The Design Document

CS 300 - Group Project (Team 1)

Bader Alshaya

Boodhi Alshaikh

Dhwanil Chokshi

Zack Salah

**Table of Contents**

***1******Introduction***

*1.1 Purpose and Scope*

*1.2 Target Audience*

*1.3 Terms and Definitions*

***2******Design Considerations***

*2.1 Constraints and Dependencies*

*2.2 Methodology*

***3******System Overview***

***4******System Architecture***

*4.1 Provider’s Terminal*

*4.1.1 Provider’s Directory*

*4.1.2 Disk Record and Copying to Members and Providers*

*4.2 Manager’s Terminal*

*4.3 Operator’s Terminal*

***5******Detailed System Design***

*5.1 Manager’s Terminal*

*5.1.1 Provider / Member Verification*

*5.1.2 Disk Record and Copying to Members and Providers*

*5.2 Provider’s Terminal*

*5.3 Operator’s Terminal*

# **1 Introduction**

***1.1 Purpose and Scope***

This paper marks the second phase of the ChocAn’s software development process. The activity is commonly known as the system and software design phase. We describe on this phase the overall architecture of the ChocAn’s software based on the specification derived from the requirement analysis, which was done by the development team to ensure a common ground between the customer and the different members of the team. We also define it within the document all the fundamental abstraction of the ChocAn’s system and how exactly they interact with each other accordingly and what the general theme of the project along with the more explicit technical issues that must be solved within the ChocAn software.

Generally, the core objective of this document is to provide a more technical resource available for the development team to work and plan the entire process of the ChocAn’s project. Henceforth, this resource provides the development team with several advantages during the entire process from the early initiation step of the project to the actual deployment and the regular maintenance of the software.

One of the software engineering advantages of the design document lies under the concept of writing a clear and a cohesive guideline for the development team. On the other hand, having such a plan helps drawing a clear path for the development team and, thus, helps in the process of dividing the work and distributing teams. This is more of a generic step to the well-thought planning that eventually helps making the project more manageable during the process and in the coming future.

To begin with, it is important to discuss the different sections of this paper in abstraction and what each part is demonstrating. This part is intended, however, to be a brief preliminary and an abstract description of the role each part plays in the ChocAn software development process. Hence, further examination of each section to collect all the necessary information is vital for a full comprehension of the content delivered by the system and software design document.

The first section is the Design Considerations section. The section explains in specific details the possible limitations that might be raised by this project, how they’re going to affect the development process of the software or the development team, and, eventually, the overall project. Furthermore, the section emphasizes the different dependencies that must occur as part of the design process. How one abstraction or system of abstractions relies on another higher abstraction and contribute in the overall system of the software. This allows developers to think more critically about how they're going to go about the design and what test cases should be included within the system.

That brings us to the third section of this document; System Overview. The section deliberates on the more general architectural design of the software and how each part is related to one another in a more descriptive approach using UML diagrams and visual aids. However, this section doesn't touch on the architectural details, but rather a brief summary.

The System Architecture section, on the other hand, describes the system as a blueprint. It states on a high-level the different features, functionalities, data structures and classes being used within the software development process of ChocAn. Finally, the Detailed System Design is the most intensive section of this paper. It extends the last section and go into a more specific details from where the last section left. It also shows the pseudo code that is used on the ChocAn software and other programming details.

***1.2 Target Audience***

The target audience for this paper is narrower than the requirement paper. It is generally meant for the development team and all the people within that project that has an expertise in either the programming side or the engineering side that occurs before, during and after the implementation of the software.

***1.3 Terms and Definitions***

The following table is a comprehensive list of all the terminologies and technical phrases that is used to describe specific parts of this project:

|  |  |
| --- | --- |
| **Term** | **Definition** |
| ChocAn (Chocolate Anonymous) | Is the organization sponsoring the software |
| Member | A registered customer |
| Provider | A registered employer |
| Manager | A supervisor of providers and financial transactions |
| Operator | A person responsible of overridden employers and managers from the system |
| Terminal | A special device that act as the interface between provider, manager and operator and have a specific functionality |
| Provider Directory | A list of services offered by ChocAn |
| Data Center | A database for member’s information |
| Disc Record | A database for the weekly service transaction’s details |
| EFT (Electronic Funds Transfer) | A financial platform that sends weekly payments to providers |
| AAS (Acme Accounting Services) | An organization responsible of member’s service status, registration and fees |

# **2 Design Considerations**

The purpose of this section is to talk about the constraints of the system and the dependencies of system’s components on multiple factors. The constraints strictly goes over the factors of designing that are going to enforce limits on the creation of certain parts. The dependencies will cover an in-depth analysis of how components of the system will work if and only if the other components they depend on work.

Additionally, the section covers the software engineering methodology we will use to tackle this project. Since the software needs to be created as per the requirements, it will be beneficial for us to use Waterfall methodologies to tackle this project. This will allow us to move forward after completing each requirement thoroughly, eliminating any expensive necessities to go back and edit or revisit parts of software.

## **2.1 Constraints and Dependencies**

The constraints for this project will typically fall under three categories; time, results, and requirement constraints.

**Constraints**

* Time Constraints

1. The first time constraint is to finish the requirements, design, and coding of the project on the days it was promised. This is a constraint which directly affects the developing team and indirectly affects the customer. Because, failure to adhere to strict due dates will cause us unnecessary delays in finishing software features which will result in a delay to delivering our product to the customer. The waterfall methodology tends to build on the previous stages, therefore it is vital that we finish all parts thoroughly and precisely so we don’t have to revisit the previous stages of the development phase.
2. The second time constraint is very project specific. One of the customer requirements state that an email attachment should be sent to all the members about the services that they received during the week. Also, an email should be sent out to the providers about the services they provided to the members during that week. These emails should be sent out at midnight on fridays. These forms are important for customers and providers to keep track of the services in case there is an emergency and all information is lost. Therefore, the time constraint of sending out the email attachments on time is an important feature of this software which should be implemented correctly.
3. The third and one of the most important time constraint is individual task completion. Each member of the team has to make sure they are aware of the correct start time and end time for their part. The development team and project management team should make sure that each part of the project is being completed as it was promised on the date it was promised. This can only be possible if each member is aware of what they have to accomplish and that they ensure that it is accomplished.

* Result Constraints

1. The first result constraint is validation constraint. All of the validation tests must be passed to assure that the final product is the exact same thing that the customer had asked for. Validation test have to ensure that the product is not overly complicated and does not deviate from the final vision. This is an important constraint because it can become very easy for developers to over-complicate the product, deviating from the main requirements and creating something that would only disappoint the customer.
2. Another constraint is the design constraint. We, as developers, have to simplify the user interaction with program as much as possible. It is important to keep in mind that the people who are going to be using the software may not be tech savvy. Therefore, when creating the provider’s terminal, we cannot display complex error messages that no one can understand. Furthermore, the main result constraint is having a fully functional project done by the time it is due. The software should include an interface, accepting member and provider cards/numbers, saving all the details about services provided, save all provider/member information, and be able to send out reports.

* Requirements Constraints

1. There might be a time when a developer might wish he could work on certain parts to make the software better but he/she can’t because of requirements constraint. With ChocAn, one of the requirements constraint is designing the interface. The provider and managers will be provided with a simple terminal that they will use. While a terminal does maintain simplicity, it can seem quite blank, boring, and dull. Therefore, we as developer want the fruit of our reward to be proportional to the amount of work we put in. If we are creating a software as big as this for ChocAn, then it makes sense that we go the whole way and make the interactive user interface also even though it might add on the work. However, this is a requirements constraint because the job of creating a user interface is the responsibility of another organization. Our responsibility as contract developers is to create the base software which has the basic functionality that meets all the requirements.
2. Other requirement constraint is that we are not creating software which will be used with internet connection connected to a remote server which fetches information for data processing. Instead we are creating an offline application which does not need any data fetching, because everything is going to be stored in text file. However, we would prefer having a server which can help our project become more centralized. Also having servers would allow us to store all the information in one place, allowing us to back up periodically so the data is not lost. Furthermore, we would have the advantage of scalability, meaning new updates can be added on the server to benefit the software, and also we could have accessibility so that the data can be accessed remotely. However, this is a huge requirement constraint holding us back from creating a software which would take the whole system to the next level.
3. Also there is a very limited choice on how to recover lost data. Since we are not connected to a server and since the security development is not part of our job, we have very less choices in case of system failure. The development team is going to use data structures to store the data from the file, therefore every detail about every service, member, and provider will be stored in a file. If the system crashes, causing the information from data structures to be lost, we can quickly have a function read from file to get the data back into the system. However, this is a limited choice because of the way the requirements are laid out for the design. The system could have a more sophisticated security network which would help the users recover data faster but it is not part of our job.

**Dependencies:**

As with any complex software system which is created today, there are going to be subcomponents of the system which are dependent on other subcomponents. The system cannot be created without dependencies, otherwise the whole system would be ambiguous. Therefore, to guarantee that the system is created successfully, all of its subcomponents must work as a whole to bring out the best functionality. Below is an in depth discussion of the dependencies regarding the software we are creating for ChocAn.

1. **Member/Provider/Developer Dependency**. The very first and foremost dependency we see is that ChocAn is dependent on members, providers, and developers. Even though this may not strictly be a technical dependency, it does provide us the big picture behind all the factors that are helping ChocAn come together and deliver on their mission. Without members, there would be no need for providers to provide services. Without providers, there would be no members to give service to. Without developers, there would be no centralized system the providers and members can use to document their services. Therefore, these three groups play a huge role in bringing ChocAn together for the creation of service which they can offer to help individuals.
2. **Textfiles**. Getting down to the technicalities of this software project, we realize the importance of text files. These files are acting as our main source of information storage making them a big part of this project. The whole system will be dependent on text files to get the information during various stages of consultation. The member information and provider information is stored in their respective text files. Therefore, member verification by the providers is dependent on the member text file which checks the records and guarantees that the member is valid or invalid. The dependency of accessibility of the whole system is based on text files. The text files will be important in checking the providers also. The providers have their respective numbers which make sure that they are valid providers, and not merely some person posing as a provider to get into the system. The confidentiality of information about members and providers is dependent on their text files, therefore it is important that the text files are secure and stored safely.
3. **Disk Record**. The disk record may seem like an unnecessary addition to the software project, but it plays one of the key roles in ensuring that the information is delivered. After each service provided, the disk record stores one of the key pieces of information; the current date and time. Therefore, the date and time of the service is registered on to the disk record. The provider form, which is sent out to the provider once a week then uses the information on the disk record to create a legitimate copy which will be sent out. Therefore, the provider and member form are dependent on the disk record. If the disk record cannot store information, then there is no proof of the service(s) provided by the provider to the member. Furthermore, the system of ChocAn is dependent on the disk record to make sure no providers are cheating the system. Since it stamps the exact date and time and writes out to the file, there is no way a provider can cheat the system with adding the fees and services which they have not provided.
4. **Summary Report and membership.** The payment for ChocAn providers for their services is dependent on the summary report which is generated and sent to the manager of accounts payable. The summary report consists of the total added up fees for the services that was provided so the providers can be credited the proper amount that they have earned. Furthermore, the admission of the members into the ChocAn system requires membership fees. Members who do not comply on paying these fees will be suspended until the fees have been paid. Even though the job of handling the membership fees has been given out to ACME, it is still a dependency on the overall network framework of ChocAn.

## **2.2 Methodology**

The methodology we plan to use for this software project is **Waterfall Model**. This is the most basic and one of the oldest software engineering model which is still used today. Even though there have been introduction of fast paced software development models like Agile, Waterfall still holds a breakthrough methodology for tackling large scale applications, applying fundamental methods of software development and planning.

There are a number of reason we chose the Waterfall method.

1. Waterfall model has a very strict drop down from one stage to another. For people who follow this model religiously, it will be a violation of the rules to move through stages while leaving the previous stage incomplete. The waterfall methodologies apply very strict rules towards the how software development should take place. Rigorous requirements gathering, design process, testing, and coding are some of the characteristics which drew our development team towards adapting the waterfall model. The requirements were gathered very precisely for ChocAn. This included reading the project description multiple times to make sure that every piece of request from the customer has been gathered, leaving little room for errors or revisitation. Therefore moving on the design without completing the requirements would have proven to be very expensive for our company. The designing phase of the model is currently being employed as this document is being finished. The design covers the abstract level documentation and in depth system architecture of the whole project. It is essential that every part of the system is documented in the design for developers to have an idea of how to distribute task and finish each part on time. The implementation part of the software will be done after the design. The testing phase is towards the end of the development model and will be completed after the code has been finished and the software needs to be tested.
2. Furthermore, we have chosen waterfall because it can be expensive to revisit some parts of the software. Due to the complex nature of the dependencies that exist in the architecture, it can become overwhelming and costly to revisit requirements, design, or implementation and modify them. Therefore, it is important that every developer working on the project is in sync with the rest of the team during each stage of the process.
3. A large scale application like the one being created for ChocAn will need to use a methodology with strong footprints in the world of software. This large application is beneficial to the software developers because there is going to be little to no change in the initial and final vision of the product. The software has a list of requirements which are not likely to change, because changing one part of the software means changing the dependent components also which would only further delay the project. Therefore, since this application is huge, it makes more sense to use a model like waterfall which will help the team precisely tackle each level of creation. Additionally, there is no need for prototyping the software. The customer have given us a list of what they want, and we are going to implement all of the features. Therefore, the little to no change in the software will require no need for the customer to come in and test it consistently.

# **3 System Overview**

ChocAn Data Center software will help simulates the employee's work. The software will provide a terminal in which the providers, operators, and managers interact with. Each employee must enter his or hers ChocAn card numbers in the terminal to access the menu corresponding with their job. For example, the provider’s terminal contains two items, search for provider’s directory and add a service to members, while the operator's terminal has three items, add, delete, and update for members, providers, and managers. Furthermore, the software will also generate files to members, providers, and managers. Further explanation is listed below:

* Providers terminal:

After the provides enters his or her card number the software validates and direct the provider to a menu.

* Menu:
* Provider’s directory

The provider needs to access the provider’s directory each time he or she adds a service to a member. This ensures that each that the provider enters the correct code corresponding to the services provided to the member. Additionally, it ensures that every provider has access to the services list. The provider’s directory is an important piece of the program because it contains the services, with the service code and service fees. The fees are copied into the provider’s form, so the accounts payable can credit all individual providers who have provided services with the right amount.

* Adding services for Members

Each time the provider provides service, it is documented in the disk record which saves it in a file, saves it in the BST for providers, and saves the information in the BST for members. This eliminates the need to store all the information at once and write to the BST later. Storing the information in the file will guarantee that the services providers have provided will not be lost when the system shuts down or crashes due to complications. Furthermore, a member can receive multiple services in a week and the providers can provide multiple services in a week. All the service information will be documented on the disk record and provider/member form as proof.

* ChocAn Data Center - Reports

After a member receives a service from a provider, they will receive a file containing the list of services they previously had. The file will contain the date of service, the provider name, and the service name. As for the providers, they will receive a file containing the list of services they provided to ChocAn members. This file will also, contain the number of consultation and the total fee that ChocAn owes the provider.

* Managers terminal

Summary reports: The summary reports are important in keeping tabs on the payments the providers will receive from ChocAn for the services that have been provided each week. The manager will get a summary report which describes every provider that is to be paid that week, the number of consultations each had, and his or her total fee for that week. This summary report is used with account’s payable to credit the correct amount of money into the provider’s account. Summary reports are generated once a week to help ChocAn provide the correct payments. If there are discrepancies in the payment, then the service can use the form or disk record information to double check all information. Therefore, the summary report is dependent on the provider and member forms being finished.

Individual Provider Form: The manager can also pull out individual provider forms to check the information or for other miscellaneous purposes. The manager terminal has access to all the providers and all their information about the services at all times during the day when the ChocAn data center is functioning.

* Operators terminal

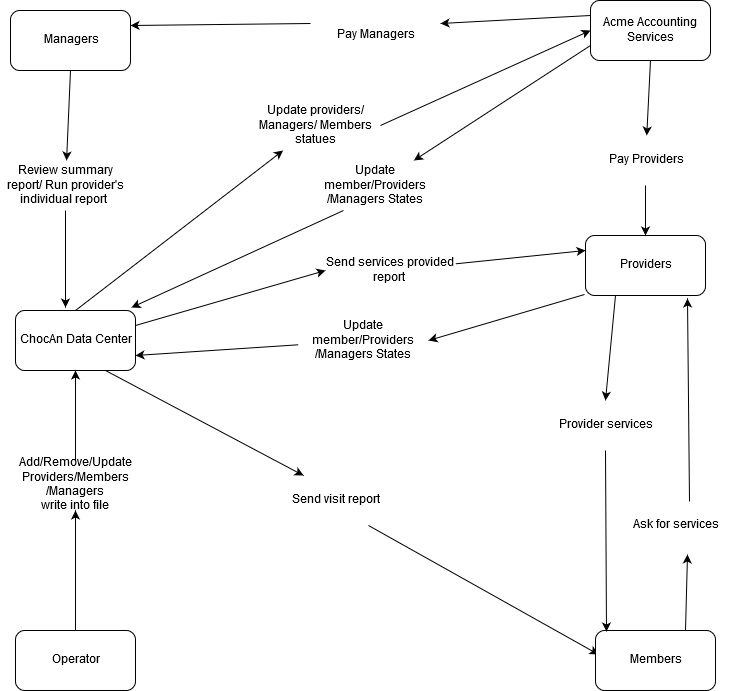
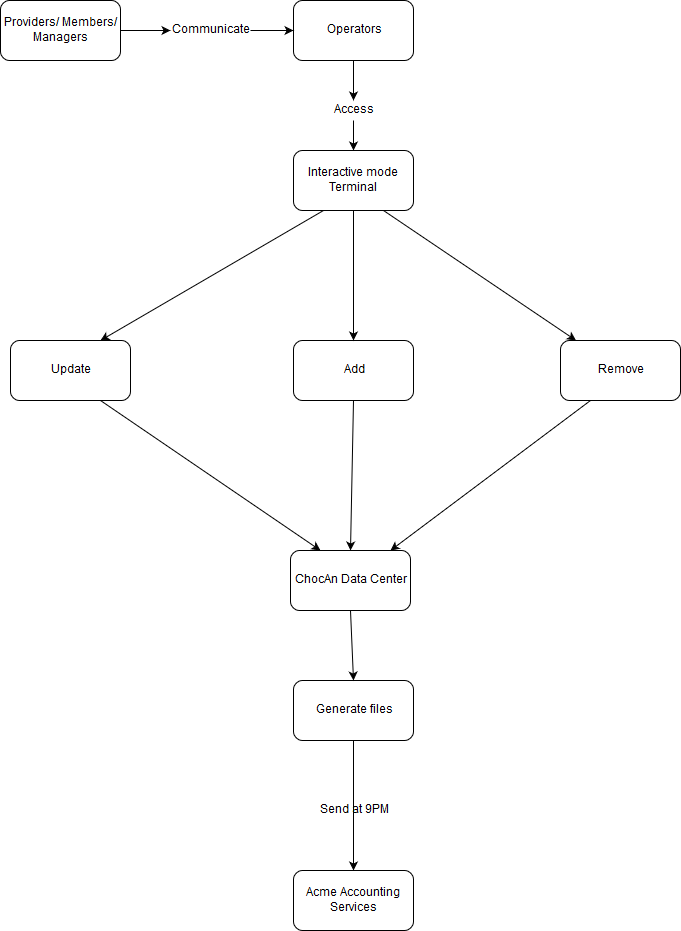
The operator’s terminal is responsible for the manipulation functionality of software.

The operator’s terminal will allow the operators to add members who are new to ChocAn, delete members who have resigned from ChocAn, and update member records. Similarly, the operator can adjust the provider records as needed. ChocAn operators are the only individuals from the staff team that have the administrative privileges to add, delete, and adjust member and provider record as needed. When members and providers need to be added or deleted, the operators will remove do the necessary operations and create a new file which will be sent to ACME. Every evening, ACME will update the file into the ChocAn servers so the necessary changes have been added.

* Diagrams Below:

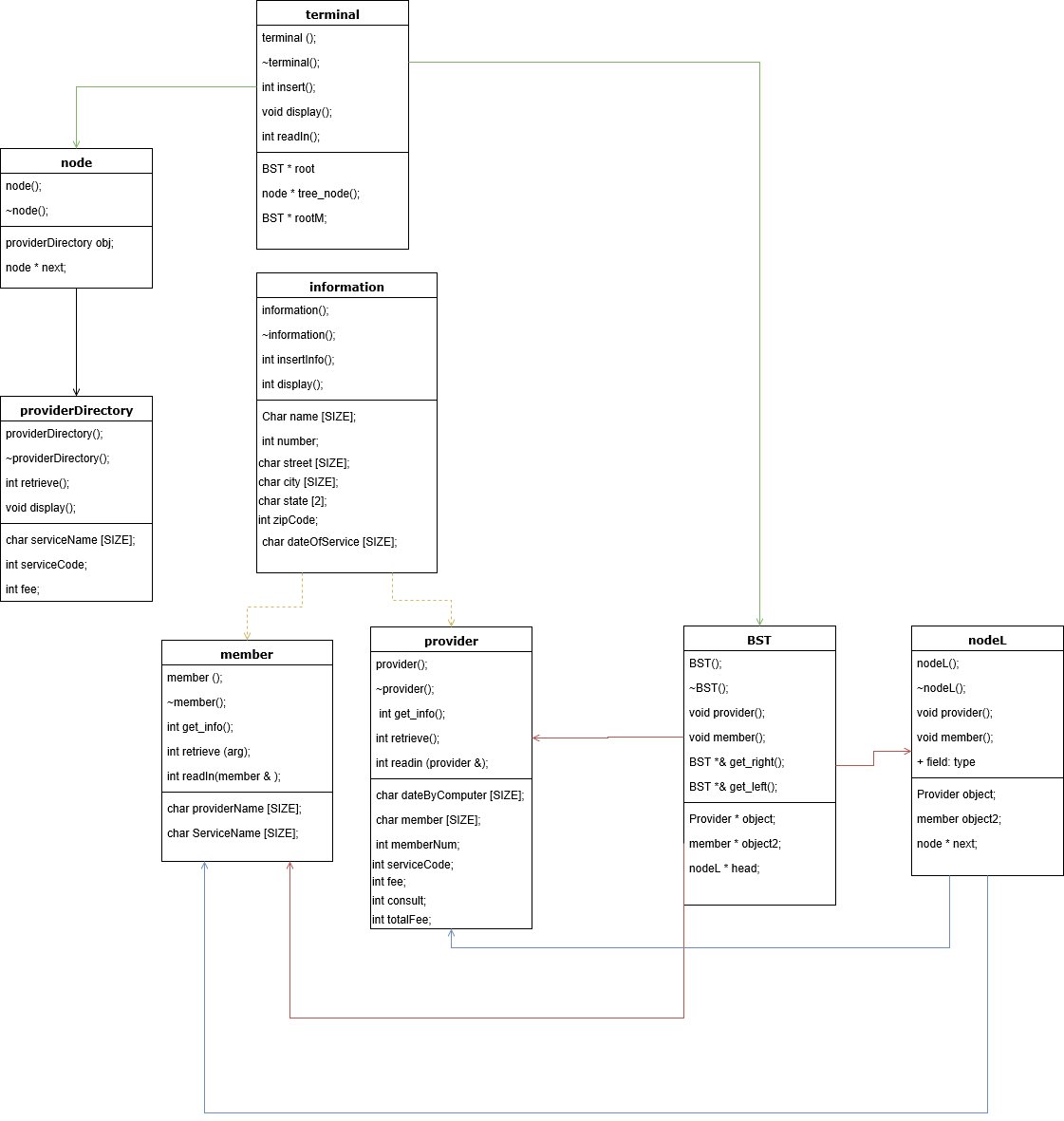
The first diagram below shows the communication stream that the providers and members can use for getting the information adjusted. We see that the members and providers both have to have accessibility to the operators when there is a need for manipulation of data. The operators are the only people who have direct, strict access to the addition or removal of members and providers. The software is more safe when tasks are distributed evenly across the spectrum. The ChocAn could have the addition and deletion features be in the provider’s terminal, but that would make it very easy for providers or malware to manipulate information causing harm to all the data files and information. Therefore, outsourcing the job of deleting and adding members/providers to a separate staff party like operators is necessary.

The second diagram is a diagram for the whole system. It represents the unique connections between the different components of ChocAn and other third party vendors that play a role in the functionality of ChocAn as a whole. The diagram also represents the entities with their respective relations to other entities.



# 

# **4 System Architecture**

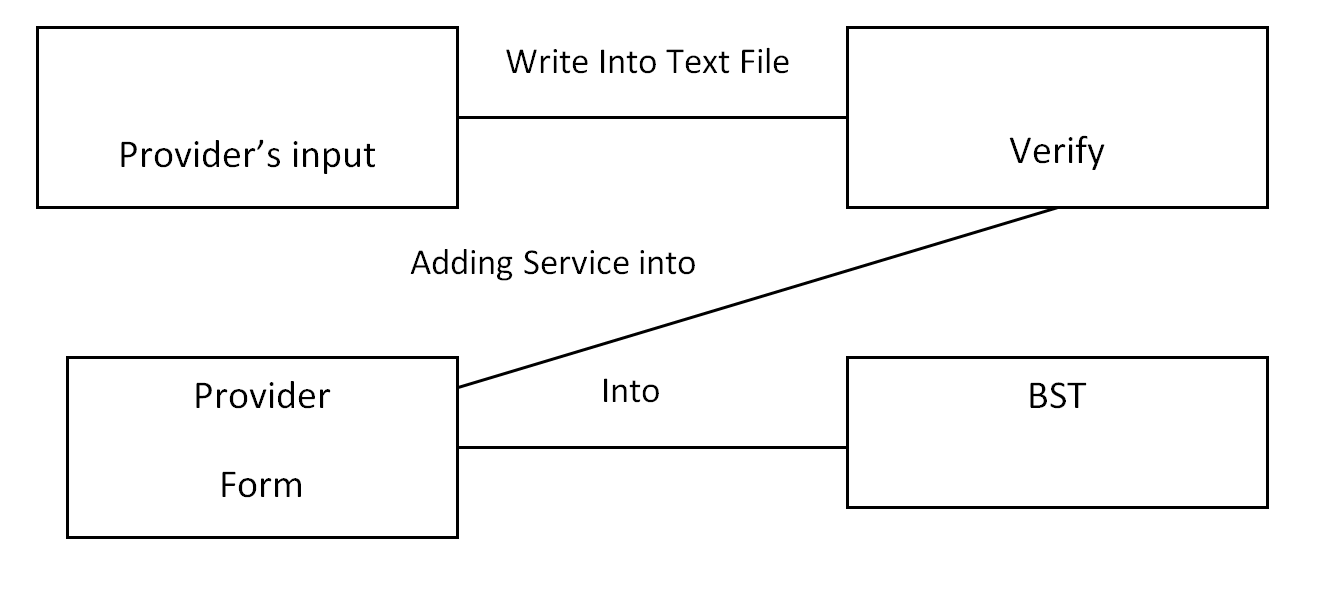


The UML diagram above explains how the classes in our application will behave and what type of relationship each class will have with another. The terminal class will act as a manager class that will be responsible of creating the data structures. The terminal class will have containment relationships with multiple classes like node class and BST, but it will have two BST pointers because we will be using that class for creating a binary search tree for the provider forms and the member forms. There will be three different terminals in the application and the terminals will be determined by the code entered by the user. For example, if the number was entered with a number that starts with 111 a manager terminal will be opened, if a number starts with 222 then a provider’s terminal will be opened and if the number starts 333 an operator terminal be opened. These terminals will be the most important parts of our system and by these terminals our system will act as a whole.

## **4.1 Provider’s Terminal**

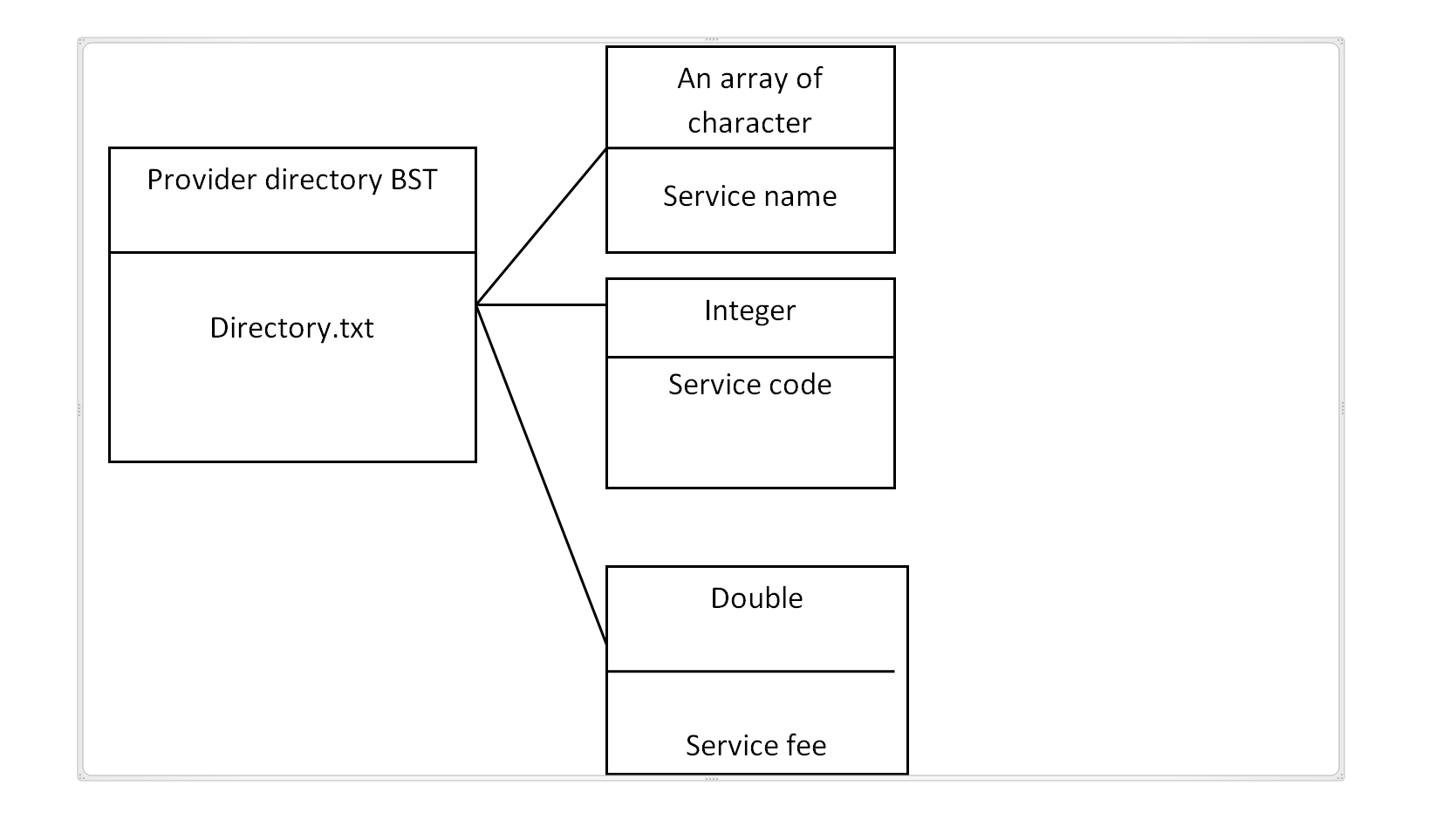
The provider terminal is going to be the main subsystem because it will be the main information we will rely on to store data and it will be the one to check if the member’s status is valid. Then the provider is going to be prompted to enter the service code which will be retrieved from a binary search tree sorted by the service name.

Furthermore, the provider will add the service for that member if the service was found. Then the provider will be prompted to verify the information and that will be saved into an external text file. That information will be not saved into a data structure because there will be no purpose of doing that since we only need the data for the provider’s form.



### **4.1.1 Providers Directory**

The provider’s directory will be a binary search tree sorted alphabetically by the service name. it will be created from the terminal class by reading data from an external data file. The provider’s directory will be the reference for the provider to get information about all service that ChocAn offers, so that the provider can add the service to a member. The provider’s directory will have the service’s name, services code, and fee. That information will be needed to store data into the provider’s and the member’s forms. The provider’s directory will output an error message if the code service wasn’t found in the binary search tree. The list of services will be supplied for the application by a third party company, so no terminals will have the permission to edit it.



### **4.1.2 Adding services to members**

Services will be added to members as forms in a Binary Search Tree, but we will not have duplicates of a member’s data in the BST. However, we will approach that by making a linear linked list for members with multiple service forms. We will get the data of the service from the provider’s directory then store it into a temporary object of that class. Then the application will search in the BST for that member. If the member wasn’t found, a node will be created for that member and the service information will be copied into that node, but if the member was found in the BST, a linear linked list node will be created for that tree node and then the service information will be copied along with the member’s information. Member forms will be written into an external file with the member’s name and the date of the report.

## **4.2 Manager’s Terminal**

The manager’s terminal will be responsible of reviewing the summary reports for the provider’s. The report will be generated by looking into the BST for the providers from and get their information. However, providers who didn’t offer any services for any members will not be in this report. It will keep track of services fees The summary report will be saving into a text file as follows (weekly\_Summary.txt ):

This is the weekly summary of ChocAn providers for the services they provided for the week.

ONLY providers who provided services will be listed in this file.

The Total Number of Providers: \_\_\_

The Overall Total Fees: \_\_\_

The Overall Total of Consultations: \_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Provider’s Name: \_\_\_

Provider’s Code: \_\_\_

The Number of Consultations: \_\_\_

The Total Fee: \_\_\_

\_\_\_\_\_\_\_\_\_

## **4.3 Operators terminal**

The Operator’s terminal will be responsible of adding providers for chocAn. A list of providers will be read from a text a file. The providers will have the option to add providers through the text file or through the application. However, the main adding feature will be through the application because the text file must be written in a specific way, so that the information can be read back into the application. It will also be responsible of deleting provider’s record and update them as well. Furthermore, the operator's terminal will be updating, adding and removing member’s forms.

# **5 Detailed System Design**

This section of the document contains a deeper level description of the whole software system. We describe the system through the representation of functions and data structures we will use which will aid in the flow of our product. The description of the system focuses on three main subcomponents (provider’s terminal, manager’s terminal, and operator’s terminal) of the project that will help us build our project.

## **5.1 Provider’s Terminal**

Provider’s terminal is one of the most important parts of this project. Without the provider terminal, there is no way of ensuring that member and provider verification has been completed to make sure that the member to whom the service is provided is valid. Therefore, to avoid unnecessary conflicts and confusion, the provider’s terminal will have access to many parts of the software.

The two important subcomponents of the provider’s terminal is provider and member verification and copying the information from the disk record to the member and provider form, so we can have it all saved.

### **5.1.1 <Provider/Member Verification>**

So when the provider is using the computer, to access the terminal they need to enter their respective provider number. Also, it will have adding member’s

We ask the user if they are provider, manager, or operator. If they pick provider or manager, we ask them to enter their number and check if they are verified and part of the system. Below is the pseudo code and helper functions we will use when a provider is using the system.

int number = //ask the provider to enter in his/her number and save it;

//since terminal has root pointer, we will call the BST function through root

terminal\_take\_provider\_number()

{

//the verify function will traverse the tree and check the number with each provider

that is in the system

if(p\_root->verify(number) == success) //p\_root = provider root

//give access

else

//state the number was incorrect and ask them to enter it again

//traverse with the root

}

verify(number)

{

return provider.verify\_provider\_number(number);

//this function will do the literal checking of the number to see if it matched

by calling the function using the provider object

}

verify\_provider\_number(number)

{

//this function will contain a simple if statement to compare the number that is coming in

with the number that already exists in that node, returning success or failure

if(number == already\_existing\_number)

return success;

else

return failure;

}

The member verification will be done in a similar manner. Each member that is saved in the BST will have information if they are invalid or valid, so the provider knows if they should provide the service or not. This will also require a simple checking with data fields to see if the member is valid or not.

//Once inside the provider terminal, the provider can check if the member is valid or not.

int m\_number = //enter the member number/slide their card

check\_member\_validity()

{

if(m\_root->verify\_member(m\_number) == success) //m\_root = member root

//display that the member is valid

else

//display that the member is invalid and deny service

//traverse with m\_root

}

verify\_member(m\_number)

{

return member.check\_number(m\_number);

//function which provokes a function using the member object to verify the member

number

}

check\_number(m\_number)

{

//contains if statement to check the member validity

if(m\_number == m\_number already existing in the node)

{

if(member\_validity == valid)

return success;

else

return failure;

}

else

return failure;

}

### **5.1.2 Disk Record and Copying to Members and Providers**

The disk record is going to store all the information after the service has been provided to the member. The provider enters the information about the service, and it is immediately written out to the file, copied into the BST for providers and members. The BST nodes for the members and providers will have a linear linked list containing the member and provider forms, since the provider can provide a service multiple times in a week, and a member can receive a service multiple times in a week. This is also an important part of the provider’s terminal

//from the provider’s terminal, once the service has been provided, the provider will enter the information about the service which will be directly copied into the BST

//once the service has been provided

//look up the service that was provided in the provider’s directory

int prov\_number = //get the provider number

//save the information about the service and date into their respective variables and

pass it into the function which will copy it into the BST for member and providers

info = date + current date/time + service name + provider number + member number +

service code + member\_info;

//call the function in terminal to pass the provider side of information to the BST and copy it there

enter\_disk\_record(info); ← copies all information for the providers into BST

//call the function in terminal to pass the member side of information to the BST and copy it there

copy\_for\_members(info) ← this function will do the same thing as it does for providers (below) except it will access the member root and store the information in the member node of the tree

//function definition (for providers -- (used exact same way for members except with member root))

enter\_disk\_record\_info(info)

{

//now the pass the information to the provider part of BST

//if the provider is found in the BST, then copy the information to the provider form

if(P\_root->verify(prov\_number) == success)

P\_root->copy\_provider(info);

//traverse with p\_root

}

copy\_provider(info)

{

//we will use the provider object to copy the information for that provider

provider.copy\_info(info);

}

copy\_info(info) //for the provider form

{

//access the linked list node and copy all the form information for the provider, with their

name, address, number, information about member and service provided to that member

}

## **5.2 Manager’s Terminal**

The manager's job is to check the validity of all the report generated by the ChocAn software to ensure that each provider is paid in full. The report will contain the list of providers who provided services to ChocAn members in a certain week. The manager terminal will allow the user to access the summary report and to run an individual report at any time.

Verify\_manager(Manager\_file\_name)

{

Return Manager\_number(Manager\_file\_name);

//This return true if the number is found in the file of valid manger numbers otherwise it returns false.

}

Manager\_Menu()

{

//This function allow the manager to chose any items displayed in the menu. After the item is chosen, the Menu functions will call the right function associated with the chosen item.

}

Access\_Summry\_Reports()

{

Provider\_Summery\_Generator(Provider\_Object);

//This function will go through all the provider’s reports and get all the data necessary for the summary report and generate the file.

}

View\_Summary\_Reports()

{

//this function view the file that has been generated.

}

Run\_indiviuall\_reports()

{

Return Access\_Provider\_report();

//This function will prompt to enter user to search for a provider report. It displays the report and returns true, otherwise, it return false if not found.

}

## **5.3 Operator’s terminal**

The operator's terminal is essential for the software. During the day, the software is switched to an interactive mode which allows operators to add, delete, and update records for members, providers, and managers. Moreover, it allows adding new members, providers, and managers. Similarly, it can remove them. All new and deleted members, providers, and managers will be placed in a file. This file will be retrieved by Acme from the ChocAn Data Center and send back a file with the valid numbers. Below is some of the functionality must be implemented in the operator terminal.

Menu()

{

//This function allow the operator to chose between update, delete, and add record data for members, providers and managers.

Int

}

Update\_Records(}

{

//This function is a menu that let the operators to update the record for Member, Manager, and providers.

}

Update\_Manager\_Record(Manager\_number)

{

//The operator must receive the card from the manager and keys in the number on the card to allow the operator to update the manager data.

}

Update\_Provider\_Record(Provider\_number)

{

//The operator must receive the card from the provider and keys in the number on the card to allow the operator to update the provider record data.

}

Update\_Member\_Record(Member\_number)

{

//The operator must receive the card from the member and keys in the number on the card to allow the operator to update the member record data.

}

Add\_Records(}

{

//This function is a menu that let the operators to add the record for Member, Manager, and providers.

}

Add\_Manager\_Record(Manager\_number)

{

//The operator must receive the card from the manager and keys in the number on the card to allow the operator to add the manager data.

}

Add\_Provider\_Record(Provider\_number)

{

//The operator must receive the card from the provider and keys in the number on the card to allow the operator to add the provider record data.

}

Add\_Member()

{

//Append new member to file send to Acme.

}

Add\_Manager()

{

//Append new managers to file send to Acme.

}

Add\_provider()

{

//Append new providers to file send to Acme.

}

Add\_Member\_Record(Member\_number)

{

//The operator must receive the card from the member and keys in the number on the card to allow the operator to add the member record data.

}

Archive\_Records(}

{

//This function is a menu that let the operators to add the record for Member, Manager, and providers.

}

Archive\_Manager\_Record(Manager\_number)

{

//The operator must receive the card from the manager and keys in the number on the card to allow the operator to remove the manager data.

}

Archive\_Provider\_Record(Provider\_number)

{

//The operator must receive the card from the provider and keys in the number on the card to allow the operator to remove the provider record data.

}

Archive\_Member\_Record(Member\_number)

{

//The operator must receive the card from the member and keys in the number on the card to allow the operator to remove the member record data.

}

Archive\_Member(Member\_number)

{

//Append removed member to file send to Acme.

}

Archive\_Manager(Manager\_number)

{

//Append removed managers to file send to Acme.

}

Archive\_provider(Provider\_number)

{

//Append removed providers to file send to Acme.

}