

Paper Programmed Project 1

Juan Navarro Camacho, Arturo Chinchilla, Malcolm Davis Steele, Ariel Piedra

September 2015

1 Abstract

Abstract: Distributed shared memory (DSM) is an architecture where the physical memory of each machine can be seen as one, at the time where a program tries to use the memory the systems allocate a chunk of memory in any of the machines which form the DSM and a single machine can address memory in any of the machines of the DSM. The main goal of this project is design and implement a distributed shared memory manager at user level, which means create a library that can be include to extend the available dynamic memory operations, this memory manager is going to implement new features like smart pointers and automatic free of memory (usually known as garbage collector), the system is going to allocate a lot of memory as the first

instruction to ensure that we are going to avoid the use of system calls and all memory allocations are going to be resolved at user level.

2 Design Problem

It requires a distributed memory administrator that allows us to abstract partially some dynamic memory concepts. Moreover it has the possibility that extends the computer memory dynamically adding more memory nodes using **TCP/IP** to share their main memory to some client app.

3 Proposed Solution

The following diagrams describes the Solution made by the project team.

3.1 Package Diagram

3.1.1 User View

This section is the User view, it means this is the description of all the modules to build the app client application. It will have **Data structures** made by the students, which will have the function of organize any dynamic data of the App Client. A **JSON/C++** module which has the function of establish the structure of the message to send and protocol of communication. An **XML-Parser** to set configurations options, like max length of the message, **IP** and **Ports Numbers** to communicate the app client with the **SDSMMemory Nodes**. The Test App Module is the application that will prove the functionality of the **dHeap**. The **dHeap** is the module that knows every variable made with the dynamic distributed memory and it will communicate the actions to the nodes using the Client/Server module sending the JSON-Formatted message.

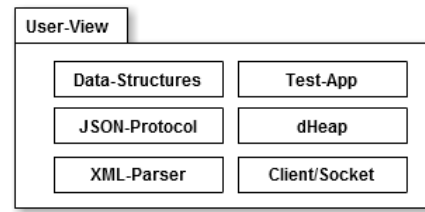


Figure 1: Server View Package

3.1.2 Server View

This package is the another side of the system, it has the administration of the memory, the nodes, and it will communicate with the **dHeap sending** information of the memory saved in this Node o managing data. It has a JSON protocol module, like the Client View, due to it needs to establish the structure of the message to send. An XML-Parser module is needed to set configurations options like Port number, IP address, Memory to share. Data Structures will have the function of organize the information of the memory pointers and the clients that has the data saved. Using some GUI/C++ tool for represent the Memory Monitor to make a easily way to view the interaction of the memory node with the clients. The Web Query is optional, to approach extra functionality using the web

browser.

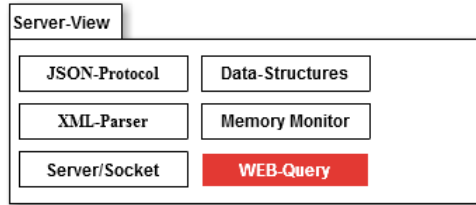


Figure 2: User View Package

3.2 Class Diagrams

This is the description of all the Diagrams created for the design solution applications.

3.2.1 XML_Handler Class

This class the following function: Load configurations to start the application, in this case SDSMemoryNode, ServerSocket and ClientSocket require to establish some information given by XML_Configuration Files. Thus the maximum memory that a Node can share or the port numbers for the socket to establish the communication are loaded using this module, developed using **OPP and C++**. Using the public methods

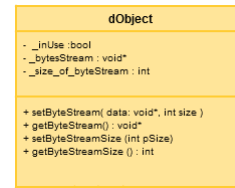


Figure 3: Client Node Class Diagram

3.2.2 dHeap Class

This class handles the methods to make requests about memory administration from the Client APPs. That is to say tell to the SDSMemoryNode for memory request. This Heap must be created with singleton pattern to ensure that there is only application running in every Client app and avoid runtime errors with the threads besides **mutex** is using as execution threading protection.

3.2.3 dGarbageCollector Class

This module, has the function to travel across the Data Structure the has all the instances of dObject in every Client App and check if each dObject instance is being used, on the other hand if not, then the garbage Collector will tell to the heap and this will send a message to the SDSMemoryNode to free that unused memory.

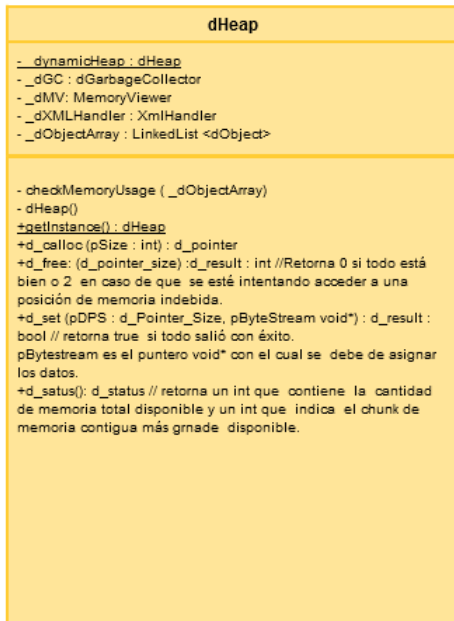


Figure 4: dHeap Class Diagram

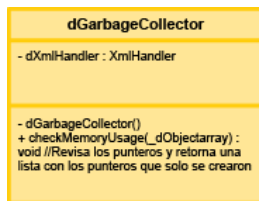


Figure 5: dGarbageCollector Class Diagram

3.2.4 dObject Class

The dObject class, as its name says, is an abstraction of the Dynamic Memory Object that encapsulates all the overloaded operators functions to create a similar behaviour of the pointers management in C++. What it really happens is that those overloaded operators make messages to the heap for

memory communication and administration with the data saved on the nodes. A void* pointer is needed to save temporally and locally the data before use the dHeap to save it.

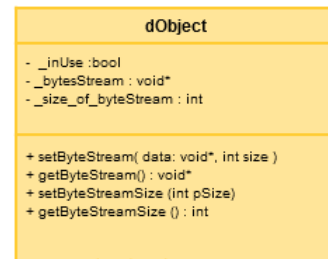


Figure 6: dObject Class Diagram

3.2.5 ClientSocket Class

This is the Client side of the communication system, the ClientSocket Class has the responsibility to send every message for memory administration to the SDSMMemory Nodes associated. The method used to approach this target is **sendMessage** returning and string as a result message from the server telling the status of the action.

3.2.6 ServerSocket Class

The ServerSocketClass is the Server side of the communication system between the app clients and the SDSMMemory Node. To es-

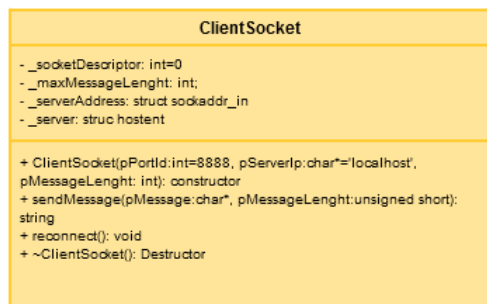


Figure 7: ClientSocket Class Diagram

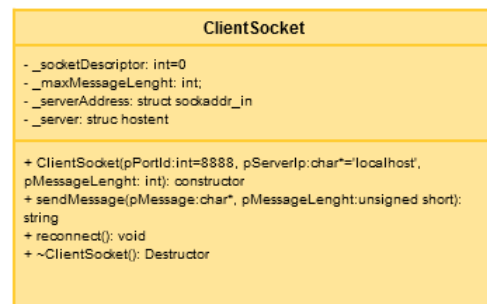


Figure 8: ServerSocket Class Diagram

establish the communication with the clients the SDSMMemoryNode uses the following methods among other ones.

acceptConnections Accepts the Clients connections to make the communication with the applications clients .

closeConnection Close the connection in some specific client.

getFromSocket This method returns the information with string format from the sockets.

sendThroughSocket This method is used to send messages in every client.

the administration of the ClientSockets the Client-List mainly uses the following methods:

SearchClient: Search inside the data structure for some client that has same ID.

InsertClient: This method adds a client node in the data structure.

DeleteClient: this method deletes the Client Socket, if exist, of the Client List.

GetSize: This method is used to know how mouch elements has the client list.

3.2.7 ClientList CClass

This is the data structure to save all the references of client sockets. Every Client Socket is saved in Client Node. To control

3.2.8 ClientNode Class

This is the abstraction of a Client Socket container. Every container is managed by the Client List. we only use the client socket

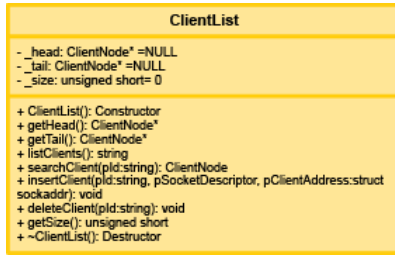


Figure 9: Client List Class Diagram

for communications, but a client node is used to add another functionality to the client socket if we want to implement some kind of data sorting.

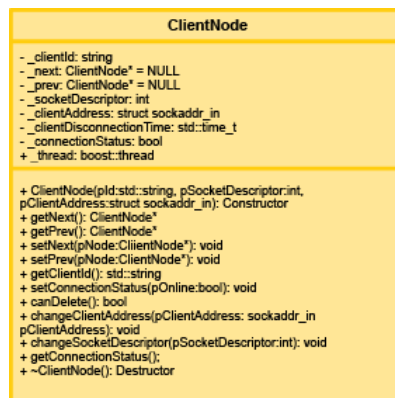


Figure 10: Client Node Class Diagram

3.2.9 SDSMMemoryNode CClass

This Class was created to implements the real heap memory usage on the computers that has function to be a Dynamic Memory Node. with the following methods we can use the heap of every computer running the SDSMMemory App.

- **memoryAvailableRequest:** This method return an integer that represents the maximum available memory that the SDSMMemory Node can share as an entire chunk.
- **memoryRequest:** This method return a dPointer with some unfilled fields when a client make a request of some amount of bytes. The completed dPointer will send to the dHeap.
- **freeMemory.** this is method the the client can use indirectly when it doesn't need the memory given by the heap, this method returns and state-ment tellign the result of the operation.
- **getMemory.** This method returns a transformation of the data
- **setSetMemory.** This methods update the information saved on the SDSM-Memory node, to the arguments given by the call. This method returns the status of the transaction.
- **searchMaxChunk.** This method looks for the maximum memory chunk available to share.

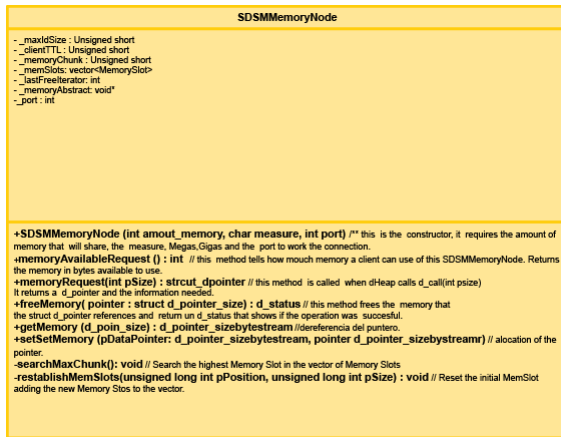


Figure 11: SDSMemoryNode Class Diagram

4 C++ Structures designed

4.1 d_pointer

This structure is to encapsulates the IPv4 of the computer that has saved the data. The size is an unsigned long int that stores the size in bytes of the data. There is another value saved in this structure call status, this value is a Boolean that show us if the memory is available to use.

4.2 d_status

This is a struct that has information about the transaction of any operation made by the dHeap.

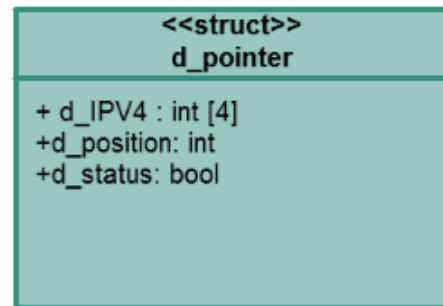


Figure 12: d_pointer Structure Diagram

4.3 d_pointer_size

This structure is used when we want to tell to the dHeap that is time to erase or free the memory saved in the SDSMMemoryNode. This structure is composed by a d_pointer struct and a unsigned long int that means the size of the data that we want to delete.

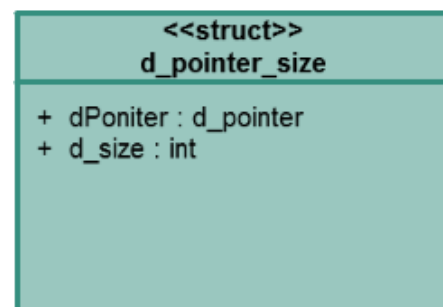


Figure 13: d_pointer_size Structure Diagram

4.4 d_pointer_sizebyStream

This structure is used to call d_set method from the dHeap, it requires a dpointer that has knowledge about the dynamic memory node and the data that will be set and a unsigned long int that has the size of the bytes to be set.

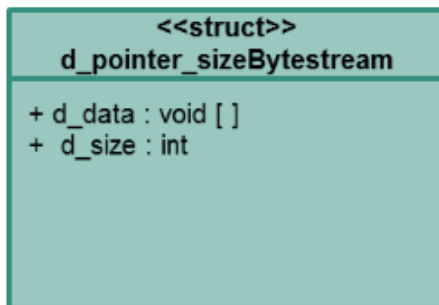


Figure 14: d_pointer_sizebyStream Structure Diagram

4.5 d_MemorySlot

4.6 JSON File

This is the protocol made by the team for establish the messages to be used for the communication of the SSDMMemoryNode and the dHeap. The following image shows the basic structure of the message: A variable called **message** that stores the type of the message. The **argc** is the argument

counter, that stores the number of arguments given. The **argv** is the data by itself to be used as an argument.

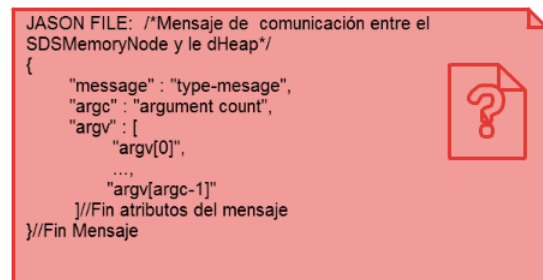


Figure 15: JSON Template Diagram

4.6.1 connect

This is a message to establish the first connection of the SDSMemoryNode. The variable **argv** stores the identification of the client.

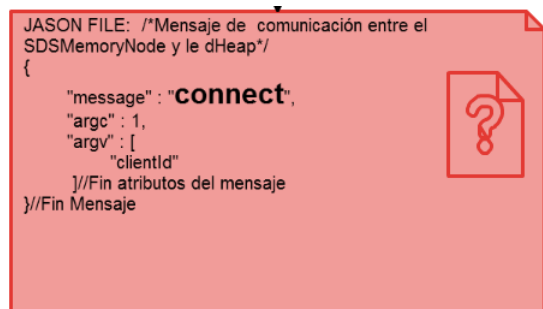
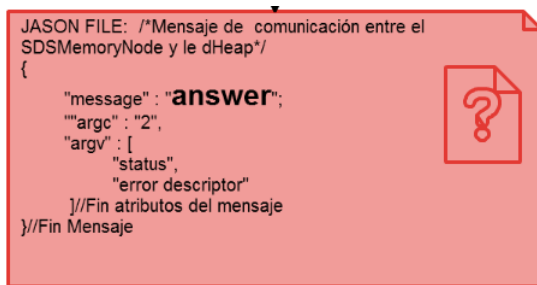


Figure 16: JSON Connect Diagram

4.6.2 connect answer

This answer is given by the SDSMemoryNode that set or not the connection of the

client and the server. The **argc** has a value of two, to indicate the **argv** has two values, the status of the connection and the message of the transaction.



```

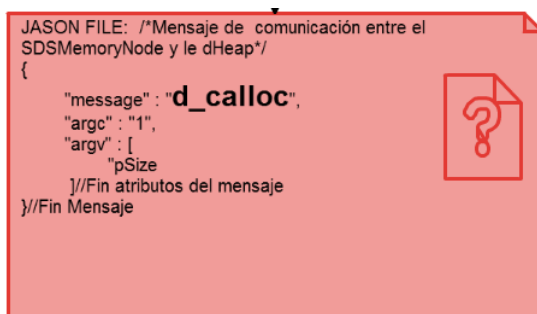
JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_malloc";
  "argc": "2",
  "argv": [
    "status",
    "error descriptor"
  ]//Fin atributos del mensaje
}//Fin Mensaje

```

Figure 17: JSON Connect Answer Diagram

4.6.3 d_malloc

This message has **argc** equal to 1 to indicate that the **argv** has just one value, and this value is the size of the memory that client application is requesting.



```

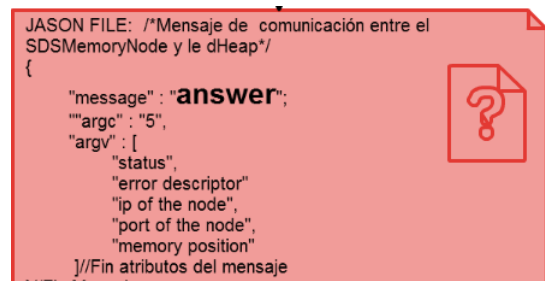
JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_malloc",
  "argc": "1",
  "argv": [
    "pSize"
  ]//Fin atributos del mensaje
}//Fin Mensaje

```

Figure 18: JSON Connect Answer Diagram

4.6.4 d_calloc answer

The answer is the **argc** equal to 5 and the **argv** has the following information: The **status** of the operation, the **Operation error** to show us if something bad just happened. The other information is not null when the operation was successful, the **IP** of the dynamic node which saves the memory. The last one is the real position of the pointer saved in the memory node.



```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_calloc";
  "argc": "5",
  "argv": [
    "status",
    "error descriptor",
    "ip of the node",
    "port of the node",
    "memory position"
  ]//Fin atributos del mensaje
}//Fin Mensaje

```

Figure 19: JSON Connect Answer Diagram

4.6.5 d_free

This message tells to the dHeap that the related dPointer is not needed any more. The 4 parameters given by JSON message are: The IPv4 to the related SDSMemoryNode. The Position of real pointer, the status of the data, and the size in bytes of the data to delete on the SDSMMemory Node.

```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_free",
  "argc": 4,
  "argv": [
    "ip",
    "position",
    "status",
    "size"
  ]
}
//Fin atributos del mensaje
}
//Fin Mensaje

```

Figure 20: JSON Free Diagram

4.6.6 d_free answer

The result of this operation consist in tow values, one of the is the status of the operation and the second one is the message describing what just happened after the memory deallocation.

```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "answer";
  "argc": "2",
  "argv": [
    "status",
    "error descriptor"
  ]
}
//Fin atributos del mensaje
}
//Fin Mensaje

```

Figure 21: JSON Free Answer Diagram

4.6.7 d_set

This message tells to the SDSMMemoryNode to set the information saved, it uses d_pointer to refer the SDSMMemoryNode and the stream of data to be loaded.

```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_set",
  "argc": "5",
  "argv": [
    "ip",
    "position",
    "status",
    "size",
    "Luyesleerrrr"
  ]
}
//Fin atributos del mensaje
}

```

Figure 22: JSON Set Diagram

4.6.8 d_set answer

This answer show us the status of the operation.

```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "answer";
  "argc": "2",
  "argv": [
    "status",
    "error descriptor"
  ]
}
//Fin atributos del mensaje
}
//Fin Mensaje

```

Figure 23: JSON Set Answer Diagram

4.6.9 d_get

This message gets the data inside the d_pointer inside the SDSMMemoryNode

```

JSON FILE: /*Mensaje de comunicación entre el
SDSMemoryNode y le dHeap*/
{
  "message": "d_get",
  "argc": 4,
  "argv": [
    "ip",
    "position",
    "status",
    "size"
  ]
}
//Fin atributos del mensaje
}
//Fin Mensaje

```

Figure 24: JSON Get Diagram

4.6.10 d_get answer

The answer is the `char*` version of the data that needs to be casted to the real value.

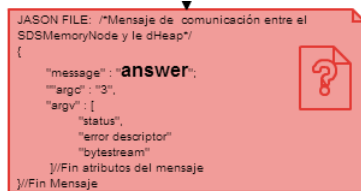


Figure 25: JSON Get Answer Diagram

5 Programing Tools Description

5.1 Netbeans

NetBeans is an open-source software development project with an active community of collaborating users and developers.

Though the latest release of NetBeans IDE focuses very heavily on dynamic languages, and PHP in particular, there are many new and enhanced features that should appeal very specifically to Java programmers.

- Performance Improvements: Performance covers many areas, from start up time, to editor responsiveness (i.e.,

if you need to wait for the code completion box to appear, you have a performance problem), to the speed at which dialogs such as the Options window appear. Also, as applications increase in size, to a point where they become extremely large with many thousands of files, there is an inevitable performance impact: the trick is to keep this impact as limited as possible.

- Compile & Deploy on Save: It's one of the key features that has distinguished. When we compile and deploy our projects automatically Netbeans save our changes.
- Java Editor Enhancements: An area where NetBeans IDE has consistently improved, while continuing to lag over its peers, is the Java editor, in terms of its performance (see above) and, in particular, its range of refactorings.
 - Code Completion: The jewel of any code editor is, I believe, together with the "hints/tips" feature, the code completion feature.
 - Creating Getter/Setter from Ex-

isting Variable When you want to refactor a variable such that it acquires a getter/setter combination

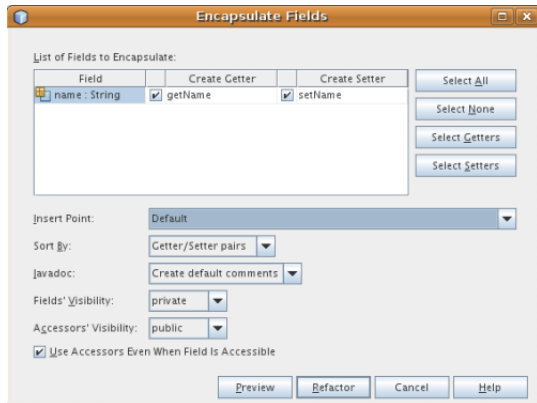


Figure 26: Getters & Setters in NetBeans

5.2 QtCreator:

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which is part of the SDK for the Qt GUI Application development framework. It includes a visual debugger and an integrated GUI layout and forms designer. The editor's features include syntax highlighting and autocompletion, but purposely[4] not tabs (although plug-ins are available). Qt Creator uses the C++ compiler from the GNU Compiler Collection on Linux and FreeBSD. On Windows it can use

MinGW or MSVC with the default install and can also use Microsoft Console Debugger when compiled from source code.

5.3 Git:

Is a widely used version control system for software development. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed by Linus Torvalds for Linux kernel development in 2005. Characteristics:

- **Branching and Merging:** Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.
- **Small and Fast:** With Git, nearly all operations are performed locally, giving it a huge speed advantage on centralized systems that constantly have to communicate with a server somewhere.

- **Distributed:** One of the nicest features of any Distributed SCM, Git included, is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.
- **Data Assurance:** The data model that Git uses ensures the cryptographic integrity of every bit of your project. Every file and commit is checksummed and retrieved by its checksum when checked back out. It's impossible to get anything out of Git other than the exact bits you put in.
- **Staging Area:** Unlike the other systems, Git has something called the "staging area" or "index". This is an intermediate area where commits can be formatted and reviewed before completing the commit.
- **Free and Open Source:** Git is released under the GNU General Public License version 2.0, which is an open source license. The Git project chose to use GPLv2 to guarantee your freedom to share and change free software—to

make sure the software is free for all its users.

5.4 Bit Bucket

Distributed version control system that makes it easy for you to collaborate with your team. The only collaborative Git solution that massively scales. Characteristics:

- **Pull requests:** Pull requests facilitate code reviews, resulting in higher quality code and an opportunity to share knowledge amongst your team.
- **Branch permissions:** Branch permissions provide granular access control for your team, ensuring the right people can make the right changes to your code.
- **Inline discussions:** Collaborate right in your code with inline comments, threaded conversations, and mentions.

5.5 Cacao:

Cacao is a diagram creation tool that runs in your web browser. Multiple people can work together on the same diagram in real

time. Diagrams can be published directly to websites, wikis, and blogs. Characteristics:

- Elements can be dragged and drop to easily create diagrams.
- You can invite collaborators to work with you in Cacao.
- Multiple people can edit a diagram in real time.
- There is a chat function in the editor so people can communicate while creating diagrams.
- When editing with multiple people, users icons appear on selected objects.

5.6 Evernote:

Evernote is an independent, privately held company headquartered in Redwood City, California. Founded in 2007. As one workspace that lives across your phone, tablet, and computer, Evernote is the place you write free from distraction, collect information, find what you need, and present your ideas to the world.

5.7 Developing tools:

5.7.1 RapidJSON:

RapidJSON is a JSON parser and generator for C++. It was inspired by RapidXml.

- Is small but complete. It supports both SAX and DOM style API. The SAX parser is only a half thousand lines of code.
- Is fast. Its performance can be comparable to `strlen()`. It also optionally supports SSE2/SSE4.2 for acceleration.
- Is self-contained. It does not depend on external libraries such as BOOST. It even does not depend on STL.
- Is memory friendly. Each JSON value occupies exactly 16/20 bytes for most 32/64-bit machines (excluding text string). By default it uses a fast memory allocator, and the parser allocates memory compactly during parsing.
- Is Unicode friendly. It supports UTF-8, UTF-16, UTF-32 (LE & BE), and their detection, validation and

transcoding internally. For example, you can read a UTF-8 file and let RapidJSON transcode the JSON strings into UTF-16 in the DOM. It also supports surrogates and "u0000" (null character).

5.7.2 Boost: (Thread)

Enables the use of multiple threads of execution with shared data in portable C++ code. It provides classes and functions for managing the threads themselves, along with others for synchronizing data between the threads or providing separate copies of data specific to individual threads.

5.7.3 PugiXML:

Pugixml is just another XML parser. This is a successor to pugxml (well, to be honest, the only part that is left as is is wildcard matching code; the rest was either heavily refactored or rewritten from scratch). The main features are:

- Low memory consumption and fragmentation (the win over pugxml is 1.3 times, TinyXML - 2.5 times, Xerces (DOM) - 4.3 times). Exact numbers

can be seen in Comparison with existing parsers section.

- Extremely high parsing speed (the win over pugxml is 6 times, TinyXML - 10 times, Xerces-DOM - 17.6 times)
- Extremely high parsing speed (well, I'm repeating myself, but it's so fast, that it outperforms Expat by 2.8 times on test XML)
- More or less standard-conformant (it will parse any standard-compliant file correctly, with the exception of DTD related issues)
- Pretty much error-ignorant (it will not choke on something like `<text>You & Me</text>`, like expat will; it will parse files with data in wrong encoding; and so on)
- Clean interface (a heavily refactored pugxml's one)
- More or less Unicode-aware (actually, it assumes UTF-8 encoding of the input data, though it will readily work with ANSI - no UTF-16 for now (see Future work), with helper

conversion functions (UTF-8 to UTF-16/32 (whatever is the default for `std::wstring & wchar_t`))

- Fully standard compliant C++ code (approved by Comeau strict mode); the library is multiplatform (see reference for platforms list)
- High flexibility. You can control many aspects of file parsing and DOM tree building via parsing options.

6 Task's Distribution

6.0.4 Malcolm's Tasks

Design the Communication between the socket and the client. Is needed that the connection will be thread safe and has the function to use JSON as protocol for communication.

6.0.5 Arturo's Tasks

Develop the XML parser and make the SDSMMemory Node with Juan.

6.0.6 Juan's Tasks

Create the SDSMMemory Node and make the examples for Malcolm to use the JSON Parser.

6.0.7 Ariel's Tasks

Create the functionality of the dObject class and its basic data types.

7 Programming Used Concepts

7.1 Padding Bytes:

Data structure alignment is the way data is arranged and accessed in computer memory. It consists of two separate but related issues: data alignment and data structure padding. When a modern computer reads from or writes to a memory address, it will do this in word sized chunks (e.g. 4 byte chunks on a 32-bit system) or larger. Data alignment means putting the data at a memory address equal to some multiple of the word size, which increases the system's performance due to the way the CPU handles memory. To align the data, it may be

necessary to insert some meaningless bytes between the end of the last data structure and the start of the next, which is data structure padding.

7.2 Composition Objects:

In computer science, object composition (not to be confused with function composition) is a way to combine simple objects or data types into more complex ones. Compositions are a critical building block of many basic data structures, including the tagged union, the linked list, and the binary tree, as well as the object used in object-oriented programming.

7.3 Parallel Computing:

Parallel computing is a form/type of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved at the same time. There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance

computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.

7.4 TCP-IP protocol communication

TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. When you are set up with direct access to the Internet, your computer is provided with a copy of the TCP/IP program just as every other computer that you may send messages to or get information from also has a copy of TCP/IP. Use the client/server model of communication.

7.5 Objects Hierarchy

Inheritance is when an object or class is based on another object or class, using the

same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior. It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces.

7.6 Operators overloading

C Plus plus allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

7.7 JSON protocol messaging

Lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language

7.8 structural Programming

Structured programming techniques use functions or subroutines to organize the programming code. The programming purpose is broken into smaller pieces and organized together using function. This technique provides cleaner code and simplifies maintaining the program

8 Algorithms Analysis

The following Images show some examples of the Analysis of Algorithms:

8.1 Memory Request

8.2 Insert Client

8.3 Send Message

8.4 Connection Handler

8.5 Get p List

9 Testing

The Following example is a basic Test to ensure the functionality of the SDSMMemoryNode.

```

struct d_pointer SOSMemoryNode::memoryRequest(unsigned long int psize){
    d_pointer* result;//this is the result. O(1)

    unsigned long int position_of_chunk = 0; O(2)

    //Verify if psize (bytes) is greater, smaller than the maximum chunk
    //Of memory.
    if ( psize <= this->max_chunk->size) { O(1)

        /**start search the array of memory slots to find the accurate memory
        * slot.
        */
        //travel inside the vector
        for ( unsigned long int i=0; i< this->memslots.capacity(); i++){ O(n)
            //if the slot is free to use , we have to check if this slot
            //has the memory greater than or equal to psize;
            if ( this->memslots[i]._freespace == true ) { O(1)
                //checking the size

                if (this->memslots[i]._size >= psize) { O(1)
                    //show a message.
                    std::cout << "A position of free memory was found...Max "
                        "size of chunk: " << this->memslots[i]._size << std::endl;

                    //save this position to use when this method call.
                    //reestablish.
                    std::cout << "The chunk was found on the position: " << i
                        << " on the vector of memory slots" << std::endl;
                    position_of_chunk = i; O(1)
                    //the temporal memory slot
                    d_pointer newDPointer; O(1)
                    result = newDPointer; O(1)
                    result->d_position = this->memslots[i]._position; O(1)

                }

            }
        }
    }

    //ending of for
}

//ending of if
//Return a Null d_pointer in case that the if statement is false.
if ( result == 0 ) { O(1)
    std::cout << "Error: The size that you are querying is greater than "
        << "the maximum memory chunk of this node..." << std::endl;
    //indicating that the pointer cannot be used.
    result->d_status = false; O(1)
}

std::cout << " Restore the vector of memory node..." << std::endl;
//trunk the memory slot
this->reestablishMemslots(position_of_chunk,psize); O(1)
//search the next max chunk;
this->searchMaxchunk(); O(1)
//print the memory
this->printMemslot(); O(1)
return *result; O(1)
}

```

Figure 27: Memory Request Analysis

```

void ClientList::insertClient(string pid, int psocketDescriptor, struct sockaddr_in pclientAddress, int
try{
    ClientNode* tmpClient =this->searchClient(pid); O(2)
    if(tmpClient==NULL){ O(1)
        if(pid.size()==10){ O(1)
            ClientNode* pNode=new ClientNode(pid,psocketDescriptor, pclientAddress, psocketDir); C
            if(this->size==0){ O(1)
                this->_head=pNode; O(1)
                this->_tail=pNode; O(1)
            }else{
                this->_tail->setNext(pNode); O(1)
                pNode->setPrev(this->_tail); O(1)
                this->_tail=pNode; O(1)
            }
            this->size++; O(1)
        }else{
            throw -6; O(1)
        }
    }else{
        if(tmpClient->getConnectionStatus()){ O(1)
            throw -5;
        }
    }
}

catch(int e){ O(1)
    if(e==5){ O(1)
        std::cout<<"The Client already exists."<<std::endl;
        throw -5; O(1)
    }
    if(e==6){ O(1)
        std::cout<<"The lenght of the client ID have to be 10."<<std::endl;
        throw -6; O(1)
    }
    else{
        return; O(1)
    }
}
return; O(1)
}

```

Figure 28: Insert client Analysis

10 Design Problems

10.1 Fragmentation

There is no implementation to saved free₁₉ space andreallocate the free memory to

```

string ClientSocket::sendMessage(char* pmessage, unsigned short pmessageLength) {
    if(pmessageLength<=this->_maxMessageLength){ O(1)
        int n; O(1)
        char buffer[this->_maxMessageLength]; O(2)
        n = write(this->_socketDescriptor, pmessage, pmessageLength); O(2)
        if (n < 0) { O(1)
            printf("ERROR writing to socket");
        }
        bzero(buffer,this->_maxMessageLength); O(1)
        n = read(this->_socketDescriptor, buffer, this->_maxMessageLength); O(2)
        if (n < 0){ O(1)
            printf("ERROR reading from socket");
        }
        string result(buffer); O(1)
        return result; O(1)
    }
    else{
        throw -10;//The message is too big. O(1)
    }
}

```

Figure 29: Send Message Analysis

```

void* ServerSocket::connectionHandler(void * psocketDescriptor, struct SocketData * psocketData, string pclientID){
    std::cout << "Se Conecto el cliente: " << pclientID <<std::endl;
    //LogicGame* gameTmp=NULL;
    int socket=(int*)psocketDescriptor; O(2)
    string request; O(1)
    ClientNode* tmpClient=psocketData->_clientList.searchClient(pclientID); O(1)
    bool changesocket=false; O(2)
    while(!tmpClient->canDelete()){ O(n)
        //If client is connected
        if(tmpClient->getConnectionStatus()){ O(1)
            if(changesocket){ O(1)
                socket=tmpClient->getSocketDir(); O(2)
                changesocket=false; O(2)
            }
            request=getFromSocket(socket); O(1)
            cout << request << endl;
            if(request.compare("disconnected")!=0){ O(1)
                string answer=SOSWManager(request); O(1)
                this->sendThroughSocket(socket, answer); O(1)
            }
        }
        else{
            tmpClient->setConnectionStatus(false); O(1)
            changesocket=true; O(1)
        }
    }
    }else{
        usleep(1000000); O(1)
    }
}

psocketData->_clientList.deleteClient(pclientID); O(1)
return 0; O(1)
}

```

Figure 30: Connection Handler Analysis

```

vector<string> XmlHandler::getIplist(){
    pugi::xml_node node = _DpSettings.child("Settings").child("ipclients"); O(2)
    string node_name = node.name(); O(2)
    if (node_name == "ipclients"){ O(1)
        for (pugi::xml_node child_node = node.first_child(); child_node; child_node= child_node.next_sibling()) O(n)
            string stringIp = child_node.child_value(); O(2)
            this->size++; O(1)
            this->_ipvector.resize(_size); O(2)
            this->_ipvector[_size-1] = stringIp+" "+stringPort; O(2)
        }
    }
    return _ipvector; O(1)
}

```

Figure 31: Get p list Analysis

make new memory chunks,

10.2 Ideal Memory Chunk

The search of the memory chunk is based if the size is equal or less than the size of the free chunk. If that is true then the iteration that looks across the vector will be broken.

10.3 Char Data Casting

Data Casting is useful with the void* due to there are some problems when is need to send a void* using tcp/IP protocol communication.

10.4 Int to unsigned long int

The maximum number that an int can save is less than a long, so we decided to use it to manage big amount of bytes.

10.5 Uninitialized pointers

A wrong performance is presented in the code, giving unexpected and unclear results, as a solution Debugger NetBeans is used to monitor step by step variables and decision-making code, until you notice that a pointer I was taking a rather different to what should value, then initializing the

pointer is sought and that had not been initialized to nULL, this is changed to solve the problem.

10.6 Loop stop condition

When several Memory Slot are created because of memory requests by a user, a vector with each of these Memory Slot and free memory space is created; to remove any Memory Slot the Vector Memory (release that memory) if then enters a smaller amount of memory should be assigned to this new space, most however, was assigned to the last free memory space to find, using the NetBeans debugger and reviewing the method of creating the memory Slot note that in the loop through the vector of memory Slots needed a stop when you find the first empty space that meets the requested amount of memory.

10.7 Memory not released

Whenever testing memory allocation is performed, it was not released, and although tests were made with small amounts, this memory was being occupied, it is taken as

a measure using a small piece of additional code: the "delete ()".

10.8 Not significant variables names

While coding, some methods gave erroneous results, using the NetBeans Debugger is that they were using incorrect names of variables to what we wanted to do, the problem was because some variables were not significant and very similar names together. The more distinctive names are changed.

10.9 Optimization of structures

Optimization problems were found with the different structures that were implemented to keep track of areas in memory, such as the complexity of maintaining a structure for free slots and one for busy when free space or redundancy by having to go around the table from the beginning to find an empty space whenever a `d_calloc` done if one vector is implemented. I was management with a vector, to avoid the complex-

ity of free memory (that by joining contiguous free space), and to solve the problem of travel lastFree vector iterator indicating where you should look is maintained.

11 Results Achieved

We cannot make all the modules, for example the MemoryViewer. We now can make communications using server socket with a protocol. Usage of GIT can now give us a powerful tool to develop software.

12 Recommendations

- Use Git rather than SVN, because is more easier to use.
- Absolutely recommended initialize all pointers NULL to avoid memory problems Dirty and our programs have strange behaviors.
- The NetBeans Debugger is useful to check step by step what is happening internally with our program.
- It is vital to realize our proper memory management teams to achieve this

recommendation is to manually release the memory that are no longer used.

- We recommend using meaningful names to easily distinguish them from other variables, in addition to proper code documentation indicating their use.

13 Conclusions

- Git is better than SVN.
- Netbeans is by default the application to develop C++ programmens.
- Evernote proved to be a very useful tool for taking notes and logs, their ability to share the notes with the working group.

14 References

- About Evernote — Evernote (s. f.). Recuperado de <https://evernote.com/corp/>
- About - Git (s. f.). Recuperado de <https://git-scm.com/about>
- Bitbucket ? Features (s. f.). Recuperado de <https://bitbucket.org/product/features>
- Bitbucket ? The Git solution for professional teams (s. f.). Recuperado de <https://bitbucket.org/>
- Cacao - Create diagrams online Real time collaboration - About Cacao (s. f.). Recuperado de https://cacao.com/lang/en/about_cacao
- Cacao - Create diagrams online Real time collaboration - Features (s. f.). Recuperado de <https://cacao.com/lang/en/features>
- Chapter 30. Thread 4.5.0 - 1.59.0 (s. f.). Recuperado de http://www.boost.org/doc/libs/1_59_0/doc/html/th
- Git (software) - Wikipedia, the free encyclopedia (s. f.). Recuperado de [https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
- NetBeans Wiki (s. f.). Recuperado de http://wiki.netbeans.org/Main_Page
- Pugixml documentation (s. f.). Recuperado de

- <http://pugixml.googlecode.com/svn-history/r453/trunk/docs/index.html>
- Qt Creator - Wikipedia, the free encyclopedia (s. f.). Recuperado de https://en.wikipedia.org/wiki/Qt_Creator
 - RapidJSON: Main Page (s. f.). Recuperado de <http://rapidjson.org/>
 - Top 10 New Java Features in NetBeans IDE 6.5 (Part 1) (s. f.). Recuperado de <https://dzone.com/articles/top-10-new-java-features-netbe>
 - Data structure alignment - Wikipedia, the free encyclopedia (s. f.). Recuperado de https://en.wikipedia.org/wiki/Data_structure_alignment
 - Object composition - Wikipedia, the free encyclopedia (s. f.). Recuperado de https://en.wikipedia.org/wiki/Object_composition
 - Parallel computing - Wikipedia, the free encyclopedia (s. f.). Recuperado de https://en.wikipedia.org/wiki/Parallel_computing
 - What is TCP/IP (Transmission Control Protocol/Internet Protocol)? - Definition from WhatIs.com (s. f.). Recuperado de <http://searchnetworking.techtarget.com/definition/TCP-IP>