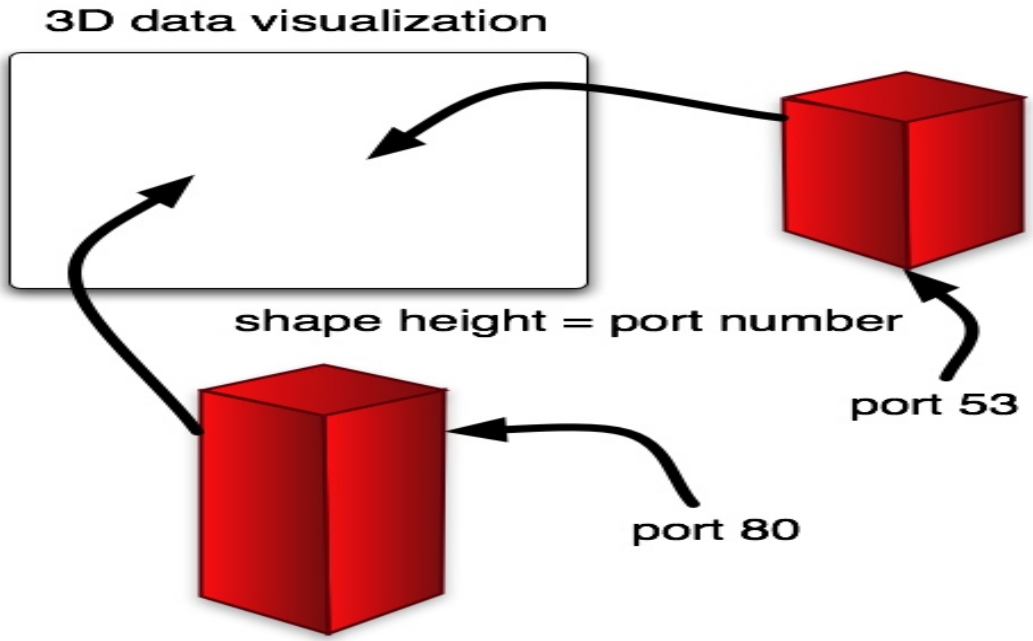


Figure 3.4: First metric attempt schema

We experiment with this schema and try to validate our choice.
We observe that the prototype produces shapes with different height, depending on the port number. However the result does not satisfy the global point of view, which is one of the goals. Our goal is to enphasize the number of open ports on a host, but giving a different height to a shape we enphasize the port itself and the first impression is that an higher shape means that the related port is more important.
We recall the graph of the network tree model (see Figure 2.1). We represent the network with the scene, but inside it we distinguish hosts and TCP/IP open ports. For both we identify an address or a number. In addition we know how many hosts are in the scene and how many open ports each host provides. The solution is to give the same shape to the hosts and identify each one with the 3D representation of its address. In the same way we give the same shape to each open port and we identify each port with a 3D representation of its number (see Figure 3.5. With this metrics we preserve the global point of view, because the user can identify at glance the whole scene. Furthermore we allow the user to go into detail and identify the address of a host and the number of each open port.

3.3.7 First validation: discussion arount 3D port scanner

The 3D port scanner is a very important step on the development of our application because it is the first prototype. The prototype complete our scenario (see Figure 3.3). Its components are a networking part and a 3D representation of data.
To validate the prototype we build an experiment. We decide to scan the internal subnet of server of the USI university, at network address 192.168.64.0 (see Figure 3.6). We define the set of ports to scan, which is from 1 to 10000. We validate the prototype and then we extract some considerations from the results (see Figure 3.7).

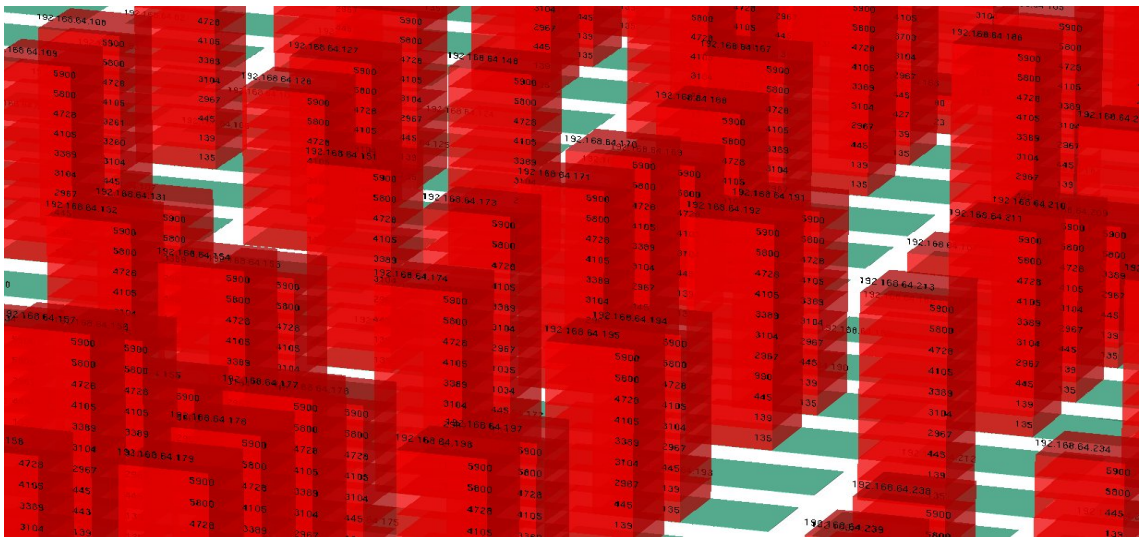


Figure 3.5: Buildings represent hosts, each floor is an open port

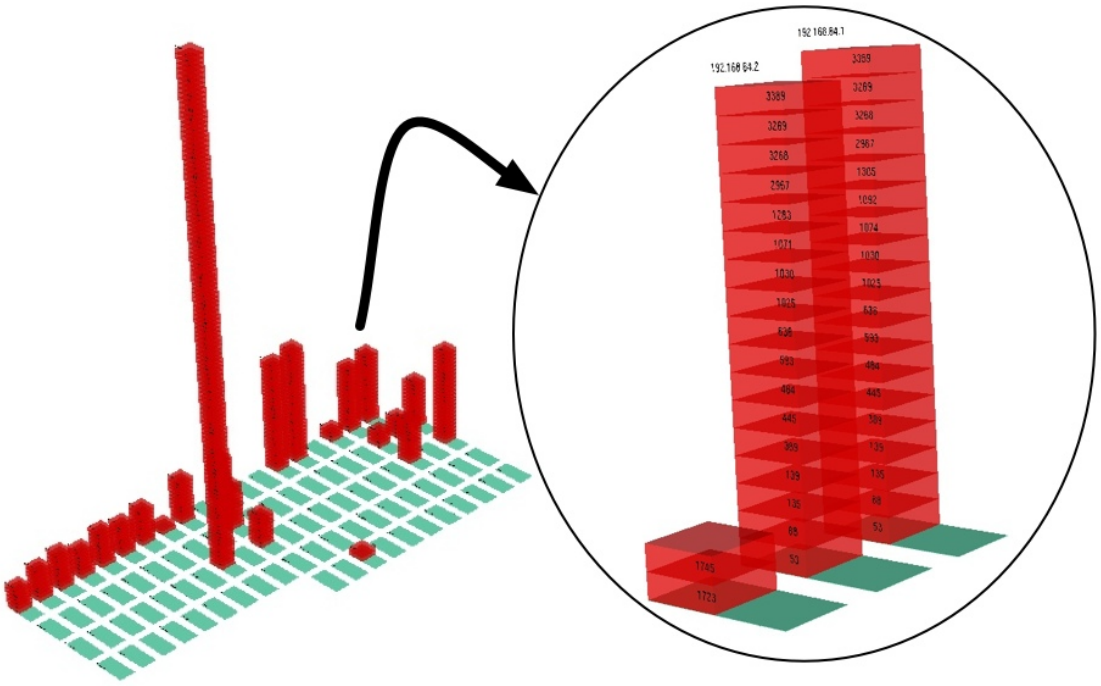


Figure 3.6: 3D port scanner experiment

3.3.8 Prototype validation

To acquire data we employ about 13 hours. The performance of the prototype changes during time: during the initial phase the process is very fast. The mean velocity is heavily influenced by the connection to each port. However, after a while data visualization begins to weigh on the system, which slows down. The bottleneck, the slowest component of the system, has changed, being now the visualization part.

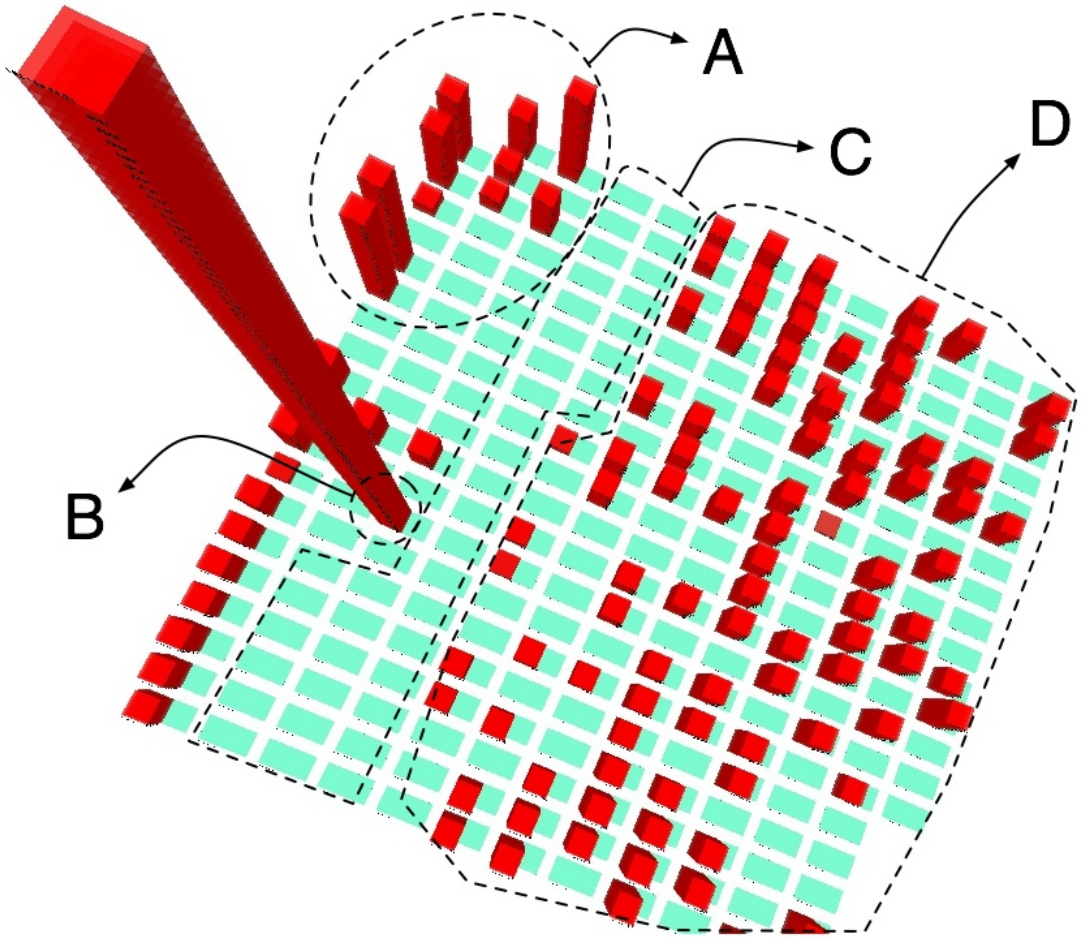


Figure 3.7: 3D port scanner experiment: final result

3.3.9 Results validation

The results of the experiment allow us analyze the network (see Figure 3.7) with a global point of view.
We divide our scene in four sections, each one representing a different zone:

- *zone A:* it is a little group of hosts on the upper left corner, but hosts have more open ports with respect to the mean of the network.
- *zone B:* this big tower is the real anomaly of the network, with its very high number of open ports.
- *zone C:* this zone shows an empty section of the scene, with no hosts or hosts without open ports in the range of addresses that represents.

- *zone D:* this zone encloses more than half of the addresses, its hosts are more spread and at glance there is not a great difference with respect to the number of open ports.

We start our discussion from zone B, which represents a great anomaly on the scene. The great issue that we see in this big tower is security. A network administrator should ask himself what is the purpose of this host, which kind of services are provided and if such services are necessary. In the last case may be possible to avoid such configuration and distribute better the offered services. For example a malicious attacker can perform a denial of service (DOS) attack [6] which may shut down the host and all its services. If there are hosts, maybe in the same subnet, which uses its services the consequences can be even worse, because other services may become unavailables too. Zone A represents a group of hosts in the upper left corner where there are many hosts with an important number of open ports. Zone D encloses the majority of hosts in the network. The global point of view does not highlight security issues but it is possible to draw some hypothesis:
it could be the case that zone D includes more general purpose hosts which share the general charge of work, providing better availability in case of failures for important tasks, while zone A provides ad hoc services, which are important but not critical for everyday life of the users.
Finally zone C is empty. It is possible that network designer need a set of addresses for future use, to add hosts and services without being forced to design again the network.

3.4 Many nodes

In this scenario we take our prototype and we add functionalities until we obtain many 3D port scanners, many nodes, which interact and exchange the results (see Figure 3.8).
Our purpose is to scan a given network and represents the results on the screen, but our purpose is to divide the workload among nodes, taking in consideration our global point of view.
To build the prototype we introduce the idea of a distributed system, which is the means by which we divide the workload between nodes. We shortly discuss about some existing frameworks for distributed systems and then we start our development, adding at each step some functionalities. Furthermore we build a protocol to let nodes communicate and share data. We introduce then a client prototype with a navigation system: the user can connect to a node and navigate the result of the scan of the participating nodes, starting from a global point of view but with the possibility to go into detail. Finally, after some experiment, we decide to add the world positioning system (WPS), which purpose is to mantain data representation consistency among nodes.

3.4.1 Distributed System

We divide the workload of a 3D scan with the purpose to speed up our task. We decide that a node is a 3D port scanner which interacts with other nodes. We divide the workload of each node among all the nodes of the Cyberspace.

[6] http://en.wikipedia.org/wiki/Denial-of-service_attack

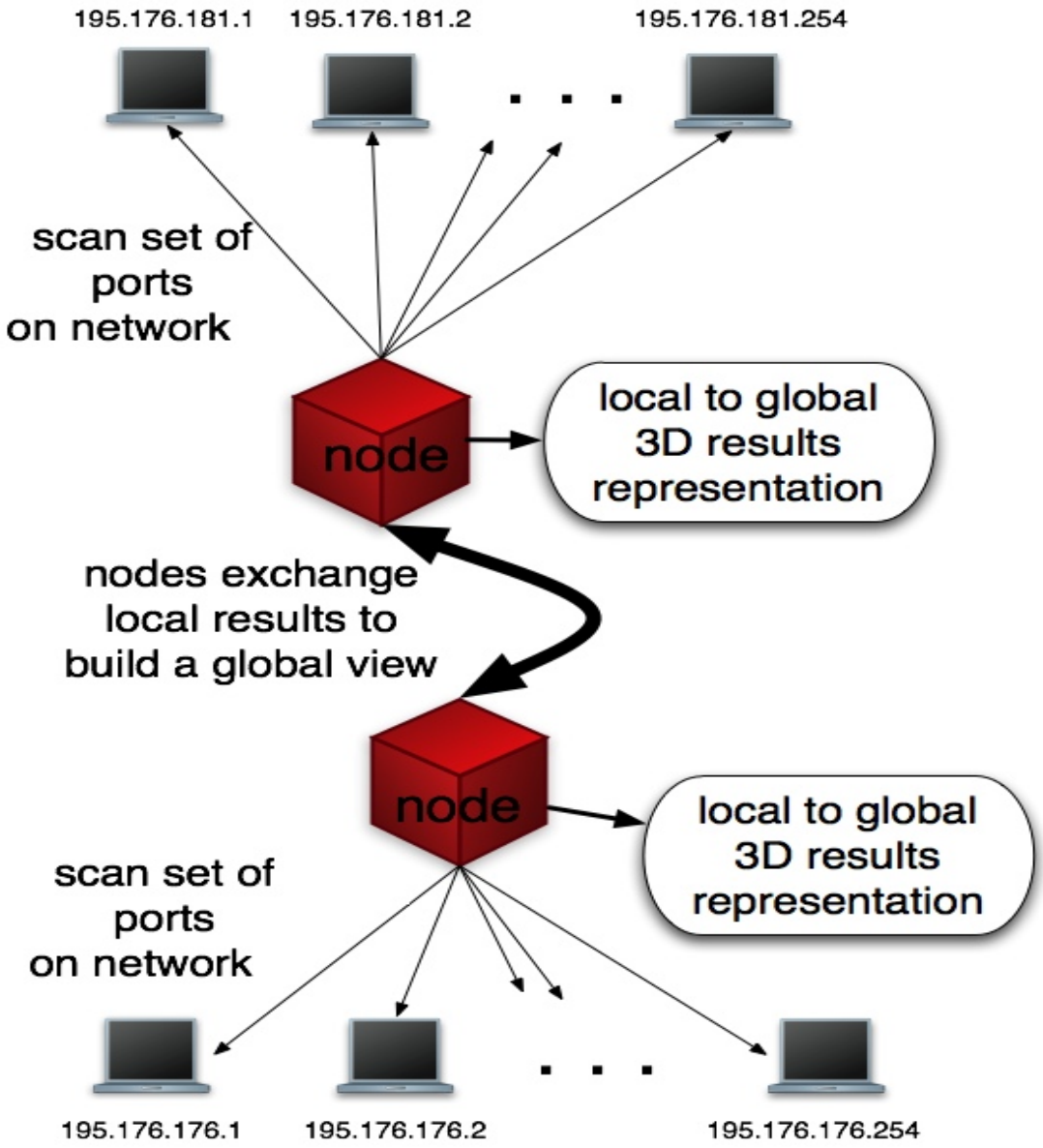


Figure 3.8: Many nodes scenario

To distribute the task among nodes we introduce a framework for distributed system, which provides such functionalities. We discuss about the two main technologies for distributed systems, and then we face a critical topic: object passing. Furthermore we procede, with a step by step approach, on the implementation and testing of different approaches to build a distributed system.

3.4.2 Technology

In Smalltalk exist two major framework to build distributed applications: DSTSmalltalk and Opentalk.
The documentation shows that Opentalk is an evolution of DSTSmalltalk. It provides

more functionalities and is well mantained and supported by the implementors. We choose it as the framework for the project as the next sections illustrate.

3.4.3 Opentalk: object passing

Passing objects remotely is critical. This section presents an evolution of steps to study problems and solutions, new approaches to take iteratively, producing an evolutionary prototype that merges the two main part of the project: visualization and networking.
The main feature of Opentalk is to export objects, so that an object can remotely reference them. There are three ways to send data to an exported object: passing by value, by reference and by class.
The more interesting are the first two. Passing by value means to send a copy of the local object to the exported one, so that in the distributed system exist two copies of the same object. Passing by reference send remotely a reference to the local object; in this case there is only a single object in the system.
We start experimenting with object passing in the next sections, which describe a step by step approach to understand the best way to pass objects across a network. We first add Opentalk functionalities to the prototype for testing purposes and finally we build the interaction between nodes.

3.4.4 Step 1: first experiment

The first experiment concerns the use of the 3D port scanner and visualize data across the network using Opentalk. We define a visualization object (CSVisualizationWindow, subclass of JunOpenGLDisplayModel) and a compound object (CSVisualizationComposition, subclass of JunOpenGL3DCompoundObject) and we export both using the Opentalk object exporting mechanism. A testing class opens requests to send data and pass them by reference, which is the default behavior, and let the remote visualization draw them. First chunk of data is sent and visualized properly (the address object, represented with a bluemarine square on the ground), but after that there is a series of time out invocation and nothing more is displayed. After some tests turns out that only one message is sent, and this is the reason why the address object only has been visualized. The following is part of the log messages that opentalk visualizes:

```
evaluatingMessage: STSTRequest[206]
{a Cyberspace.CSVisualizationWindow reDraw:
a Cyberspace.CSVisualizationComposition}
in: Transport<tcp/192.168.14.225:60110>
failedBecause: an Opentalk.OtETimeout{Remote invocation timeout!}
```

```
evaluatingMessage: STSTRequest[11331]
{JunOpenGLRenderingContext(a JunMesaPixmapHandle) polygonStipple:
RemoteObject<1739@192.168.14.225:42698>}
in: Transport<tcp/192.168.14.225:39726>
failedBecause: an ExternalAccessFailed
```

3.4.5 Step 2: experiment evaluation

To solve the problem we evaluate what appears from what should appear, which is the right behaviour.
First chunk of code builds a square (the ground of a building) and draws the square on the viewfinder[7] on the remote computer. Then we comment the remaining code and it work properly, every square been drawn on the viewfinder, as opposed with the situation before where only first square was drawn.
We continue to test adding last chunk of code, the one responsible to send and draw an opengl object representing the string of the ip address of the scanned host. It does not work. This result leads to new experiment and some conclusions: problems are caused by instruction regarding this Jun representation of a string.
For the next experiment we comment all the messages regarding Jun representation of strings. The experiment is successful, in the viewfinder on the remote computer buildings ground and floors are visualized correctly.

3.4.6 Step 3: testing JunOpenGL and Opentalk

We perform three tests, we send over the network a cube, a string and a given number of rectangles.
The first test passes without any problem.
The second test failes. This test leads to a consequence: we avoid to send Jun strings across the network. However, it is possible to draw some hypothesis.
From the opentalk log shown above is possible to extract where the code fails:
$JunOpenGLRenderingContext(aJunMesaPixmapHandle)polygonStipple$
The chunk of code shows that the application tries to manage JunMesaPixmapHandle. Mesa drivers are responsible to manage OpenGL operations on linux and Jun manages the binding with those drivers with the mentioned DLCC class. While locally it works, there is something wrong when we send to remote a string. A deep insight into Jun and Opentalk can provide more information to solve the problem.
The third test is critical. The central point is the number of compound objects sent across the network: with 50 rectangle objects, at the moment of drawing them on the viewfinder, the system handle a timeout, but catching it and continuing the operation leads to a correct visualization.
Testing shows another problem regarding brokers initialization, which require time.
It is not convenient to have a single method which starts server and client broker initialization and than perform the test, the time is not enought and a “message is not on the server exception” is raised. The solution requires that different processes start the server side and initialize it, than start and initialize the client side. Only after that the given task can start.

3.4.7 Step 4: handling timeouts in Opentalk

We face to the timeout issues which is critical for our prototype.
A timeout has two consequences: a wrong data visualization, i.e. some objects are not visualized, and blocking. The problem is critical because a timeout blocks operations,

[7]The viewfinder is the default gui obtained sending the message open to a JunOpenGLDisplayModel, with predefined functionalities like zoom and light management

causing new timeouts and blocks.
The domain of this problem is Opentalk and object passing. We procede with some experiment to have a better understanding and focus with more detail the domain.
Opentalk uses request brokers to provide transparent remote communication between Smalltalk images and represent the communication layer to communicating applications[8]. Request brokers support different type of configuration, with a default configuration that we use during the implementation of the application.
The first part of the problem regards timeouts: we solve the problem increasing the amount of time before a broker send a timeout message. We have an improvement, less initial timeouts, but the blocking problem cause them to appear soon or later.
We focus the blocking issue at request broker level. To solve the problem we change configuration, overriding the default message to create an instance of a broker. The adaptor object handle network transport protocols and support two type of communication: symmetric and asymmetric. Symmetric communication let a process start a request and block untill the request is accomplished. Asymmetric communication instead let a process start a request and continue operation without waiting. The default behvior of a broker is to block and wait, which is part of the problem we encountered. We solve it changing to the asymmetric communication.
The prototype works properly but is not optimized. During next section we introduce the interaction between nodes, which leads to the final general configuration of our prototype. Finally we describe some optimization tricks.

3.4.8 Node model and nodes interaction

The new important concept regards the node. The node is a single cyberspace entity with an address and a task (i.e. the port scanner) to perform. It contains a visualization, that is, its graphical representation, and a control network to exchange messages not related to its behavior.
We model and implement the node. A node has a visualization model, a control network and a composition of 3D objects acquired by means of its assigned task or sent by another node. To allow the last feature, it exports its composition object and visualization model object using the Opentalk framework. We introduce the control network, which manages the initialization protocol. Its responsibility is to put itself in listening mode and wait for communication. Communication type regards the hook and initialization of a node in the Cyberspace and a protocol which allow to send remotely commands not related to the specific task that a node perform, which in this prototype is port scan. The control network allow to build a network of nodes which cooperate exchanging the respective data.
To define the behavior of the control network we create a protocol:

the scenario is composed by a first machine (M1) which starts and wait for a connection. On the other hand, the second machine (M2) starts following the initialization protocol to communicate with M1. M2 connects with M1 because M1 is on the knownNodes of M2, as first access point. M2 sends his address and M1 register this address to his known neighbours and send back an answer ' got neighbours '. This is the key to send

[8]Cited from OpentalkDevGuide distributed with the framework documentation

command 'start' (from M2 to M1): when M2 receives it, send his city[9] to remote and starts the port scanner. In the same time M1 sends his IP address to M2 which register it as neighbour and when M1 receive back 'got neighbours' it sends command 'start' to M2.

3.4.9 Prototype optimization

Every node exports a visualization object and a composition object. We analyze some features of this two components.
A visualization object follows the MVC pattern. Thew view (typically a user interface) renders the model (data) and the controller allows user interaction processing and responding to events.
A nodes has a visualization model with its data been the composition object, which changes over the time during data acquisition, in contrast with the view and the controller side which does not change and it is common to all nodes. The first solution to export both the visualization and the compound objects derives from the idea to divide what changes from what does not change. The idea is a reasonable assumption, but the implementation is wrong. A node is interested in another node only for the different data acquired and this concept is well abstracted by the compound object. A node is interested in the visualization too, but the features it provides is what the node locally knows, whitout the need to interact remotely. The view and the controller have the responsibility to process themselves, which means draw data on the user interface when new acquired data arrives. This leads to the conclusion that there is no need to export the visualization object, but instead export the compound object and let the visualization process itself locally.
We improve the prototype accordingly and the first solution is to use a thread that control the visualization, redrawing the scene every ten seconds. This implementation approaches the solution but is not optimal: there is a sensible overhead due to redraw the scene even when there is nothing new to draw. In addition if new data arrives hardly after a redraw, it should wait ten seconds to be processed, causing a delay.
To obtain a better solution we use the Smalltalk dependency system. Every time new data are added to the scene, i.e. a compound object add to itself a 3D object, we send a changed message to the compound object. A watcher express interest on it and is responsible to detect any change on this object. The watcher catch a change and send to the visualization a redraw message.

3.4.10 WPS: World Positioning System

Untill this point each node perform its task and send the acquired data. At the end of the process each node has in its visualization a world composed by as many city as there exists nodes.
The problem is the coherence between each visualization: each node has its city drawn at point zero and the other city drawn with a given offset, ordered with respect to the request to join the world. This ordering system is wrong and we decide to implement a World Positioning System with as responsibility a location assignment management.

[9] the term city describes the ground, the visulization of the phisycal location of the node in the world, the set of all nodes

WPS is managed by the first peer, the first node entering the system and therefore the one which creates the Cyberspace. The node, by means of the WPS, assigns to itself location zero and stores the assignment in the positionRegistry. As soon as an host/node join the Cyberspace the WPS assigns it a new location. The node then initialize itself with the assigned location, so that, even locally, its local position reflects its place in the world.
To accomplish this task we implement a new version of the protocol, as described in the next section.

3.4.11 New version of the initial protocol

The new version of the protocol includes four new messages: after the setting of the neighbourhood to know which node participate to the Cyberspace, the new entering node ask the first node for a location. The protocol implies that the first peer manages WPS and answer with the proper location. At this point the entering node send a ready to start message and receives a start message from the first node, which starts the task (see Figure 3.9).

3.5 The Client

We build a simple client to allow an user to interact with the acquired data.
The client connects to a known node and requires data to visualize in its user interface, allowing an user to explore them with a navigation system (see Figure 3.10).

The following sections describes some consideration about the client.

3.5.1 General Design

During client design came out an issue: the code of the client is very similar to the code of the application, so that it would be possible to merge the functionalities.
From one side, this solution produces an application with no duplicated code, with some advantages: the whole application is more compact and during install or testing of the node behavior, an administrator have a better understanding on which is the result of data acquiring. The disadvantage is that during production forces the developper to distribute the whole application, client and main application. For example a scenario can be characterized by a large network where each host run a version of the application. After configuration, the network administrator needs only a copy of the client to explore data and evaluate the network. However a copy of the client is on each host, adding space overhead to the set of machines of an organization.
After these considerations we prefer to divide the main application from the client. This decision try also to anticipate some possible future evolution scenarios where, for example, an user is outside an organization but can however access its remote data. The user should be allowed to use the client and it is not interested on the main application which, in a future evolution, can have security issues if it is accessible by the user.

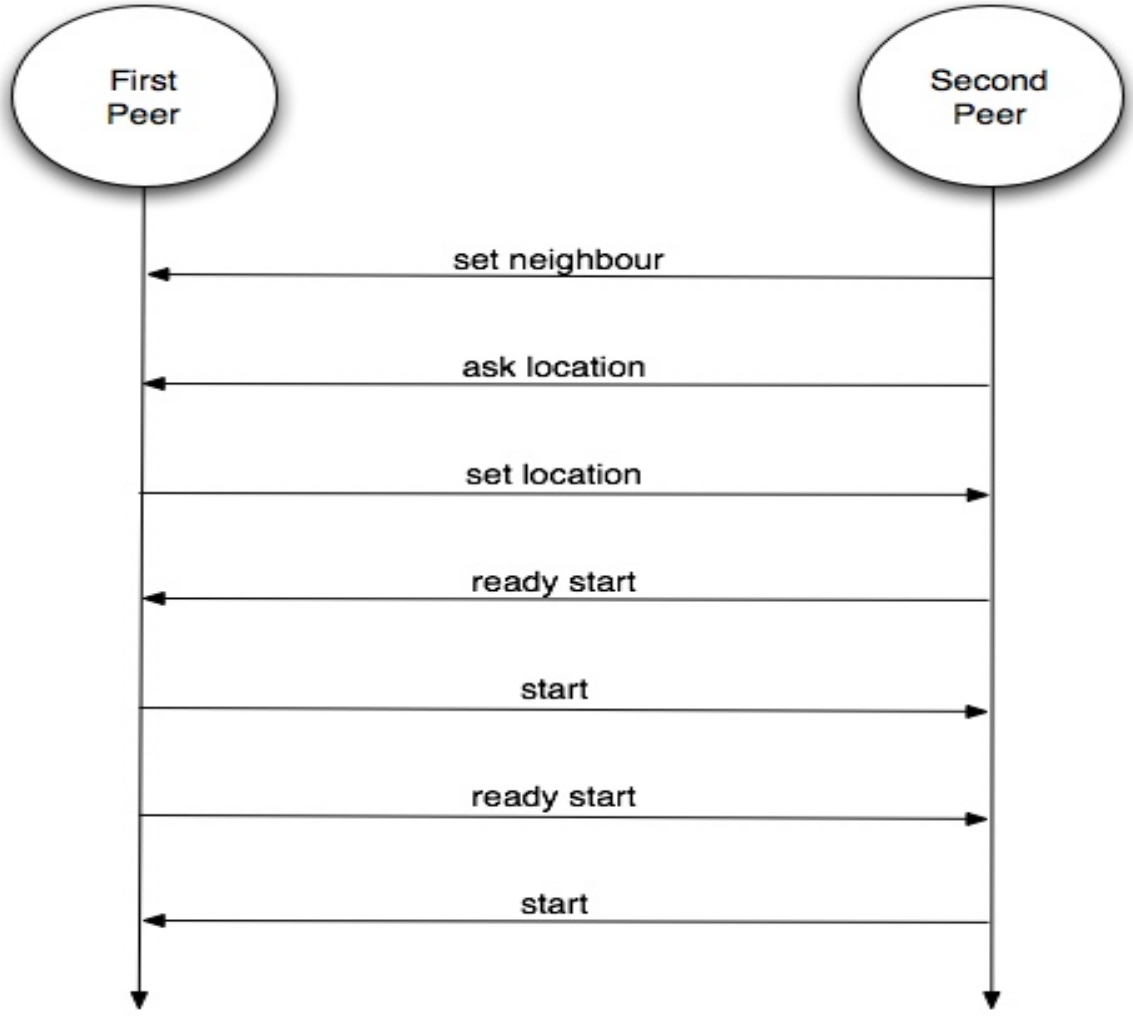


Figure 3.9: New version of the protocol

3.5.2 Navigation System

The navigation system is critical from the user point of view. It is the means by which an user explores the world and interacts with the system.
We build a basic navigation system which works modifying the opengl camera control.

The camera control (see Figure 3.11) is composed by an eyePoint, a near distance, a viefinder, a sightPoint and a far distance. The eyePoint is the location of the eye of the user, who points following an imaginary line to the sightPoint. The length of this segment is the distance between eyePoint and sightPoint. The viewfinder represents the viewport, that is, the projection of the 3D scene over a flat surface to simulate the third dimension. Near and far are two distances that represents what is before and what is after the viewport.
The navigation system works having the distance between eyePont and sightPoint constant and moving the two extremes by a viewfactor to move on and back, and changing

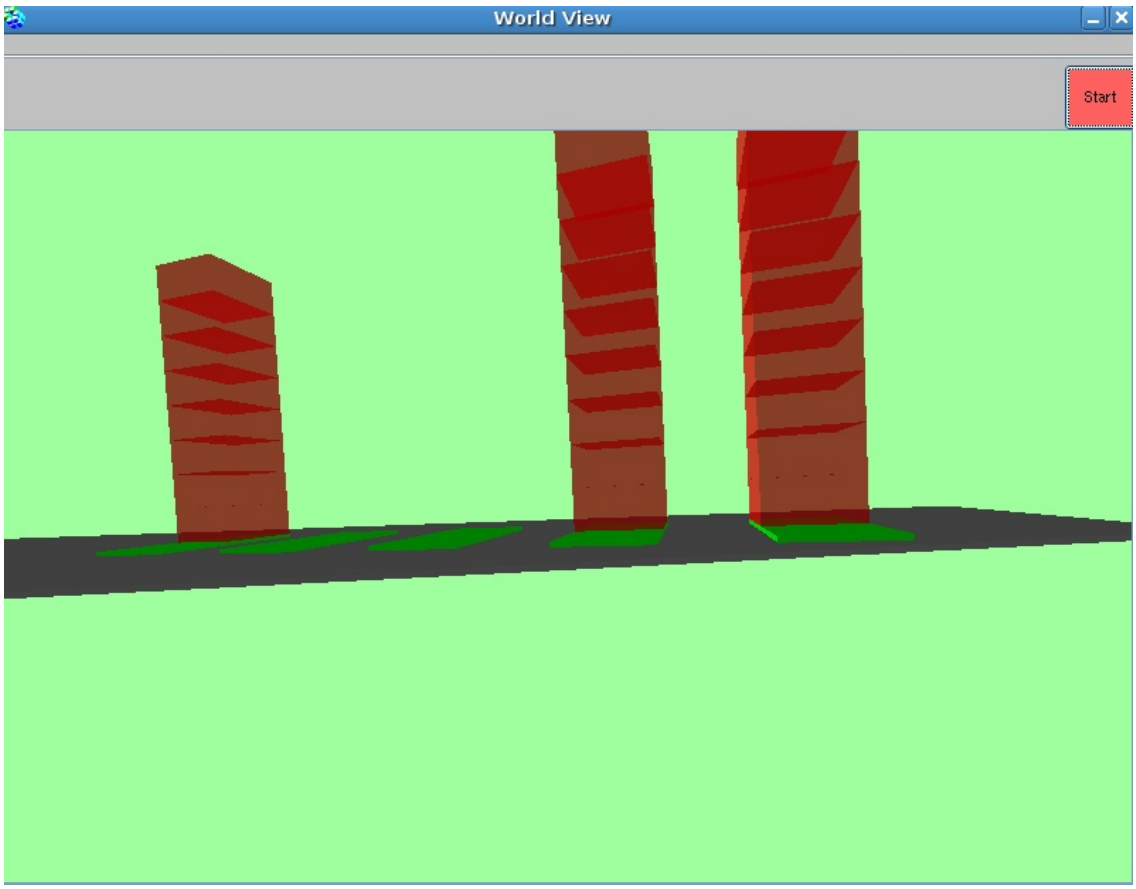


Figure 3.10: Client GUI with navigation system

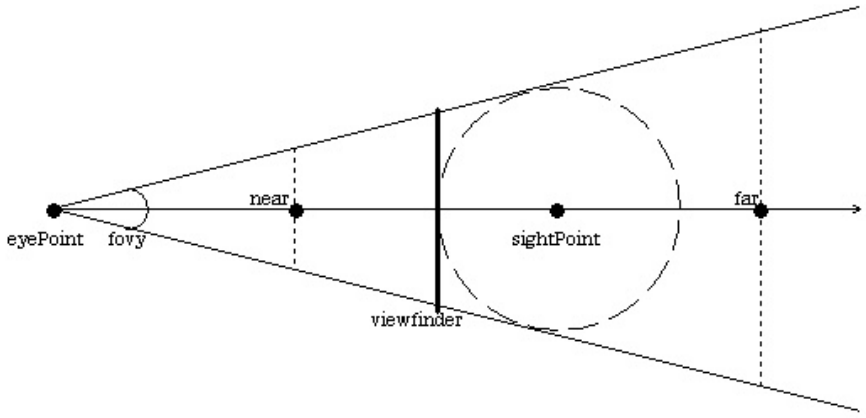


Figure 3.11: The camera control in JunOpenGL

the angle between the segment and the vertical plan to turn left and right.

The way JunOpenGL is implemented does not allow the developper to modify directly the value of near and far. Instead (see Figure 3.12) it is possible to use the ratio between the distance, the length of the segment, and the viewfactor, to modify this two values. For example, a larger viewFactor leads to have a near location closed to the eyePoint

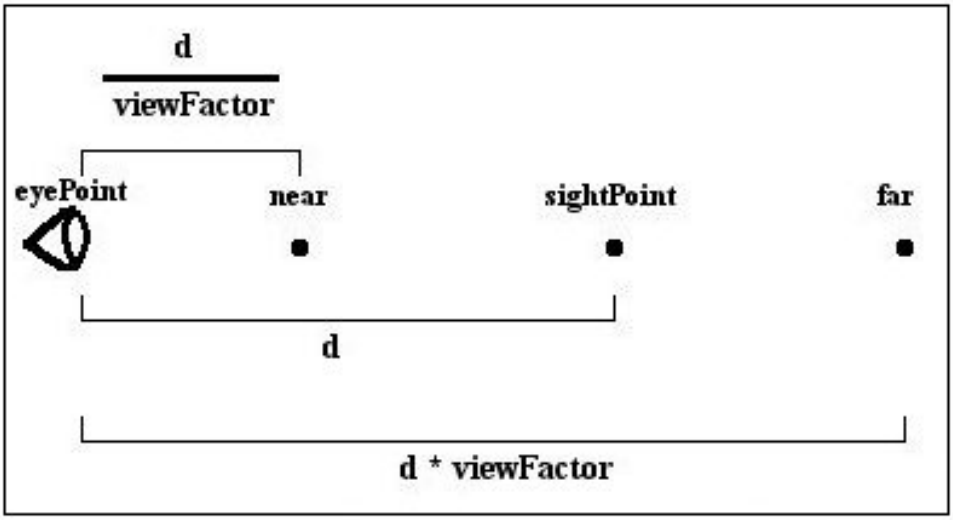


Figure 3.12: Camera main components ratio

and a far location which strech to the infinity.

3.6 GUI

This section describes the three GUI of the application. The main GUI, the configuration GUI and the GUI associated with the visualization model.
The GUI implementation is another important aspect of the application and deals with usability. We take care of this aspect building first a very basic version which we improve with the evolution of the application.

3.6.1 Main GUI

The main GUI is the starting point of the application. It has only few responsibilities, which are open the configuration GUI and start the visualization model (see Figure 3.13). We divide it in two parts: on the left a button opens the configuration GUI, while on the right another button opens the visualization model interface.

3.6.2 Configuration GUI

The configuration GUI is encapsulated in the main GUI, and its responsibilities are the IP address configuration and let the user choose the task to be performed by the node (see Figure 3.14).
For the moment we implement only these two basic responsibilitis, but for the future we introduce the possibility to store the last configuration or to create pre-defined configurations which can be loaded on occurrence.

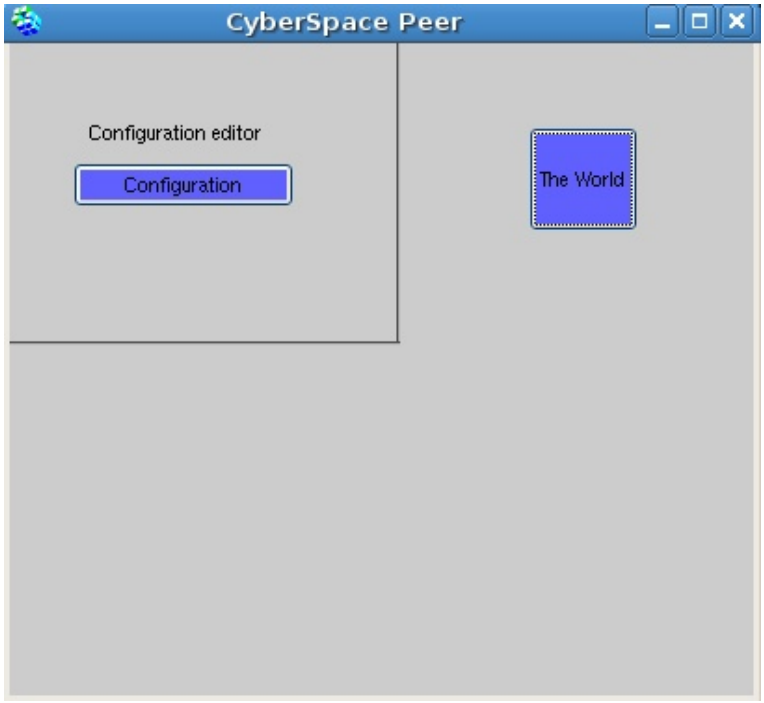


Figure 3.13: The main GUI

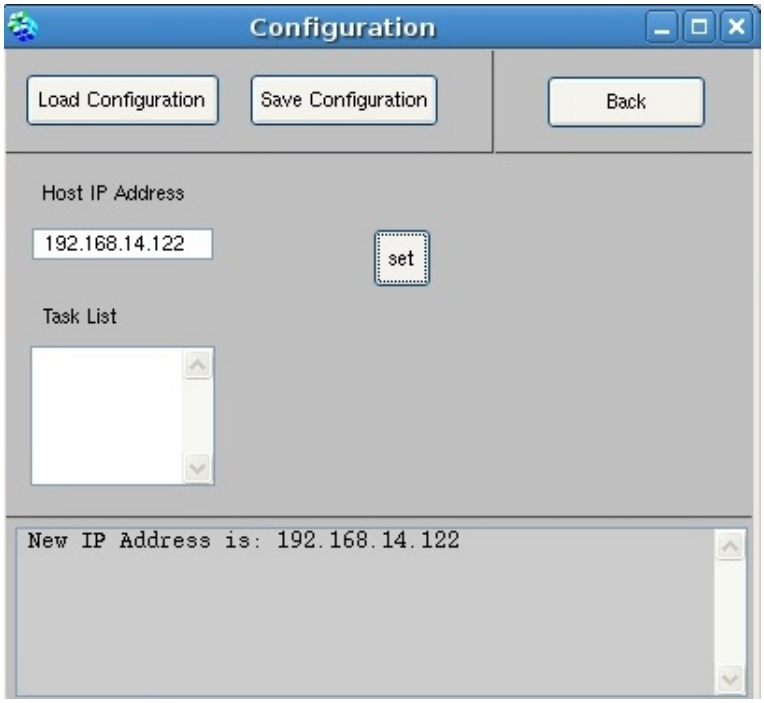


Figure 3.14: The configuration GUI

3.6.3 Visualization model interface

It is the working GUI of the application (see Figure 3.15). The central part encapsulates the view, it visualizes results as soon as new data arrives. The upper panel contains only a button to start the connection and the task to perform. The panel below allows to control and follow initial operations.
As mentioned above this interface is very basic and the reason is that we use it only for testing purposes and to have a feeling on the task performed by the application. Instead we put all the user functionality in the client GUI.

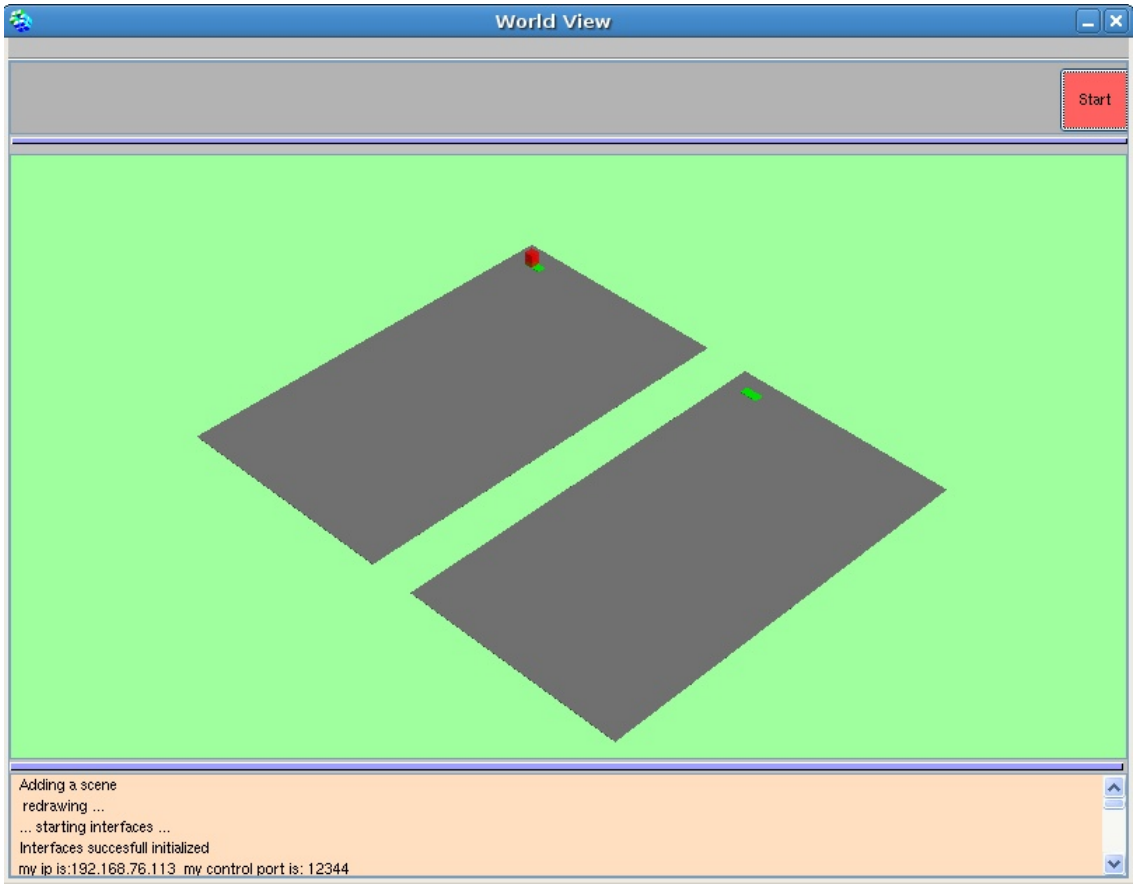


Figure 3.15: The model interface

Chapter 4

Conclusions

This chapter ends the feasibility study on the realization of the Cyberspace. We present a summary of the work and a discussion to highlight what has been done and what can be done in the future, illustrated with some scenarios.

4.1 Summary

The intention of this work was to analyze and implement a model to represent and visualize a network, in order to better understand the system. Our approach was to depict the two main aspects, data visualization and computer networking, and starting from there build an evolutionary prototype.
The model is composed of two main parts:

- *Computer Networking* let us build our task to identify which characteristics are more interesting to evaluate a network system. We started asking ourselves what is a network, concluding that it could be seen as a set of hosts with a communication layer. A host can be represented as the set of the offered services. Furthermore we decided to use the network to transport the acquired data, which was the graphical representation of services.
- *Data Visualization* is very useful in order to better understand data, and for this reason is used in many branches of science. Our approach was evolutionary, therefore we built the application starting from a simple 2D textual visualization, and ending with a 3D model. This path allowed us to make some assumption and determine pro and contra of each approach.

4.2 Discussion

We started this project from the origins of the Cyberspace, which gave us the original idea about a 3D representation of data (see Chapter 1).
We went then on the technical part (see Chapter 2). We abstracted a network in the Cyberspace and we defined the two main parts: computer networking and visualization. Furthermore we described some problems which gave us the motivation to start and continue the project, with a final introduction of related works.

The next step were the implementation and validation of the Cyberspace (see Chapter 3). We proposed two scenarios (see Section 3.3 and Section 3.4).
The first scenario (see Figure 3.3) were initially a port scanner with a textual visualization of the results. The textual visualization was not sufficient to satisfy our goals, in particular we needed a global point of view to analyze networks. For this reason we introduced 3D visualization and we moved from a textual port scanner to a 3D port scanner, which led to the first prototype. We validate the result with an experiment (see Figure 3.6 and Figure 3.7), we scan the subnet of the servers at USI university. Our experiment was successful, we discovered at least a security issue and we built a global point of view to analyze networks.
We moved then to the second scenario (see Figure 3.8). Our purpose was to divide the workload of the scanner among nodes and to share the results. We introduced Opentalk, a framework to build distributed applications. We described how Opentalk allows to send objects to a remote host and then we bagan some experiment to build the second prototype. We discovered an issue about the JunOpenGL representation of a string which prevented us to send them to remote. Moreover we faced to manage timeouts and blocks, first increasing the timeout value and then moving from synchronous to asynchronous in the domain of remote object passing. Furthermore we modeled a node and we defined a protocol to allow interaction between nodes.
We experimented with our prototype. We had a problem about data inconsistency between nodes: each node had the same data representation but in different order. We decided to introduce a World Positioning System to face to the problem, which completed our second scenario and achieved the goal to have a global point of view.
We built then a client to allow users to interact with the Cyberspace. We implemented a navigation system which allow us to start from a global point of view and go into detail.

After the implementation of our final prototype we analyzed the result. We took in consideration different properties which can affect the correct behavior of the prototype and we explained some design choice.

Robustness: Data visualization is critical, as we mentioned during the evaluation of the 3D port scanner. It is even more critical when we introduced the distributed system, because the same amount of data needed for a local visualization, should be sent to remote. This introduced the problem of timeouts, since a node employed more time to draw each object on the scene and to redraw the whole scene. With the increase of 3D objects we noticed an increase on the risk to encounter a timeout.
Another aspect to deal with in future work regards failures. Our application catches exceptions like timeouts, in order to let the application continue its task, but we require to manage other failures, like those which are related with nodes interaction.

Scalability: We started our implementation with a single node and then we introduced another node to develop the second scenario, but we never tested the prototype with more then two nodes. Our system is not scalable for two main reasons:

- *initial protocol:* we built and tested the protocol to work with two nodes only.

To allow scalability we should modify the protocol to work with many nodes.

- *robustness:* to build a scalable prototype we need to improve the robustness of the system, because is reasonable to think that a large Cyberspace, with many nodes, implies a great increase of 3D objects sent around with a considerable slow down of the performance and a risk of timeouts.

Security: This aspect is characterized by two different point of view:

1. The first one relates to the way a node exchange data with other nodes of the cyberspace. Data is not encrypted and therefore can be easily intercepted to allow a network mapping. This is not an easy task to solve because an encryption algorithm would slow down the process to send and receive data.
2. The second point of view comes from the results of the validation of the first prototype (see Figure 3.7): our application can eventually discover security problems inside a network, which is one of the goals in network analysis.

Fexibility VS Complexity: When we started implementing our application we decided to take into account flexibility.
In Chapter 1 we highlighted inside secondary goal the future evolution of our application: this goal is possible only with a flexible application which is easy to extend. To accomplish this task we decided to divide as much as possible the responsibilities of each class. The result is an application that allows us to reason about the future evolution and to watch in many directions.

Usability: We decide to put the usability of the system at different levels.
At the application level we put the system configuration, which at the moment have great limitations. There is only one task, but we preview to build an extension.
At client level we insert the navigation system. It allows the user to interact with data and in the future should be improved. The navigation system has some limitations: with JunOpenGL when a 3D object is too near the eye point of view, it disappears. A possible solution is a block at a given distance between the object and the eye point. This also is part of future work, as well as other interaction with data, as object selection with menu to analyze more information.

4.3 Future Work

We define three categories to enphasize different concepts on the development of future work:

- general improvement of the prototype
- evolution of the prototype
- application of the prototype to other domains

4.3.1 General improvement of the prototype

To apply our prototype over a large scale we need to improve robustness and scalability. The critical part is robustness. We need a mechanism that allow us to send 3D objects to remote without timeouts and in the fastest possible way. To accomplish these tasks

we should focus on the two frameworks, Opentalk and JunOpenGL. For each framework we should explore the library to reach a better understanding and discover the features they provide. Furthermore we test other softwares written with these frameworks and learn possible solutions which can help us to improve our prototype.

4.3.2 Evolution of the prototype

During our analysis and implementation of the project we focused on two scenarios which led to a first and final prototype. Of course there are many other possible scenarios and each one can be adapted to achieve a specific goal.
We propose one future scenario.

Specialized nodes

The goal of this scenario is to map a portion of the Internet and build a global view to allow us to extract information. We distinguish portion of the Internet by the addresses of the hosts. Moreover we highlight the services of each host, for example defining categories of services with different color gradient; the result would be a world with different colors depending on the service category, which make us able to build a different global point of view with new metrics.
To achieve the goal we specialize the nodes of the Cyberspace (see Figure 4.1).

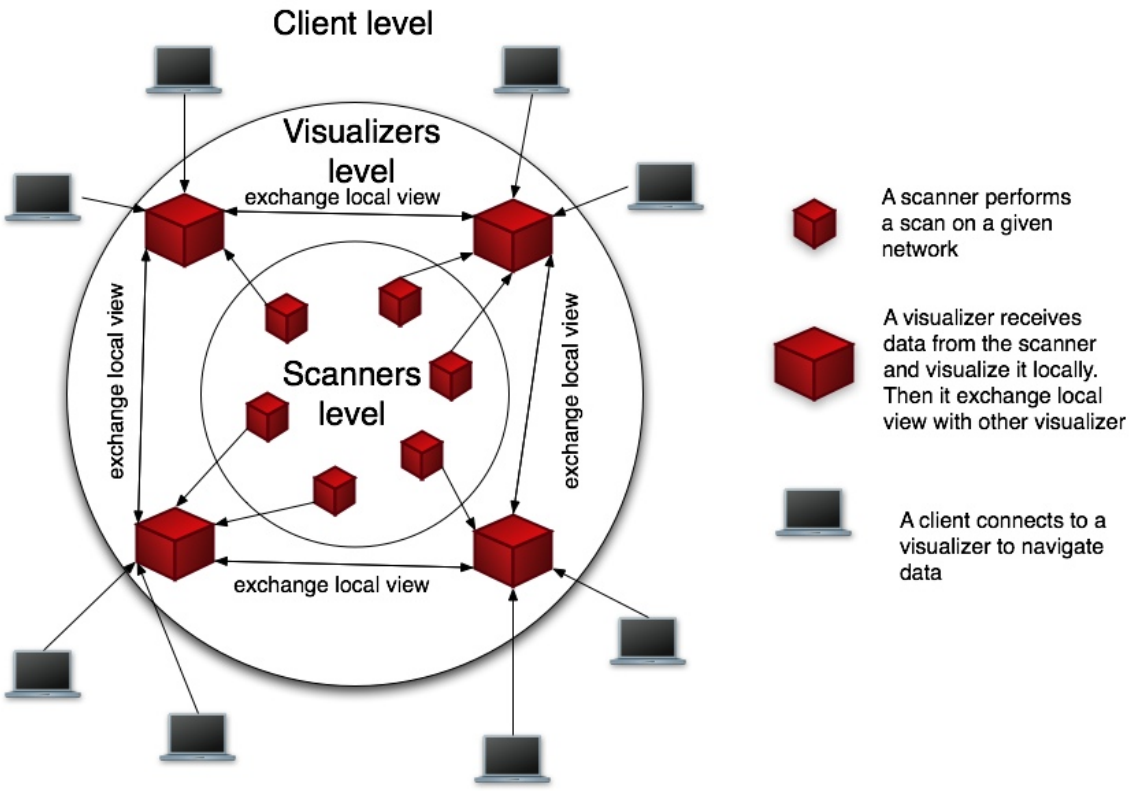


Figure 4.1: Specialized Nodes scenario

We use a set of hosts with high speed connection and great computational power,

like hosts of a university. We define each node of the Cyberspace to be either a scanner or a visualizer. We decide to apply such division because it reflects the two components of our project: networking and data visualization.
This configuration allow us to speed up the task, because each node has only one responsibility which can be optimized around the node itself.
We design a simpler configuration but we increase the complexity of the distributed system. We need a new protocol to handle hooks with different type of nodes and we should face to a better policy with failures.

4.3.3 Application of the prototype to other domains

Our work can be as well a good starting point for other applications which involve networking and 3D visualization.
We modify our prototype to join other domains. For example we use networking only for the distributed part of the Cyberspace, without been interested in it as data. We represent data as 3D objects and we could do that with all kind of data.
We propose two domain were is possible to apply our prototype:

- *games* where users interact remotely. We can build an environment composed by many nodes, where each node represents and visualize a place in an hypothetic world. Users connect to a starting node, a starting location in that world, and begin to interact with the environment, with the possibility to move to other locations.
- *applications* like CodeCity. With CodeCity it is possible to define data as software with its dependencies and represent it on the screen. It could be useful, when a designer analyze a software, to compare it with other similar softwares, to get at glance differences on design. We can build a Cyberspace around a group of research and let each developper share his software, or piece of software.

Appendix A

Implementation

This chapter provides some information on the implementation of this application. It is not intended to be exaustive, for a complete overview it is better to look at the code. However we give an idea of the implementation, of the classes involved and its hierarchy.
The first image is a UML class diagram of the main application.

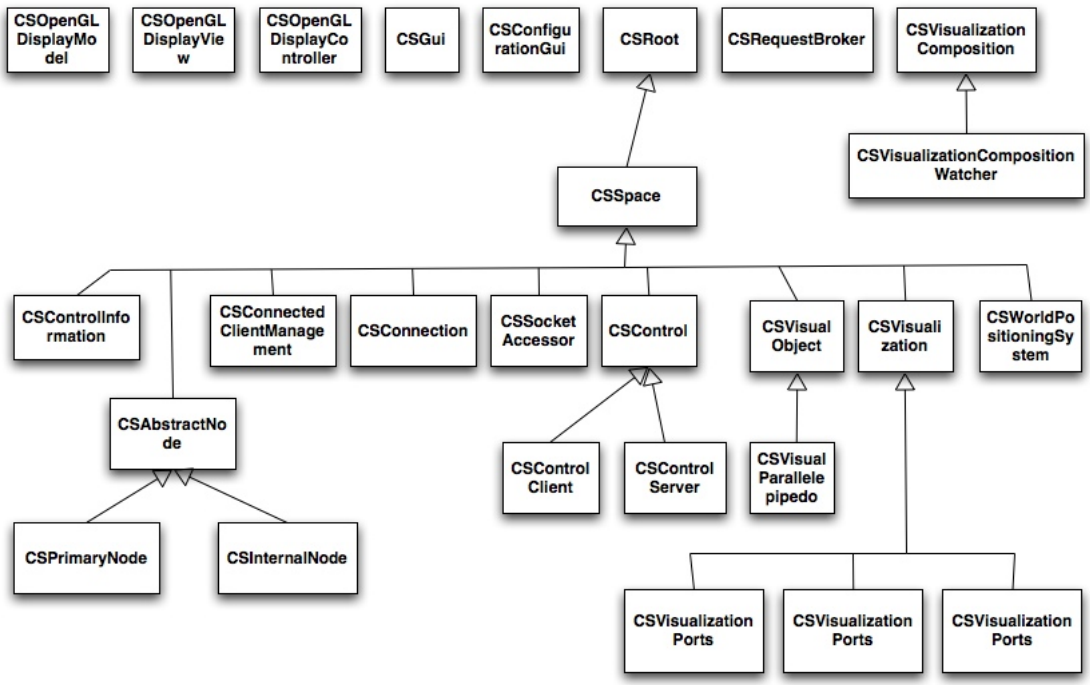


Figure A.1: The UML class diagram of the main application

As the diagram shows, class names are self explanatory. We design our system with a general CSRoot and CSSpace abstract class, and most of the other class as its subclasses. Classes that are not inside this hierarchy are bindings with other frameworks,

like JunOpenGL and Opentalk.

Another important aspect is the initialization of the system in order to better understand the behavior and where is possible to modify it for future evolution.

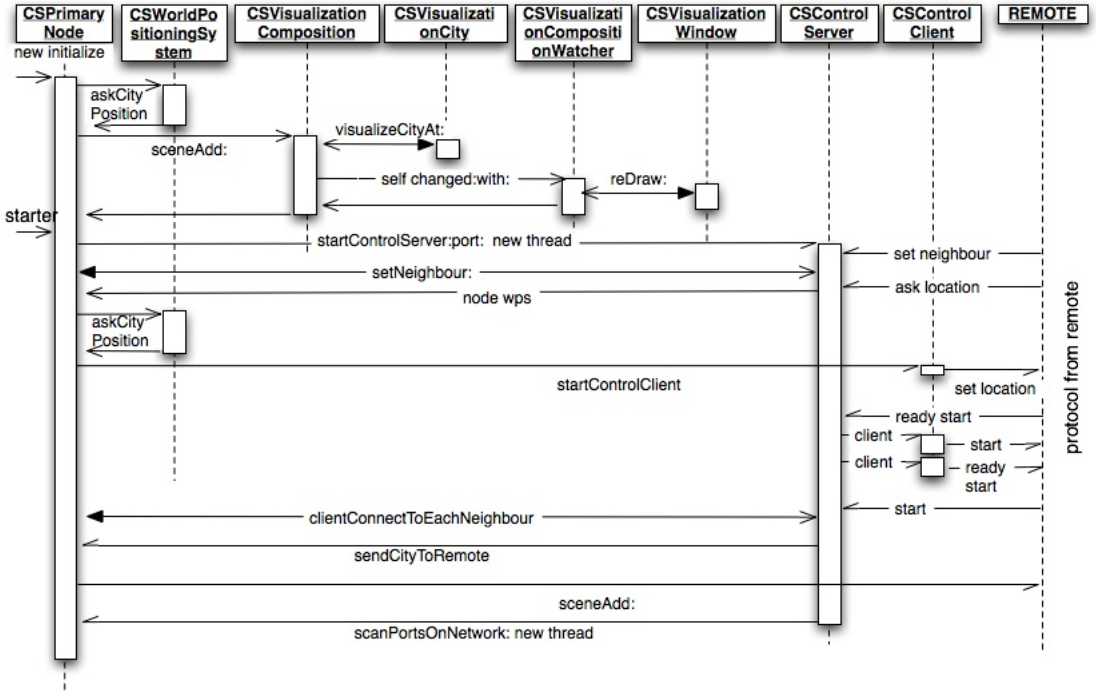


Figure A.2: The UML diagram depicts the initial sequence of the system

Figure A.2 describes the initialization process from the point of view of the first node. First part is managed by the node itself to start up, until the CSControlServer has been initialized. After this point nodes interact exchanging protocol messages: it is possible to see this process until the task to be performed starts.

List of Figures

Bibliography

[Gib84] William Gibson. *Neuromancer*. 1984.

[Ste92] Neal Stephenson. *Snow Crash*. 1992.

[WL07a] Richard Wettel and Michele Lanza. Program comprehension through software habitability. In *Proceedings of 15th International Conference on Program Comprehension (ICPC 2007)*. IEEE Computer Society, 2007.

[WL07b] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In *Proceedings of 4th IEEE International Workshop on Visualizing Software For Understanding and Analysis (VISSOFT 2007)*. IEEE Computer Society, 2007.