COMP 410
Software Engineering

Assignment 2
Case Study

Jason Bishop
3042012

July 17, 2015

Problem 12.20

Using the technique specified by your instructor (Gane and Sarsen's), draw up a specification document for the Chocoholics Anonymous product described in Appendix A.

Gane and Sarsen's structured systems analysis is a 9 step process to analyze the client's needs. The results of these steps for the Chocoholics Anonymous product are presented here. These specifications may need further modification because, as mentioned in the requirements phase, not all necessary information is available in Appendix A. Further discussions with Chocoholics Anonymous will be necessary to confirm and complete all specifications.

Data Flow Diagrams:

Data flow diagrams are provided to describe the system, showing data stores with transformation and movement of data within the system.

Figure 1 – Data flow diagram for maintaining the ChocoAn product.

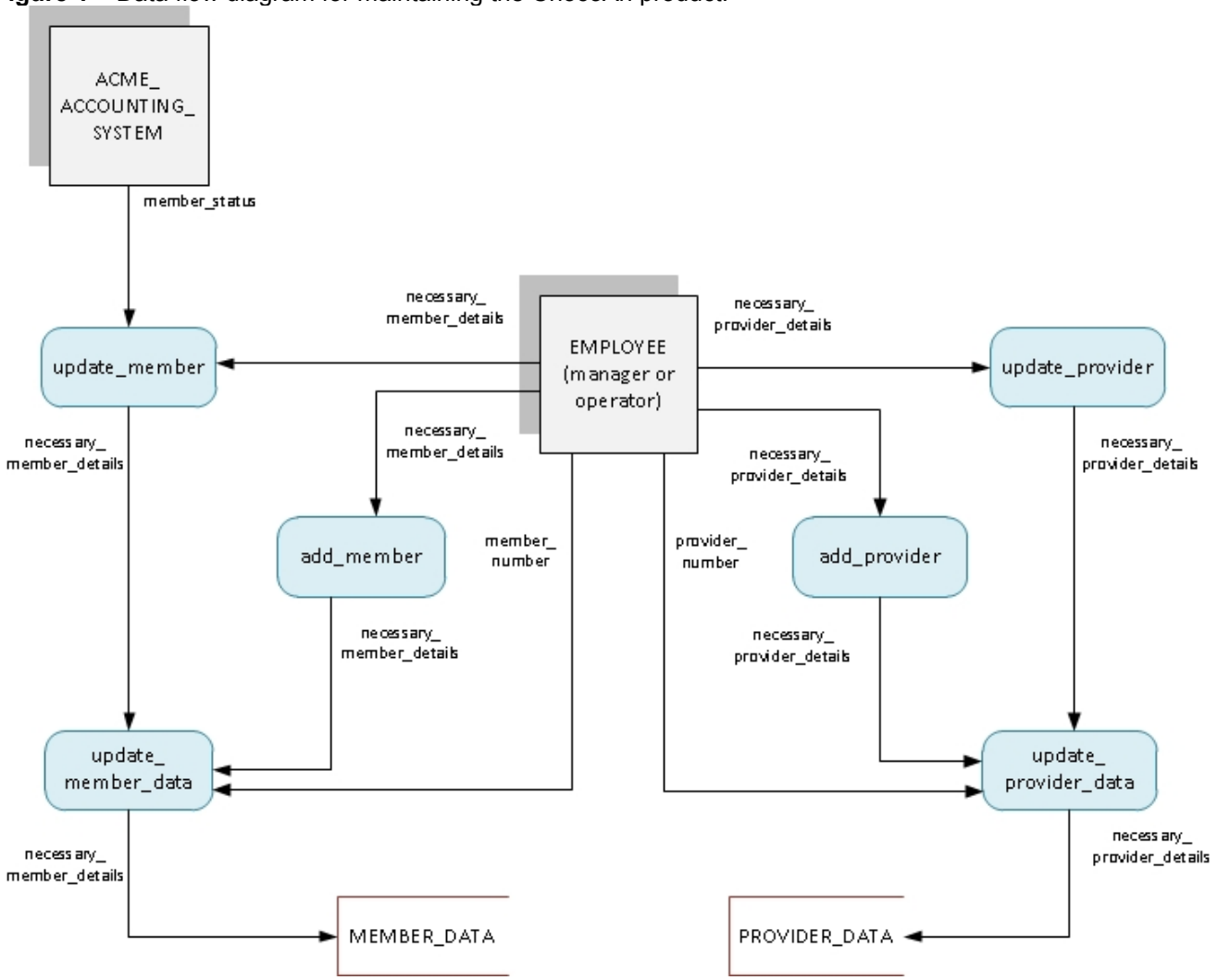


Figure 2 – Data flow diagram for report generation in the ChocoAn product.

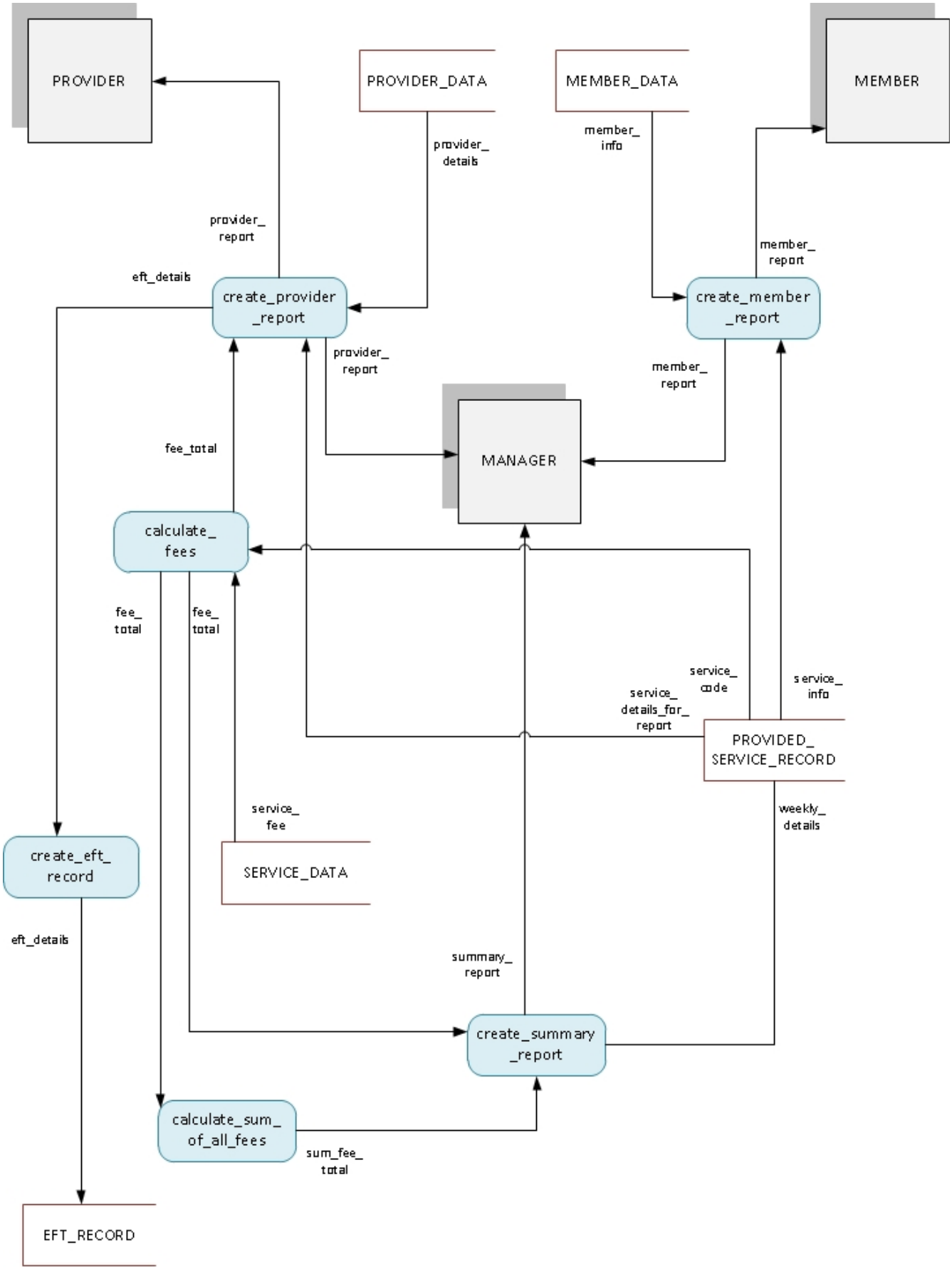
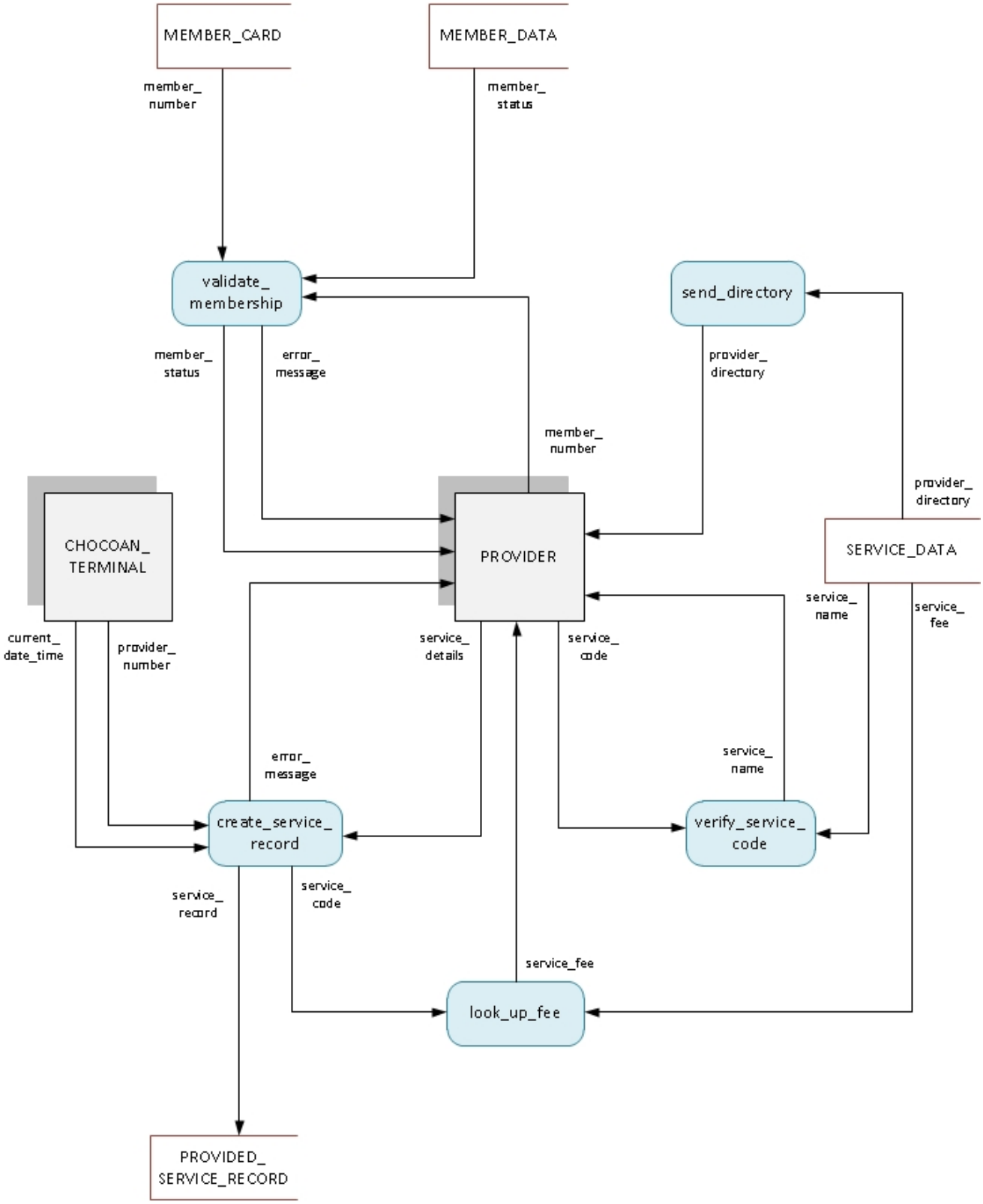


Figure 3 – Data flow diagram for provider interaction with the ChocoAn product.



Recommendation for Computerization:

The information from Appendix A indicates that Chocoholics Anonymous would like to computerize the entire process. This is definitely recommended as the interaction with the ChocoAn terminal at all provider locations necessitates computer interaction. Additionally, due to the large number of records of members, providers, and treatment information, keeping records up to date and organized would greatly benefit from computerization.

It is also recommended that most of these files be handled with batch processing. There are a large number of files to potentially process, and tight controls are needed to be able to keep track of services provided. This will allow reliable processing of all information and allow for accurate tracking of financial information.

Data Flow Details:

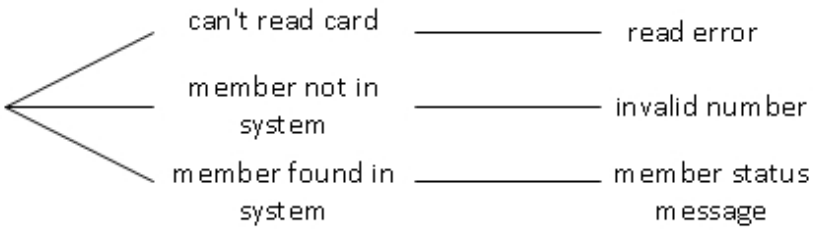
The data dictionary for dictionary for the ChocoAn project provides the details for the data flows. See the attached document "ChocoAn Data Dict.pdf"

Process Logic:

Decision trees have been made to clarify the various processes.

Figure 4a – Decision trees used in the ChocoAn product.

ChocoAn Terminal Member Card Messages



When to Print a Report

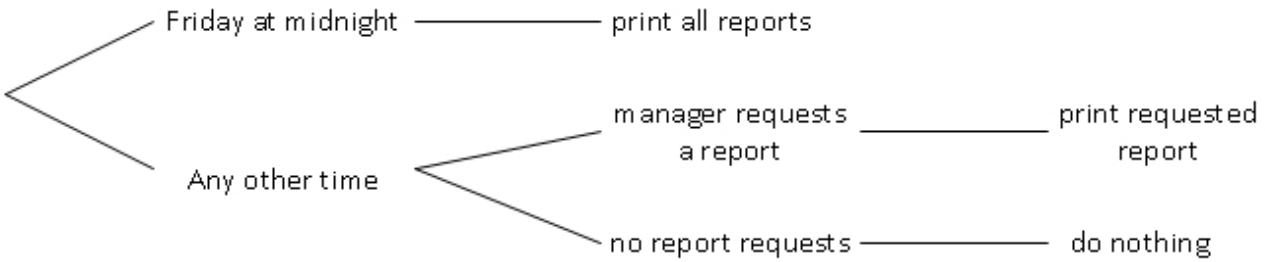
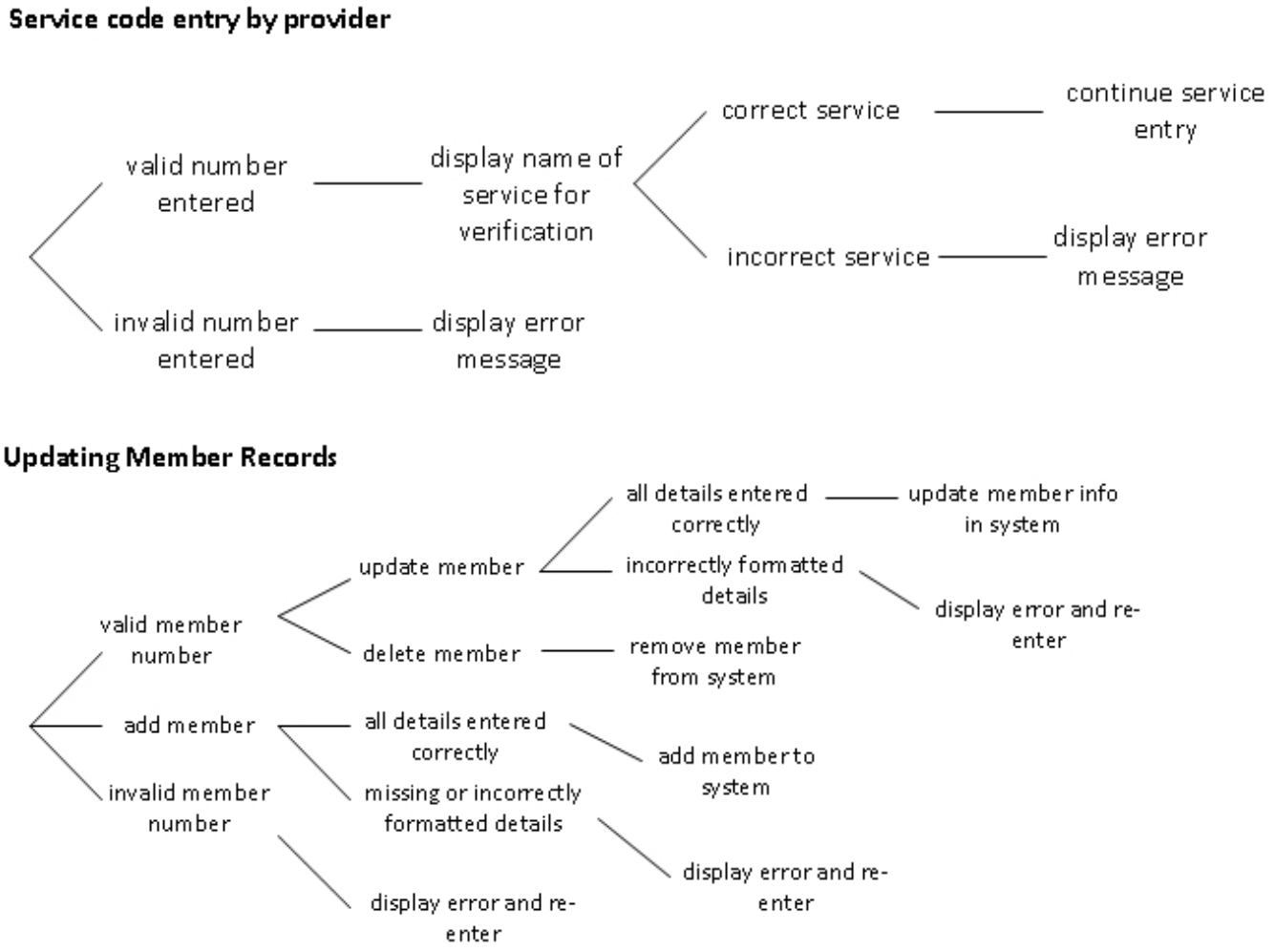


Figure 4b – More decision trees for the ChocoAn product



Data Store Definition:

The data immediate-access diagrams show the contents of the data stores and specify where immediate access is required.

Figure 5a – Data immediate-access diagrams for the ChocoAn project.

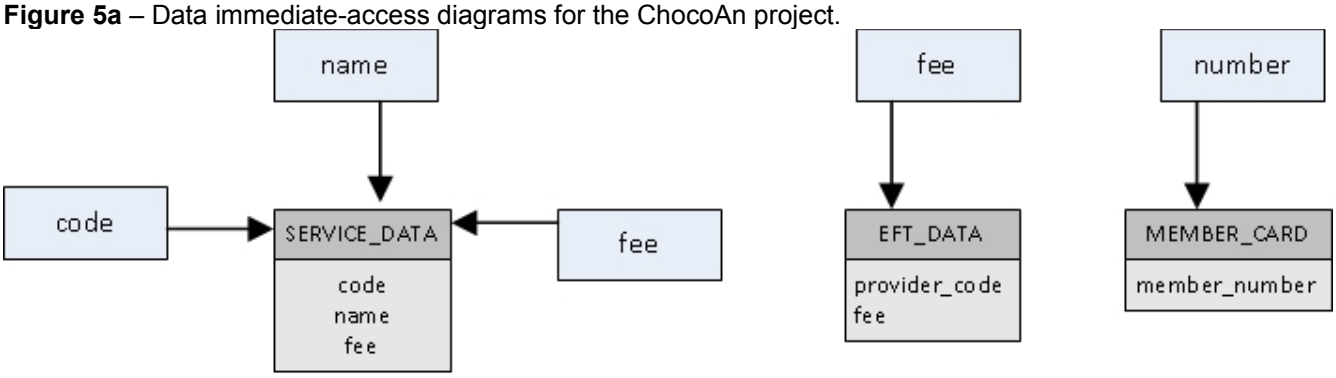
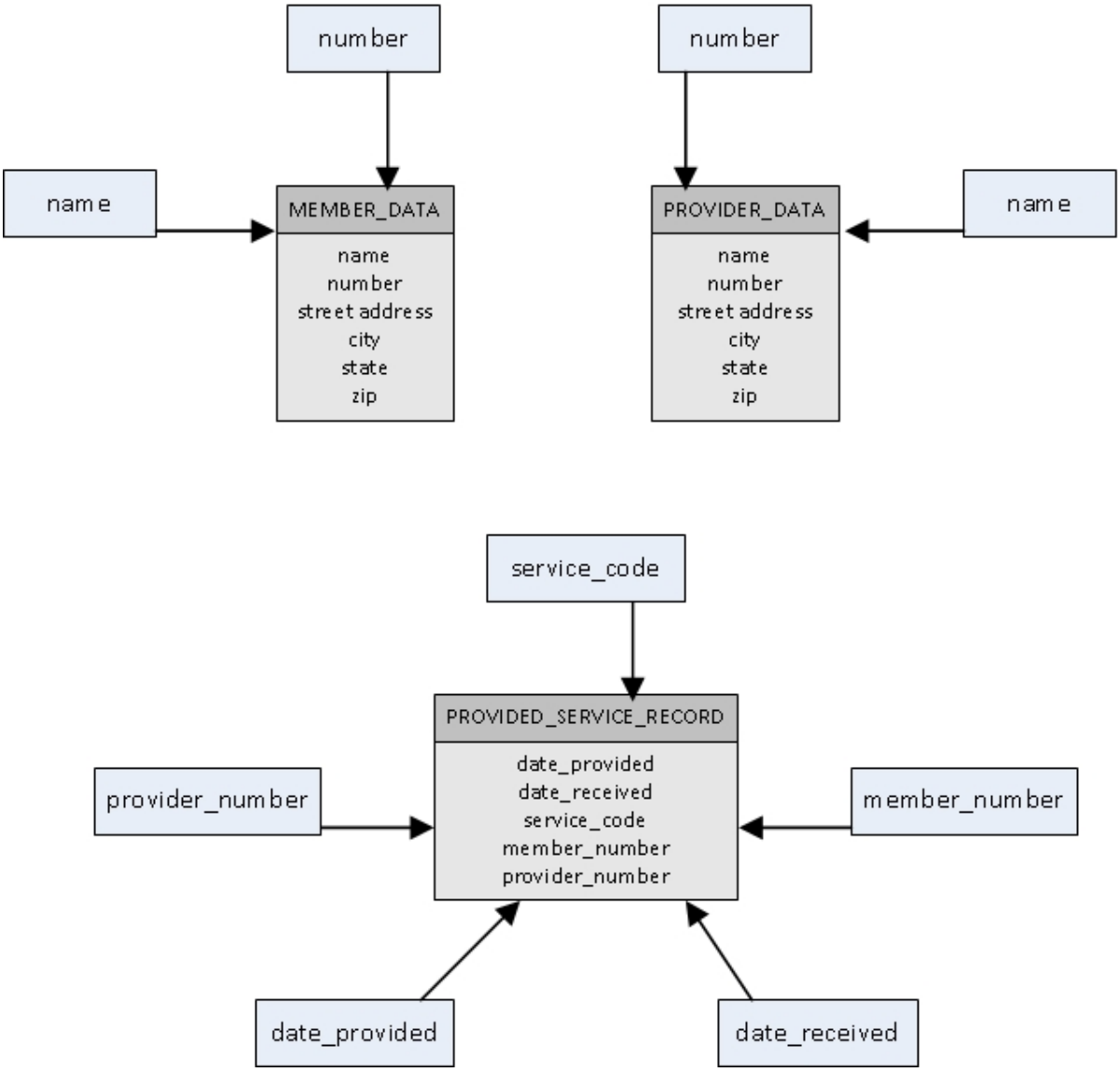


Figure 5b – More data immediate-access diagrams for the ChocoAn project.



Physical Resource Definition:

Without knowing the exact number of records in the ChocoAn system, determining blocking factors will be difficult, as we will not know the amount of data we need to be able to store. This will determine the type of hardware used, and thus determine the block size available to us on the physical storage media. More information will be required from Chocoholics Anonymous. Moreover, the decisions about the hardware for the database and associated DBMS require additional information before making decisions about these specifications (see the section "Hardware Requirements" for more details).

File names for printed reports have been specified. Member reports should begin with the member name, followed by the date of the report. Provider summaries are named in the

same way using the provider name. The Provider Directory must also be set up as a file, but the exact format of the name is not specified. The same is true for the EFT records, where only the content of the file is provided. Suggestions for the provider directory are to create a file name with the title "provider_dir" followed by the date of creation. How often this report is created depends on factors to yet to be determined. The EFT record file name can be set up with the provider name followed by the date, with a different extension than that of the provider report.

Input-Output Specifications:

The input and output specifications for the ChocoAn terminal will need to be supplied by the company providing the terminals for the system. The relevant input screens for this product will need to incorporate member and provider database updates, including addition, modification, and deletion of records. There is also an input to be able to select and create reports on manager request. A screen to log in to the system will also be required, but the details of product access have yet to be provided by Chocoholics Anonymous, so a simple mock-up will be supplied for now.

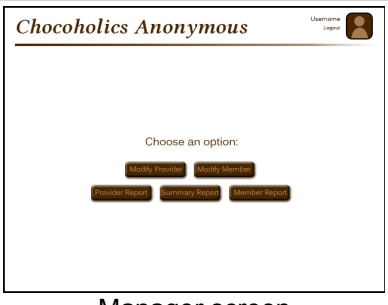
Output screens at this time are very minimal. Confirmation of modifications made to the database and EFT records will be needed. However, reports are emailed as attachments and therefore not displayed on screen. The exact format for print has not been specified either, but a sample with the required information is provided. At this time, Appendix A does not specify changes regarding the list of services on offer, or any other types of access needed within the system.

Here are the basic layouts of screens needed for the ChocoAn product.

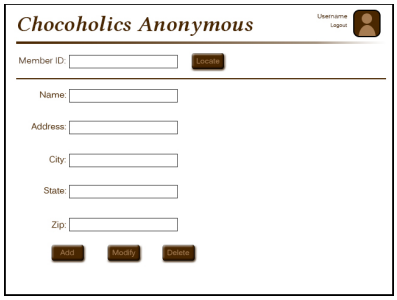
Figure 6a – Input screen layouts for the ChocoAn product. (click each for a larger version of the image from file)

Login screen

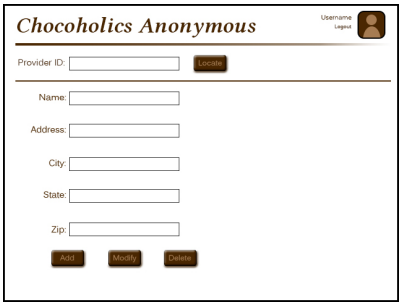
Operator screen



Manager screen

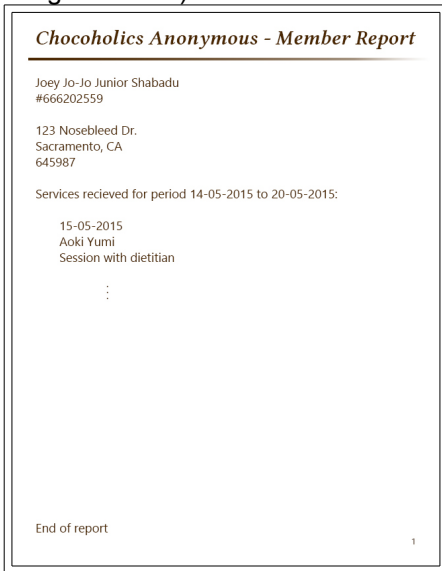


Modify member screen

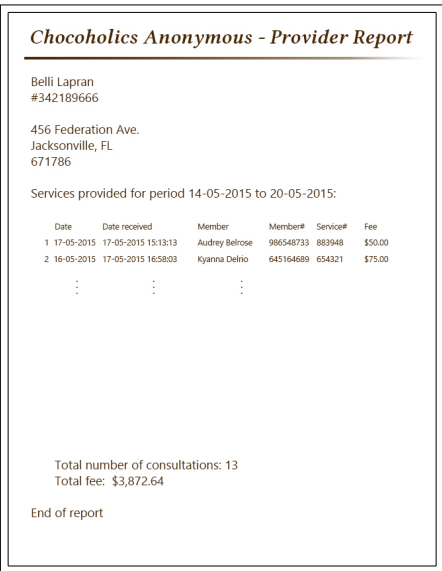


Modify provider screen

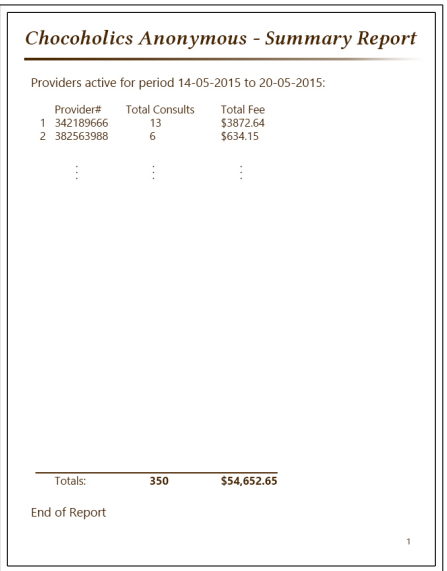
Figure 6b – Report layouts for the ChocoAn product. (each screen is hyperlinked to a larger version of the image from file)



Member report



Provider report



Summary report

Sizing:

The information to compute the size of the data needed is not fully included in Appendix A. Therefore, more discussions would have to take place with Chocoholics Anonymous to determine several values.

What can be determined from Appendix A is the number and frequency of reports. Reports are printed weekly on Fridays at midnight. There are 3 types of reports to be generated. Member reports are created for members who received ChocoAn services during the week. Provider reports are created for each provider who provides ChocoAn services during the week. One summary report is generated with information about payments to active providers and total values of consultations and payments needed for the week.

However, no data is available about the current numbers of members and providers in the system, nor is the average number of weekly reports required. This information should be obtained to be able to accurately reflect the number of reports that will need to be generated. Other information needed to be obtained from Chocoholics Anonymous includes the average number of new members and providers added to the system each week, how often the member and provider information is updated, the average number of services provided in a week, the total number of available services, and how often provider directories are requested.

Hardware Requirements:

Due to the fact that we do not have access to all the information concerning the sizing of

several system factors, we cannot make full recommendations regarding the hardware requirements of the product at this time. However, some hardware can be determined or estimated at this time.

An ink-jet printer is required to print out reports. Since the reports generated by the system are sent to recipients by email, the need to print reports should be fairly low. Therefore, we will not need a high-end, fast printer, so a relatively cheap ink-jet model should suffice for most printing purposes.

A fast and reliable network connection. The exact type of internet connection cannot be reliably determined without more accurate data. Nevertheless, a fast and reliable connection is required so that the product can reliably communicate with ChocoAn terminals at several different provider locations. To be more specific about the type of connection recommended depends not only on Chocoholics Anonymous data, but also information about the local companies provided access and the types and prices of connection services they offer.

A database with database management system are recommended to store Chocoholics Anonymous records. These will allow fast and reliable storage of information about members, providers, services, etc. Fast and reliable access is necessary for communication with ChocoAn terminals, as well as for creating reports and maintaining up to date records. Again, the specifics of the exact product cannot be recommended at this time without more accurate information concerning the number of records and how often they are accessed in the system.

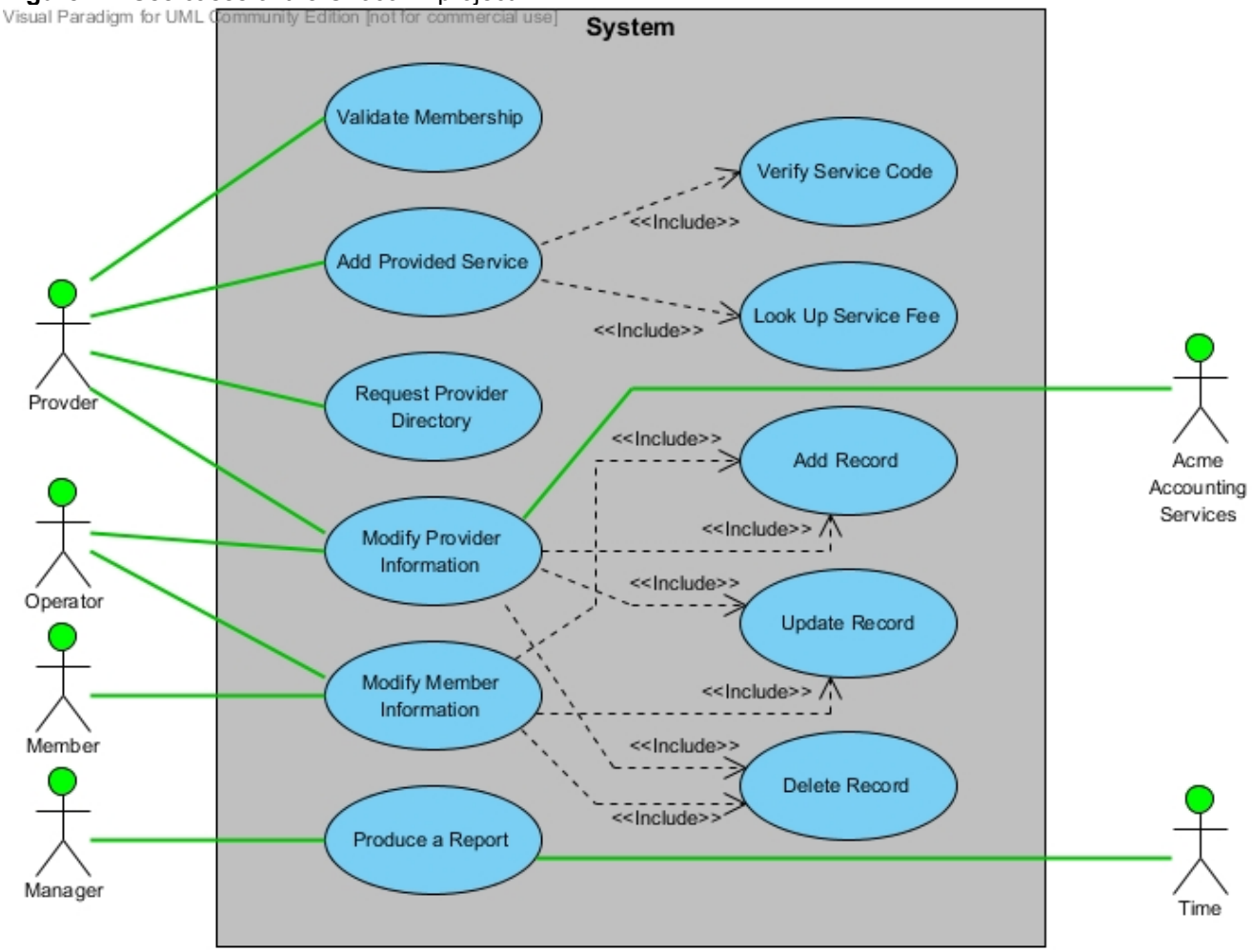
At least one workstation is required for operators and managers to access the system. The exact specifications for this workstation cannot be determined at this time without more information. Additionally, the number of workstations needed cannot be determined without the missing information above, and we would also need to know the number of employees and managers who need system access at Chocoholics Anonymous.

Problem 13.22

Perform the analysis workflow of the Chocoholics Anonymous product described in Appendix A.

The first step of the analysis workflow is to perform functional modelling. We model the use cases for the product with specific scenarios, beginning with the use case diagram obtained from the requirements workflow seen in Figure 1.

Figure 1 – Use cases of the ChocoAn project



From these use cases we can develop the following scenarios, seen in Figure 2.1 – 2.11d.

Figure 2.1 – An extended scenario to validate a membership

A ChocoAn provider wants to validate a ChocoAn member using the ChocoAn terminal.

1. The provider swipes the member card or keys in the member number
2. The system uses the member number to retrieve the member status stored in the system.
3. The system displays the member status on the terminal screen.

Possible alternatives

A) A hardware or network failure disrupts communication between the terminal and the system.
B) The terminal cannot read the card.

C) The provider enters an invalid member number.
D) The system cannot find a record corresponding to the member number.

Figure 2.2 – An extended scenario to request a Provider Directory

A ChocoAn provider wants to request a copy of the Provider Directory from the ChocoAn system.

1. The provider requests a new Provider Directory from the system.
2. The system creates a list of all services as the Provider Directory.
3. The system sends the Provider Directory to the provider's email address as an attachment.

Possible alternatives

A) A hardware or network failure disrupts communication between the terminal and the system.
B) There is no email address associated with the provider.

Figure 2.3 – An extended scenario to add a provided service to the system.

A ChocoAn provider has provided a service to a ChocoAn member and wants to enter the details of the provided service into the ChocoAn system.

1. The provider enables the *validate membership* use case.
2. The provider keys in the date the service was provided.
3. The provider *verify service code* use case.
4. The provider optionally enters additional comments for the provided service.
5. The system writes the provided service record to disk.
6. The system enables the *look up service fee* use case.
7. The provider records the service details on a paper form.

Possible alternatives

A) There are problems associated with the *validate membership*, *verify service code*, or *look up service fee* use cases.
B) The provider enters an incorrect date.
C) An error prevents the system from writing the record to disk.
D) The provider is unable to record a separate copy of the service details.

Figure 2.4 – An extended scenario to verify a service code

A ChocoAn provider wants to enter a service code for verification to continue the *add provided service* use case.

1. The provider keys in a service code corresponding to the service provided.
2. The system uses the provided service code to look up the associated service name in the records.
3. The system displays the service name on the terminal for verification.
4. The provider visually verifies that the service name is correct.

Possible alternatives

A) A hardware or network failure disrupts communication between the terminal and the system.
B) The provider enters an incorrect service code.
C) The system cannot find a record associated with the service code.
D) The system cannot find an associated service name for a service code.
E) The name associated with a service code is incorrect.
F) The provider fails to verify the service name.

Figure 2.5 – An extended scenario to look up a service fee.

The ChocoAn system must look up and display a service fee on the ChocoAn terminal to continue the *add provided service* use case.

1. The system uses the provided service code to look up the associated service fee.
2. The system displays the service fee on the terminal screen.

Possible alternatives

A) A hardware or network failure disrupts communication between the terminal and the system.
B) There is no fee associated with the service code.
C) The fee associated with a service code is not correct.

Figure 2.6 – An extended scenario to modify provider information

A ChocoAn operator wishes to modify the information of a ChocoAn provider in the ChocoAn system.

3. The operator selects the modify provider information option.
4. The operator chooses to:
    1. add a new provider to the system, enabling the *add record* use case.
    2. modify an existing provider in the system, enabling the *modify record* use case.
    3. delete an existing provider in the system, enabling the *delete record* use case.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The operator selects the wrong option.
C) There are problems associated with the *add record*, *modify record*, or *delete record* use cases.

Figure 2.7 – An extended scenario to modify member information

A ChocoAn operator wishes to modify the information of a ChocoAn member in the ChocoAn system.

1. The operator selects the modify member information option.
2. The operator chooses to:
    1. add a new member to the system, enabling the *add record* use case.
    2. modify an existing member in the system, enabling the *modify record* use case.
    3. delete an existing member in the system, enabling the *delete record* use case.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The operator selects the wrong option.
C) There are problems associated with the *add record*, *modify record*, or *delete record* use cases.

Figure 2.8 – An extended scenario to add a record to the system

A ChocoAn operator wants to add a new record to the ChocoAn system.

1. The system displays the add a new record screen for either a member or a provider, depending on the use case used to enable the current use case.
2. The operator enters in the necessary details in the provided fields.
3. The operator selects to add a new record.
4. The system writes the new record to disk.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The operator enters incorrect information.
C) The operator fails to enter some or all required information.
D) The operator fails to select to add the record.

Figure 2.9a – An extended scenario to update an existing record to the system

A ChocoAn operator wants to modify an existing record to the ChocoAn system.

1. The system displays the screen to enter a number for an existing record for either a member or a provider, depending on the use case used to enable the current use case.
2. The operator enters a member or provider number.
3. The system looks up the associated record for either a member or a provider,.
4. The system displays the associated information on screen in various fields.
5. The operator selects fields to modify by changing the information.
6. The operator selects to update the record.
7. The system updates the information of the associated record on disk.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The operator enters an incorrect number to look up the record.
C) The system cannot find a record associated with the number entered.
D) The operator enters incorrect information.
E) The operator fails to enter some or all required information.
F) The operator fails to select to update the record.

Figure 2.9b – Another extended scenario to modify an existing record to the system

It is 9 p.m. and the Acme Accounting (AA) system must update the status of existing ChocoAn (CA) members.

1. The AA system sends information regarding the status of each member.
2. The AA system process the information for each member, updating the status of each member provided.
3. The CA system sends a message that the update process is complete.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) There is missing information provided by the AA system.
C) The CA system cannot locate a member identified by the AA system.

Figure 2.10 – An extended scenario to delete an existing record from the system

A ChocoAn operator wants to modify an existing record to the ChocoAn system.

1. The system displays the screen to enter a number for an existing record for either a member or a provider, depending on the use case used to enable the current use case.
2. The system looks up the associated record for either a member or a provider.
3. The system displays the associated information on screen.
4. The operator selects to delete the record.
5. The system deletes the associated record from disk.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The operator enters an incorrect number to look up the record.
C) The system cannot find a record associated with the number entered.
D) The operator fails to select to delete the record.

Figure 2.11a – An extended scenario to produce a report

It is Friday at midnight and the ChocoAn system must produce a series of reports.

1. The system looks up all services provided for the week.
2. The system looks up all members who received services for the week.
3. The system creates a report for each member identified with an ordered list of services that member received during the week.
4. The system writes a copy of each report to disk.
5. The system looks up each provider that provided services during the week.
6. The system calculates the total number of services delivered and total fees for each provider identified.
7. The system creates a report for each provider identified with a list of services delivered by that provider during the week, the total number of consultations, and the total fee to be paid.
8. The system writes a copy of each provider report to disk.
9. The system emails a copy of an associated report to each provider identified.
10. The system writes an EFT record to disk for each provider identified, containing the total fee to be paid.
11. The system emails a copy of an associated report to each provider identified.
12. The system calculates the sum number of services delivered by all identified providers and the sum of all fees to be paid to all providers.
13. The system creates a report including a list of all identified providers with total fees and services delivered, the total number of services delivered, and the total weekly fee to be paid.
14. The system writes a copy of the report to disk.
15. The system emails a copy of the report to the ChocoAn manager of accounts payable.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The system cannot find an associated email for a member, provider, or the manager of accounts payable.
C) A provider has failed to enter a provided service for the week in question.
D) There is missing information from a record in the system needed to produce a report.

Figure 2.11b – Another extended scenario to produce a report

A ChocoAn manager wishes to produce a member report from the ChocoAn system.

1. The manager selects the option to produce a member report for a specific member over a given week range.
2. The system looks up the member information associated with the entered number.
3. The system looks up all services received by the associated member for the entered week.
4. The system creates a report with the member information and all identified services received.
5. The system displays the report.
6. The manager chooses to email the report.
7. The system emails the report.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The manager selects the wrong report option.
C) The manager enters an incorrect member number.
D) The system cannot find an associated record for the entered member number.
E) The manager inputs the wrong week range for the report.
F) The system cannot find any services received by a member for a given week range.
G) The manager chooses not to email the report.
H) The system cannot find an associated email for a member.
I) A provider has failed to enter a provided service for the week in question.
J) There is missing information from a record in the system needed to produce a report.

Figure 2.11c – Another extended scenario to produce a report

A ChocoAn manager wishes to produce a provider report from the ChocoAn system.

1. The manager selects the option to produce a provider report for a specific provider over a given week range.
2. The system looks up the provider information associated with the entered number.
3. The system looks up all services rendered by the associated provider for the entered week.
4. The system calculates the total number of services delivered and total fee for the provider in the given week.
5. The system creates a report with the provider information, all identified services received, the total number of services rendered, and the weekly total fee.
6. The system displays the report.
7. The manager chooses to email the report.
8. The system emails the report.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The manager selects the wrong report option.
C) The manager enters an incorrect provider number.
D) The system cannot find an associated record for the entered provider number.
E) The manager inputs the wrong week range for the report.
F) The system cannot find any services received by a provider for a given week range.
G) The manager chooses not to email the report.
H) The system cannot find an associated email for a provider.
I) A provider has failed to enter a provided service for the week in question.
J) There is missing information from a record in the system needed to produce a report.

Figure 2.11d – Another extended scenario to produce a report

A ChocoAn manager wishes to produce a summary report from the ChocoAn system.

1. The manager selects the option to produce a summary report for a given week range.
2. The system looks up all services rendered for the entered week.
3. The system looks up the provider information of each provider delivering services in the requested week range.
4. The system calculates the total fee and total number of services delivered by each provider for the

given week as well as the sum of all services and fees.
5. The system creates a report with the identified providers with their total services and fees, the total number of all services, and the total sum of all fees.
6. The system displays report.
7. The manager chooses to email the report.

Possible alternatives

A) A hardware or network failure prevents the use case from executing.
B) The manager selects the wrong report option.
C) The manager inputs the wrong week range for the report.
D) The system cannot find the email address for the manager of accounts payable.
E) A provider has failed to enter a provided service for the week in question.
F) There is missing information from a record in the system needed to produce a report.

The next step in the analysis workflow is class modelling. This step begins by extracting the entity classes. This can be done with the two-stage noun extraction process. Stage 1 describes the product in a single paragraph:

> ChocoAn providers add services to they ChocoAn system that they provide to ChocoAn members. The swipe a membership card or key in the member number for validation using a terminal, then enter the service code, the date of the provided service, and any additional comments. The system then writes a record of the service to disk. At any time a ChocoAn manager can request a member report, provider report, or summary report from the system. These reports are also created by the system every Friday at midnight, when the system writes an EFT record of weekly payments to providers for that week. During the day, the system is run in interactive mode to allow ChocoAn operators to modify member and provider records by adding, updating, or deleting records in the system. Member status is updated daily by Acme Accounting Services in correspondence with member payments.

The next stage in the process is to identify the nouns in the paragraph and use them as candidate entity classes. The nouns have been highlighted in the following paragraph:

> ChocoAn providers add services to they ChocoAn system that they provide to ChocoAn members. The swipe a membership card or key in the member number for validation using a terminal, then enter the service code, the date of the provided service, and any additional comments. The system then writes a record of the service to disk. At any time a ChocoAn manager can request a member report, provider report, or summary report from the system. These reports are also created by the system every Friday at midnight, when the system writes an EFT record of weekly payments to providers for that week. During the day, the system is run in interactive mode to allow ChocoAn operators to modify member and provider records by adding, updating, or deleting records in the system. Member status is updated daily by Acme Accounting Services in correspondence with member payments.

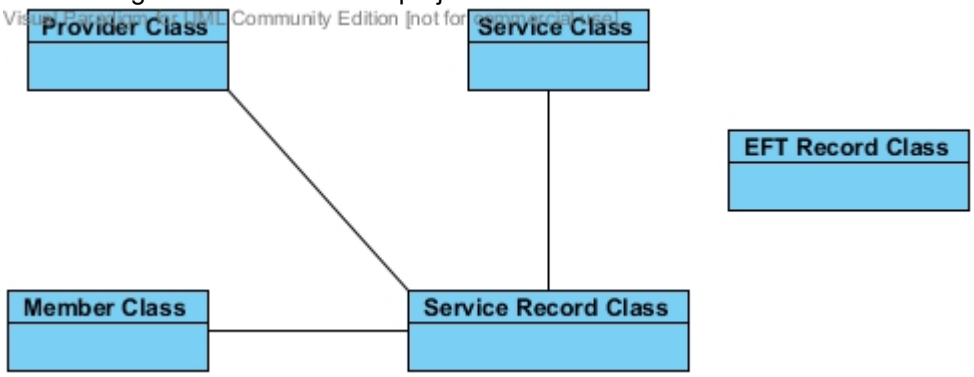
From the identified nouns above, the following table categorizes the identified nouns into classes to select the candidate entity classes. Nouns outside the problem domain are eliminated and abstract nouns are identified as possible class attributes. The remaining classes are selected as candidate entity classes, as seen in Figure 3.

Analyzing the candidate classes, the noun 'report' is not a long lived item, and is a likely candidate for a boundary class. Another possible boundary class is the noun 'terminal.' The noun 'system' can also be removed as it is the product itself, modelled by the class diagram. Using the remaining candidates, we can create the following class diagram seen in Figure 4.

Figure 3 – Table of identified nouns from the description of the ChocoAn product

Nouns outside the domain	Abstract nouns	Candidate entity classes
• (membership) card • disk • manager • payment(s) • operator(s) • Acme Accounting Services	• (member) number • (service) code • date • time • week • Friday • midnight • day • mode • status	• provider(s) • service(s) • member(s) • system • terminal • record • (member, provider, summary) report(s)

Figure 4 – Initial class diagram for the ChocoAn project.



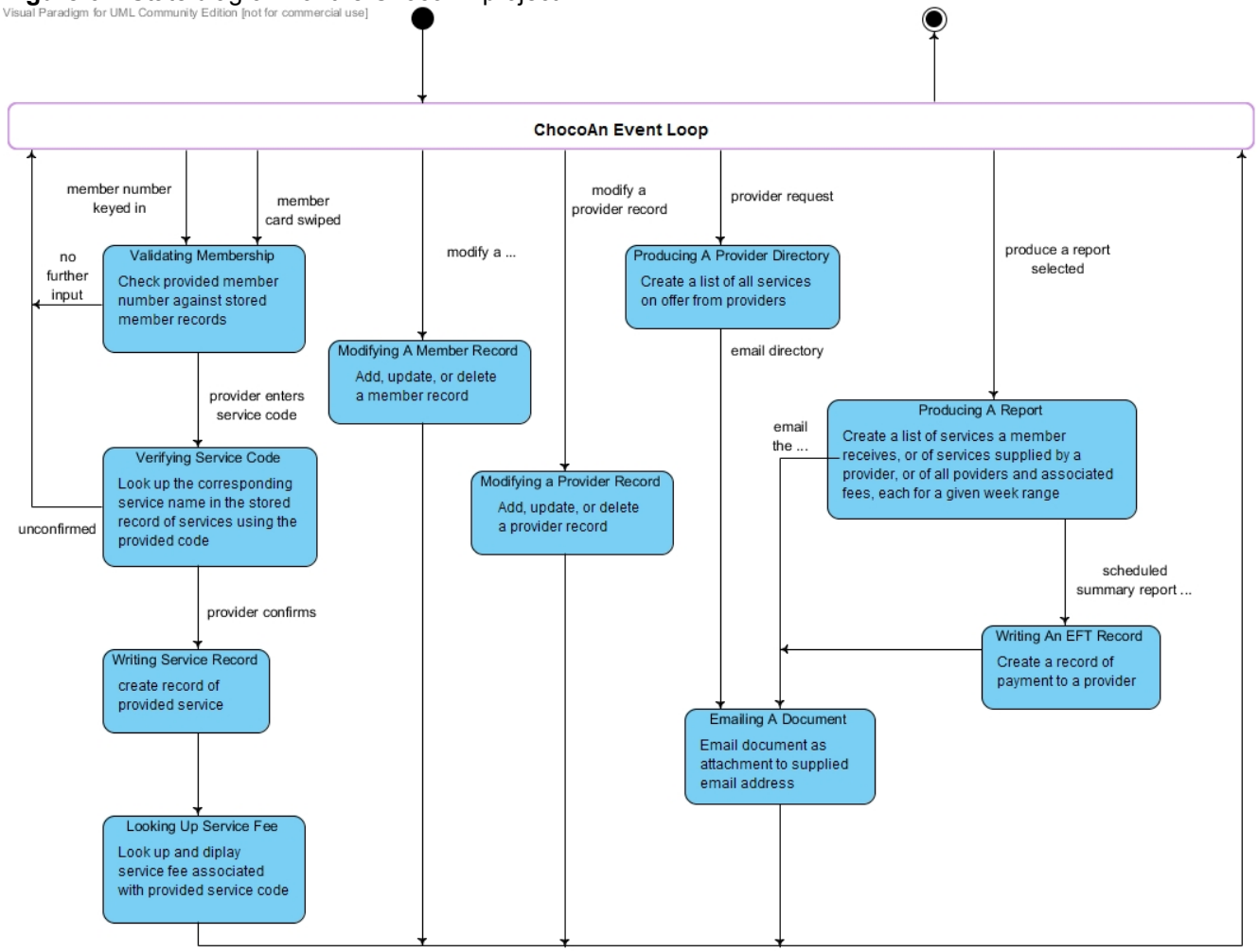
Now that we have extracted the entity classes, the next step of the analysis workflow is to apply dynamic modelling to determine the operations of each entity class and sub-classes. This is modelled using the state chart seen in Figure 6.

Moving on, we need to extract the boundary classes. These classes are associated with input and output of the system, modelling the interaction between actors and the product. Figure 5 provides a list of boundary classes.

Figure 5 – Boundary classes of the ChocoAn project.

Member Report Class
Provider Report Class
Summary Report Class
Terminal Interface Class
Employee Interface Class

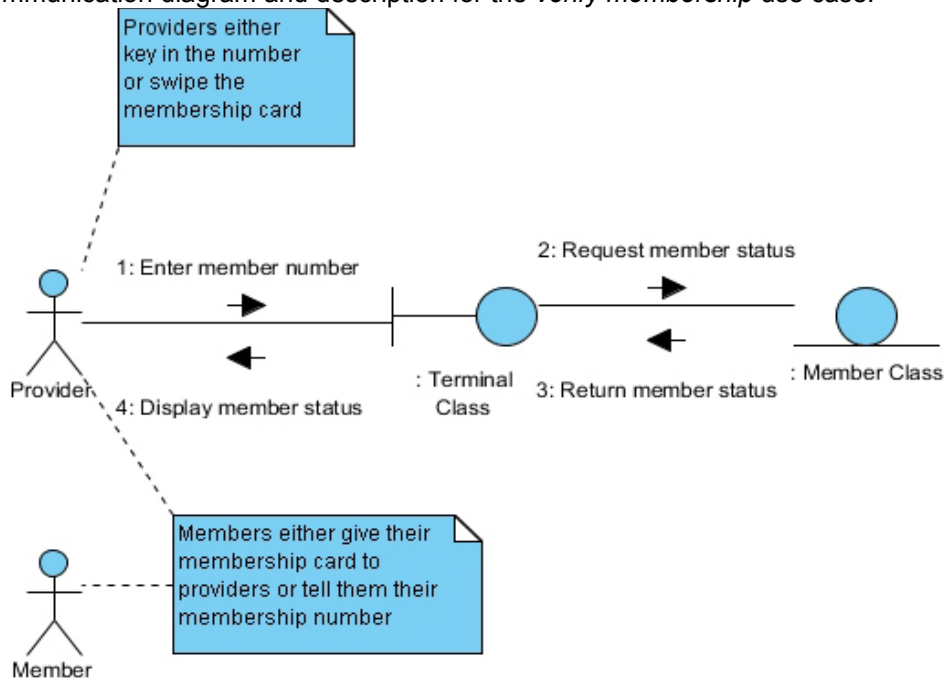
Figure 6 – State diagram for the ChocoAn project.



The final classes to consider are the control classes. These classes control complex computations and algorithms performed by the system. However, the ChocoAn product only has trivial calculations involved. The calculations for provider payments and overall weekly payment are simple sums, as is the calculation for total consultations within a week. Therefore, there are no control classes to extract and model. Nevertheless, we can still identify control classes to handle certain functions within the system. For managing member and provider records, we can use a **Manage Record Class**. Additionally, the reports gather information from different classes together, so we can use the **Report Generator** class to handle this function.

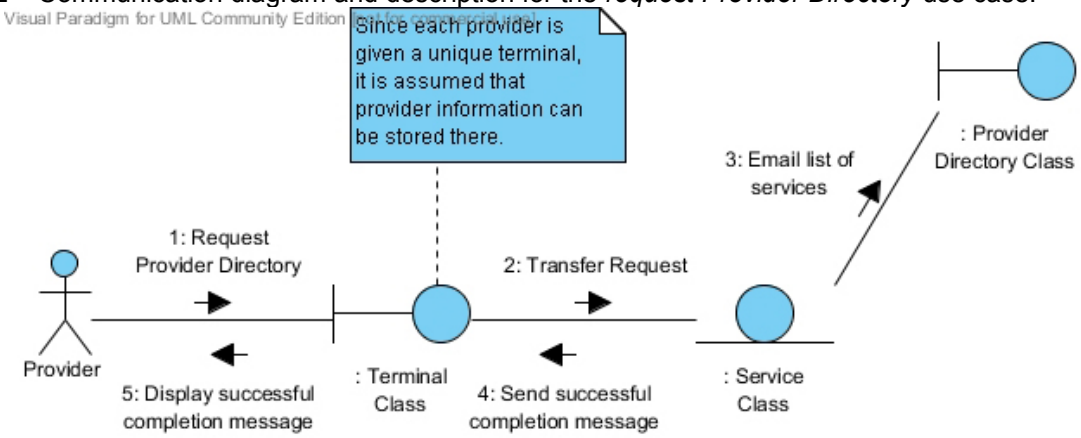
The final step of the analysis workflow is use-case realization. Since we are concentrating on extracting the classes, communication diagrams for most of the use cases will be generated, each with a written description. Equivalent sequence diagrams are not generated for most use cases because the flow of information is relatively clear from the communication diagrams. However, in cases where it is not clear, a sequence diagram has been provided instead of the communication diagram. For the *add provided service use case*, both are used. These diagrams appear in Figures 7.1 – 7.16.

Figure 7.1 – Communication diagram and description for the *verify membership* use case.



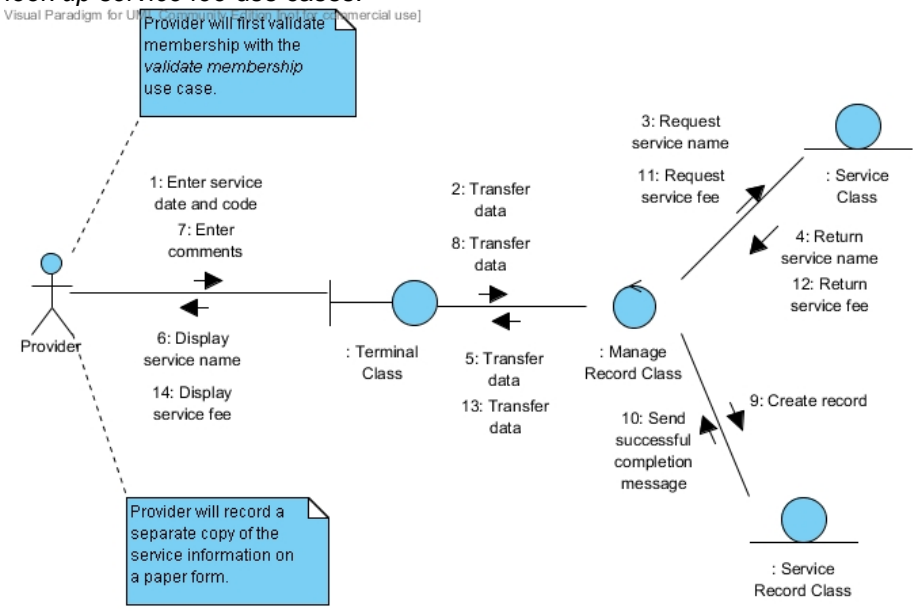
A ChocoAn provider enters a member number for verification (1). The system retrieves the status of the member (2, 3) and displays it to the provider (4).

Figure 7.2 – Communication diagram and description for the *request Provider Directory* use case.



A ChocoAn provider requests a Provider Directory (1, 2). The system emails the list (3) and confirms (4, 5)

Figure 7.3 – Communication diagram and description for the *add provided service* use case, including the *verify service code* and *look up service fee* use cases.



A ChocoAn provider enters a service date and code for a service provided (1, 2). The system looks up the name of the service (3, 4) and displays it (5, 6). The provider adds comments and verifies (7, 8). The system creates a record (9, 10), then looks up the service fee (11, 12) and displays it (13, 14).

Figure 7.3a – Sequence diagram and description for the *add provided service* use case, including the *verify service code* and *look up service fee* use cases.

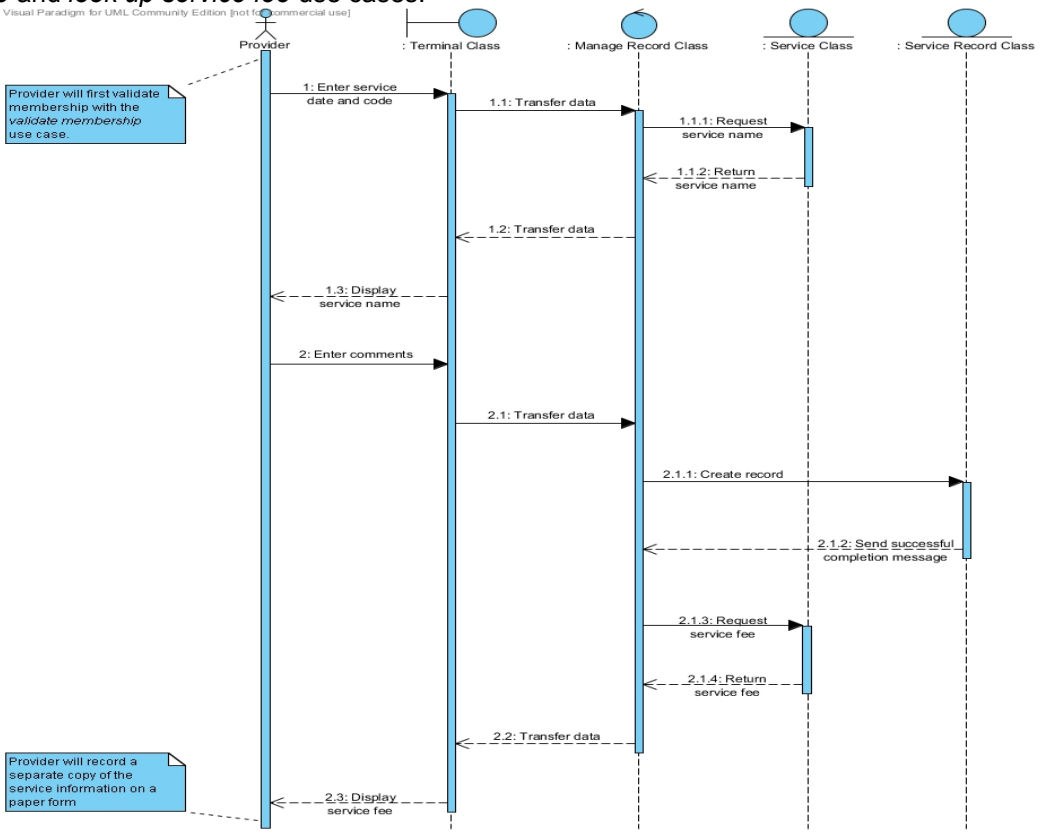
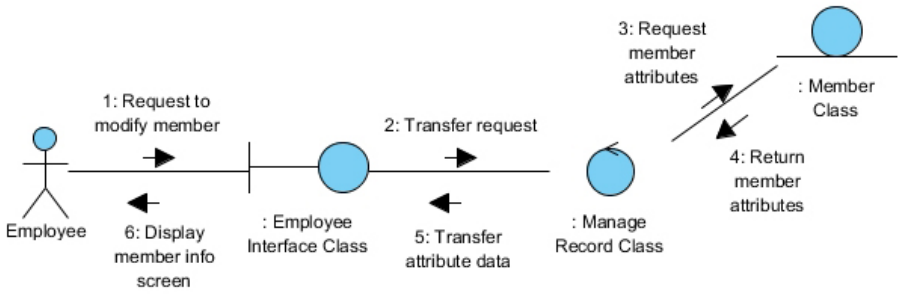
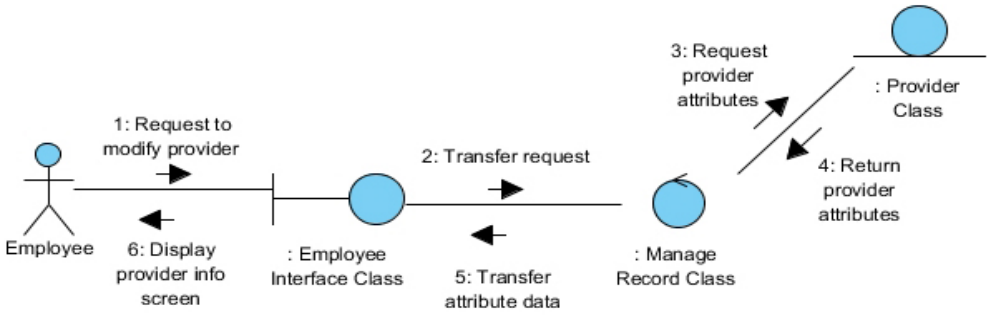


Figure 7.4 – Communication diagram and description for the *modify member information* use case.



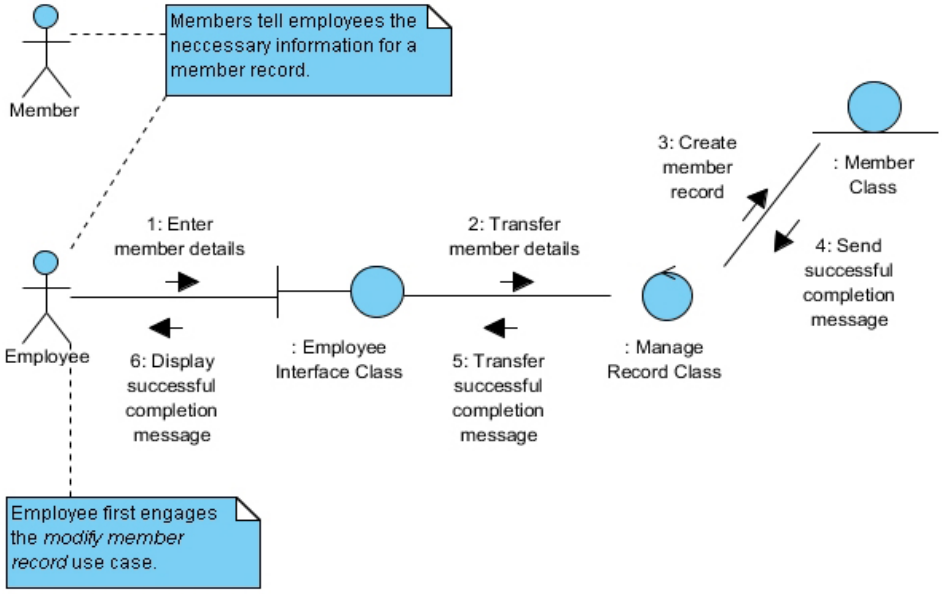
A ChocoAn employee requests to modify a member in the system (1, 2). The system looks up the list of member attributes (3) and displays them (4 - 6).

Figure 7.5 – Communication diagram and description for the *modify provider information* use case.



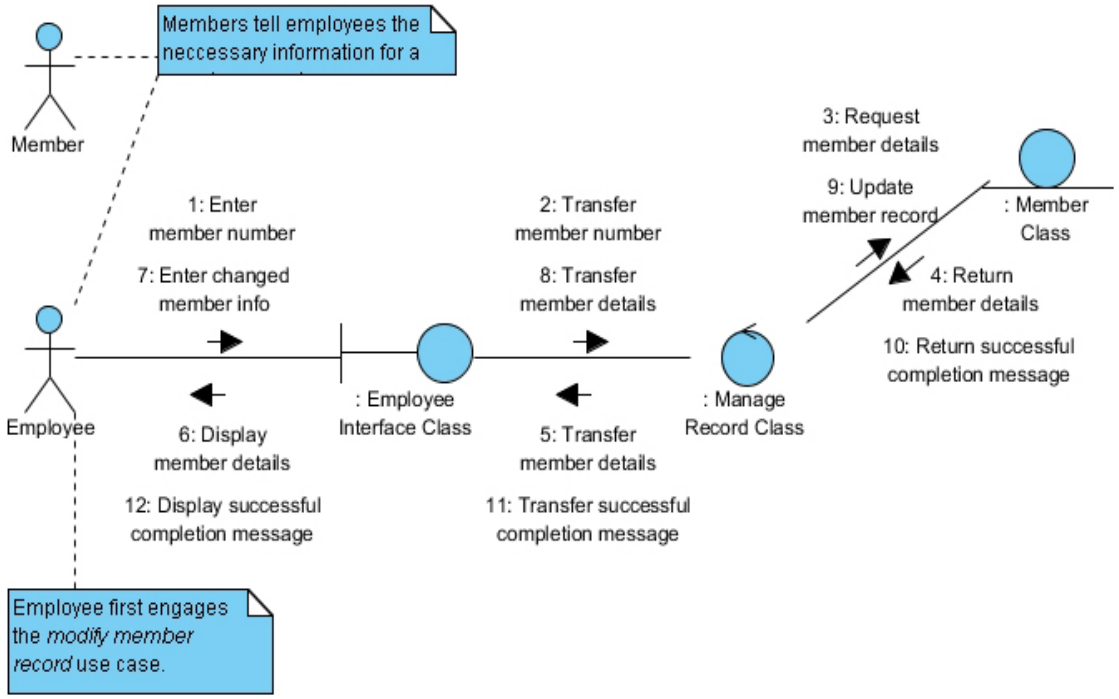
A ChocoAn employee requests to modify a member in the system (1, 2). The system looks up the list of provider attributes (3, 4) and displays them (5, 6).

Figure 7.6 – Communication diagram and description for the *add a record* use case for a member record.



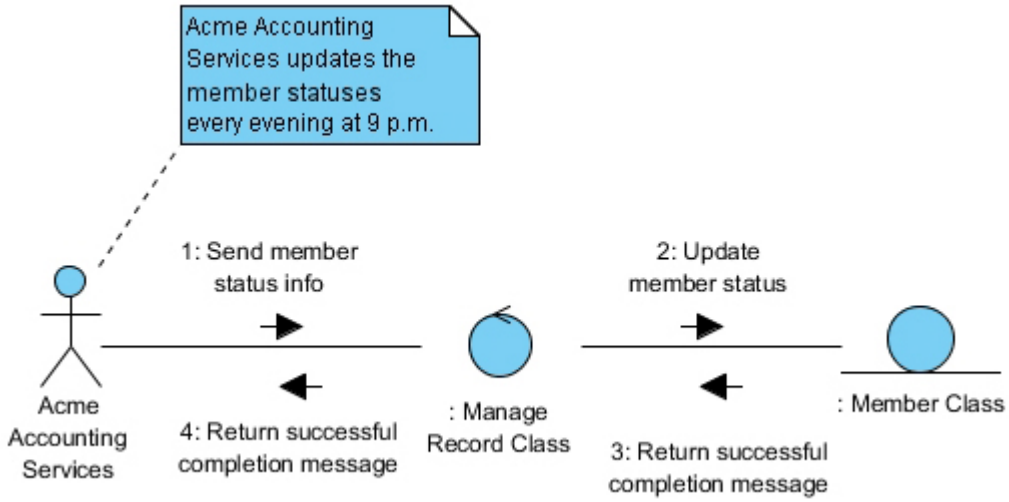
A ChocoAn employee enters the details of a new member (1, 2) and the system creates the record and confirms (3 – 6).

Figure 7.7 – Communication diagram and description for the *update a record* use case for a member record.



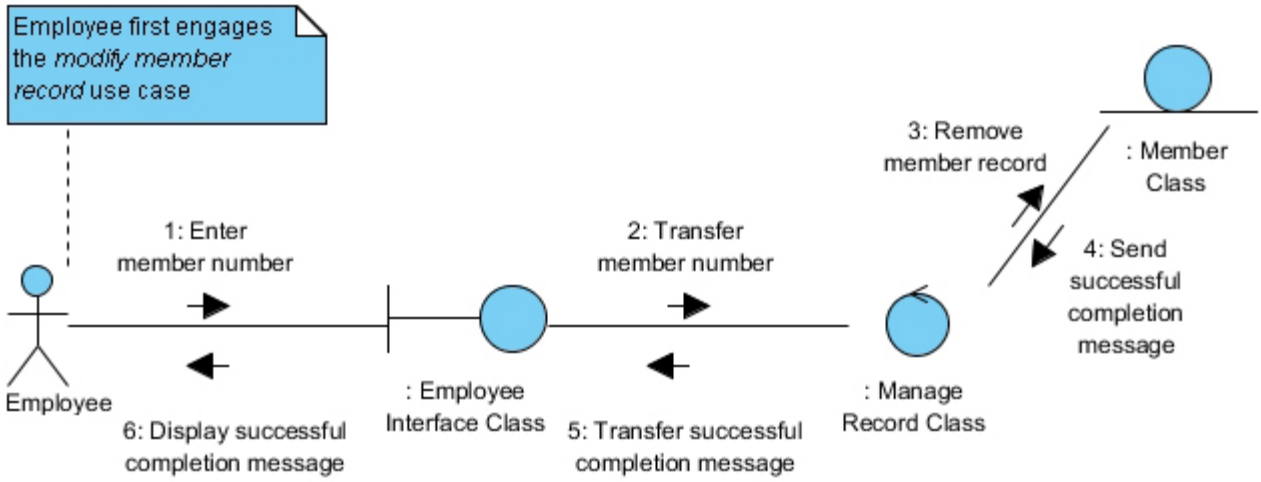
A ChocoAn employee requests to update current member information by entering an existing member number (1, 2). The system looks up the record and displays it (3-6). The employee changes the necessary details (7, 8), and the system updates the record (9) and confirms (10 – 12).

Figure 7.8 – Communication diagram and description for the *update a record* use case for a member record updated by Acme Accounting Services.



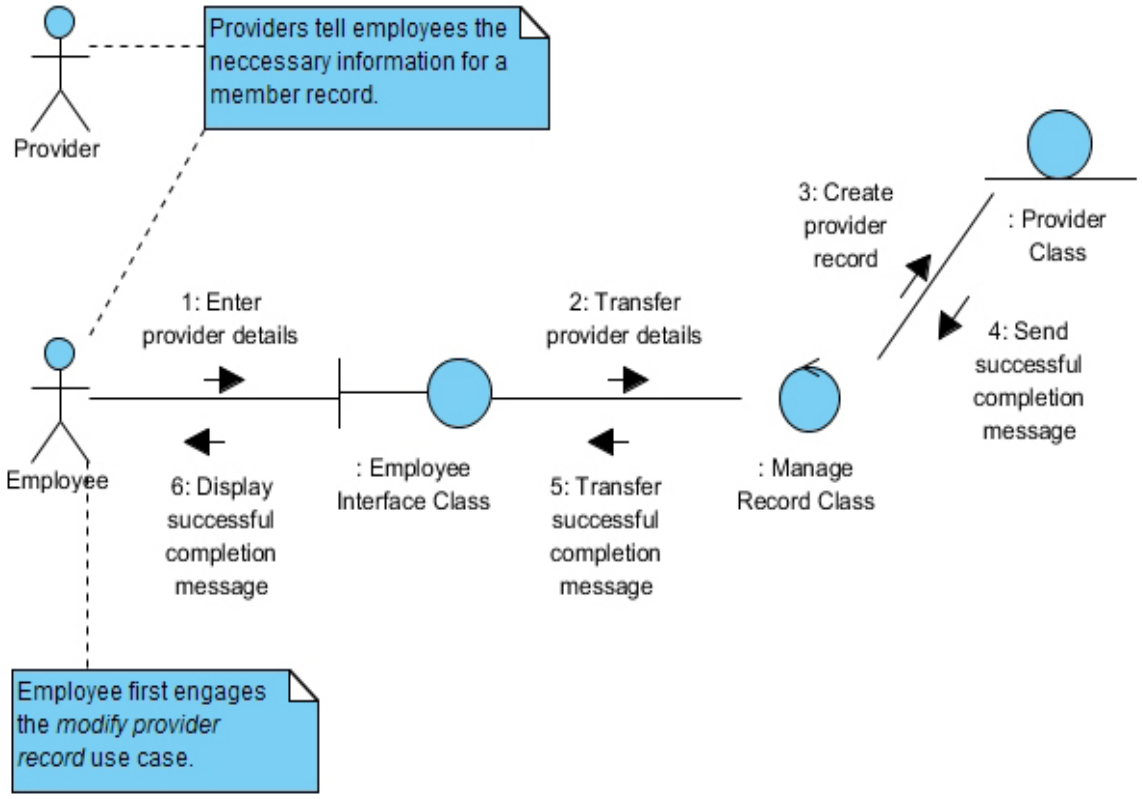
The Acme Accounting Services system sends an updated list of member statuses every evening at 9 p.m. (1). The system updates each member record (2) and confirms the update with the Acme system (3, 4).

Figure 7.9 – Communication diagram and description for the *delete a record* use case for a member record.



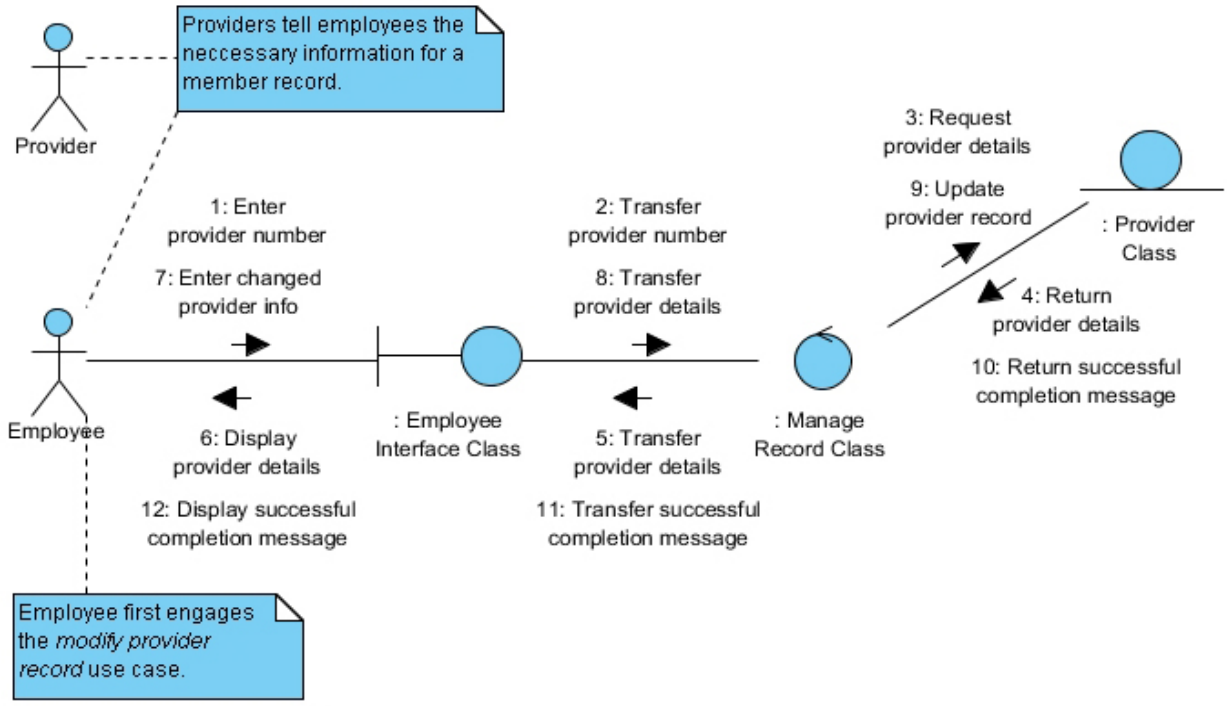
A ChocoAn employee enters an existing member number in order to delete the record from the system (1, 2). The system deletes the record (3) and confirms with the employee (4 – 6).

Figure 7.10 – Communication diagram and description for the *add a record* use case for a provider record.



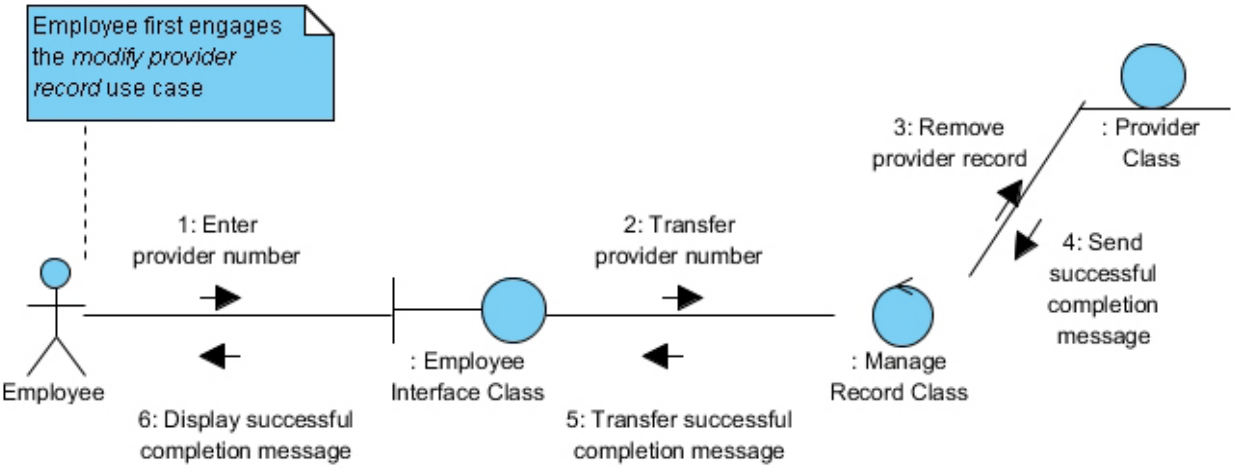
A ChocoAn employee enters the details of a new member (1, 2) and the system creates the record and confirms (3 – 6).

Figure 7.11 – Communication diagram and description for the *update a record* use case for a provider record.



A ChocoAn employee requests to update current provider information by entering an existing provider number (1, 2). The system looks up the record and displays it (3-6). The employee changes the necessary details (7, 8), and the system updates the record (9) and confirms (10 – 12).

Figure 7.12 – Communication diagram and description for the *delete a record* use case for a provider record.



A ChocoAn employee enters an existing provider number in order to delete the record from the system (1, 2). The system deletes the record (3) and confirms with the employee (4 – 6).

Figure 7.13 – Sequence diagram and description for the *produce a report* use case for a member report.

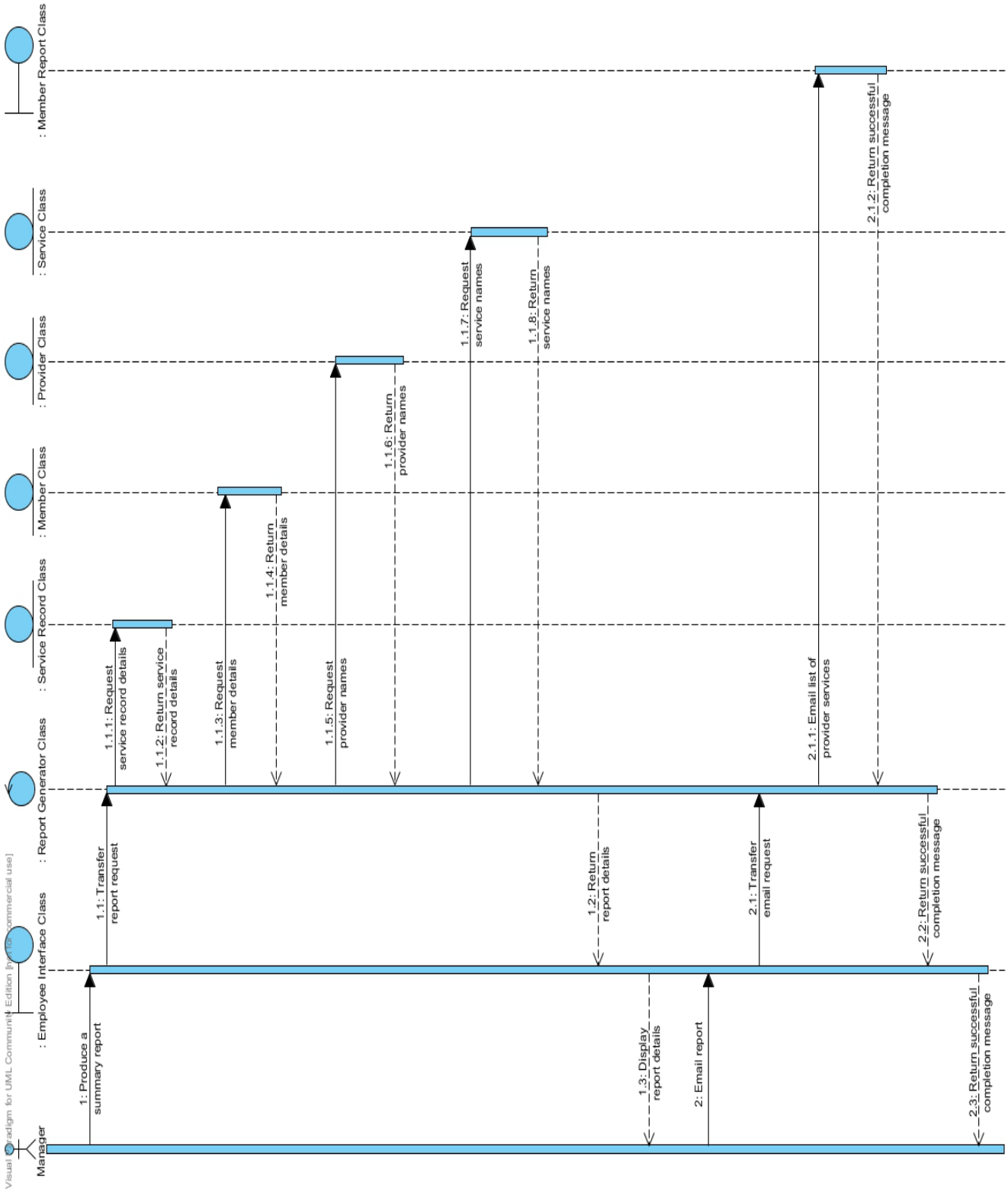


Figure 7.14 – Sequence diagram and description for the *produce a report* use case for a provider report.

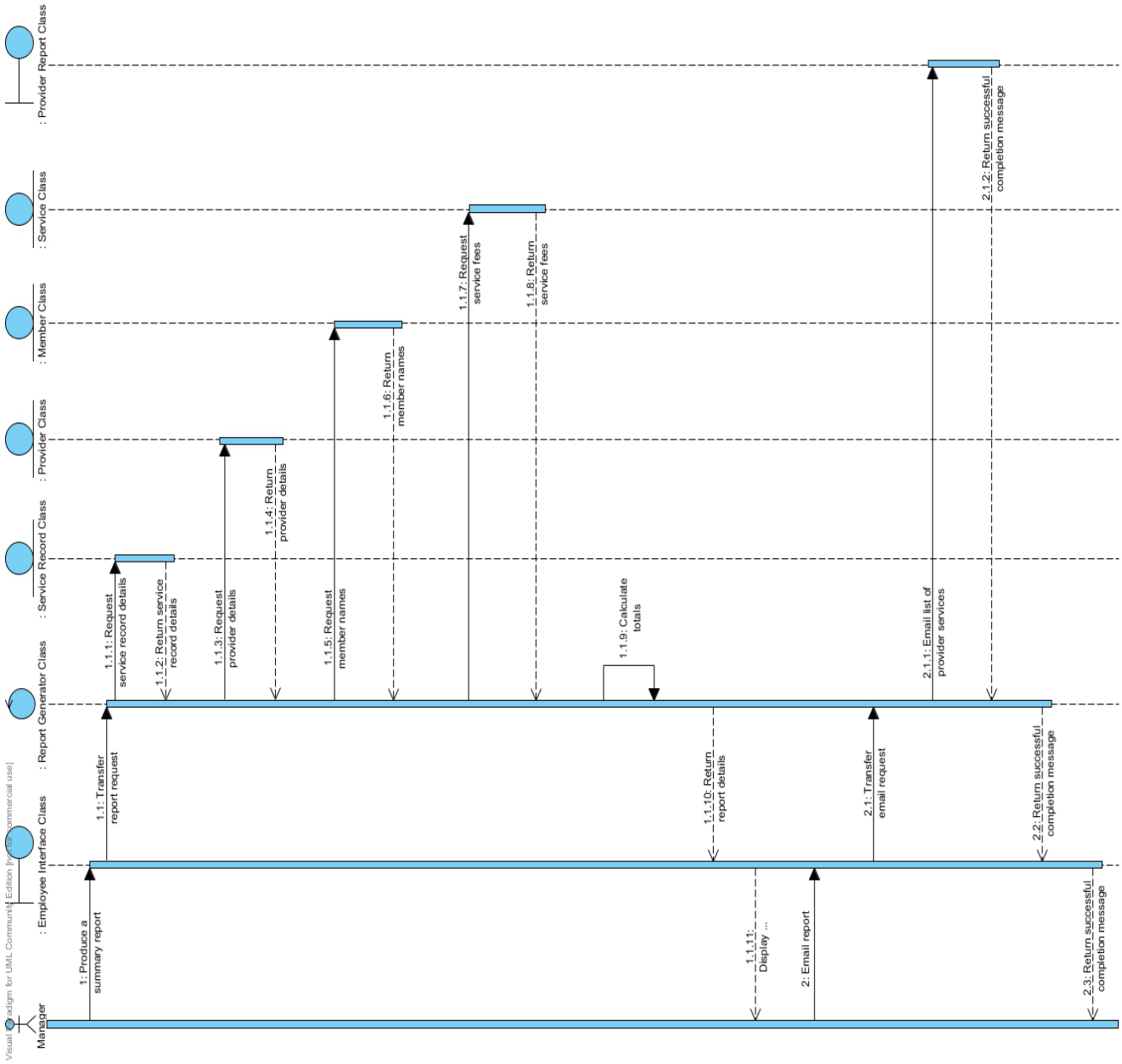


Figure 7.15 – Sequence diagram and description for the *produce a report* use case for a summary report.

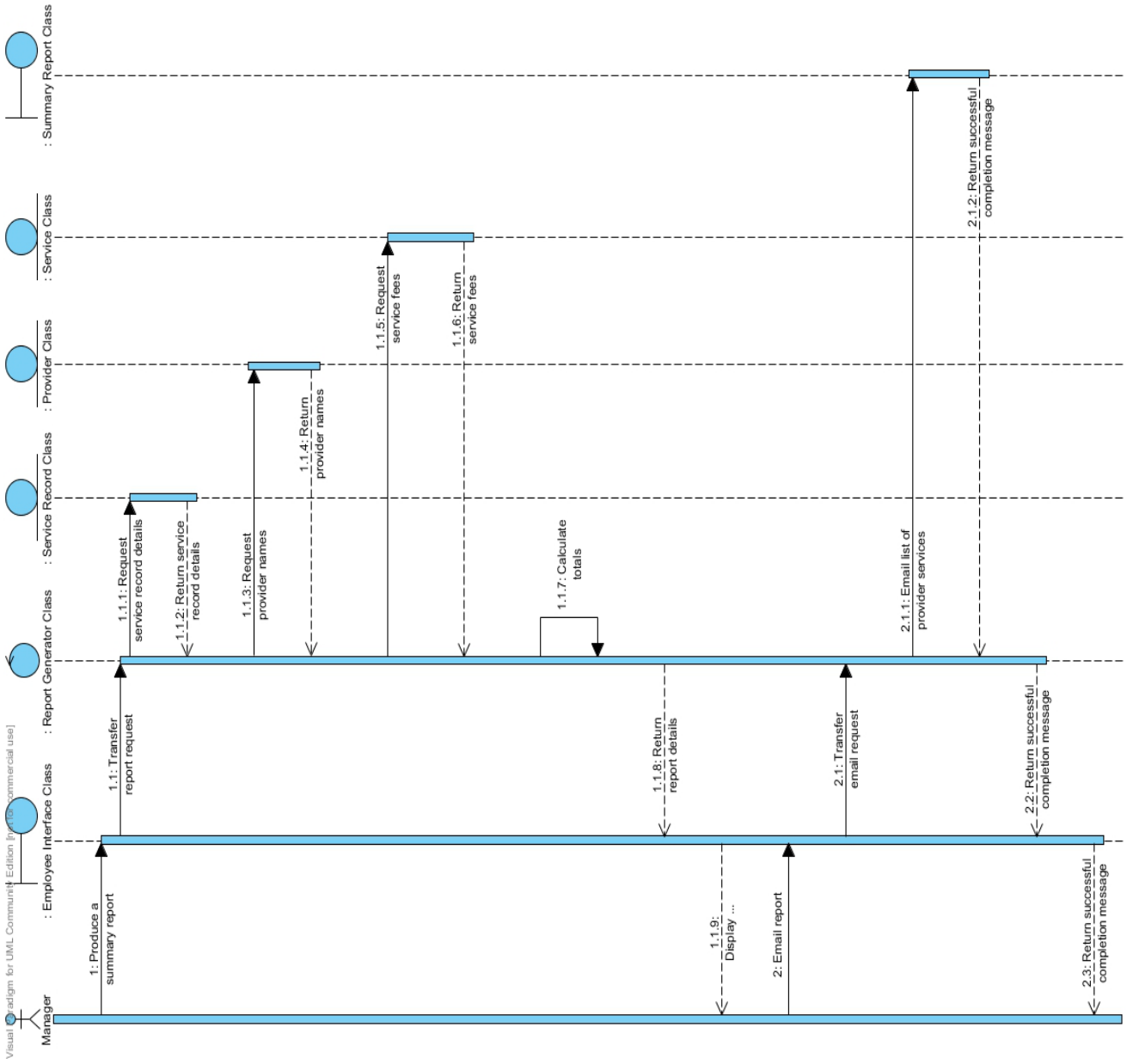
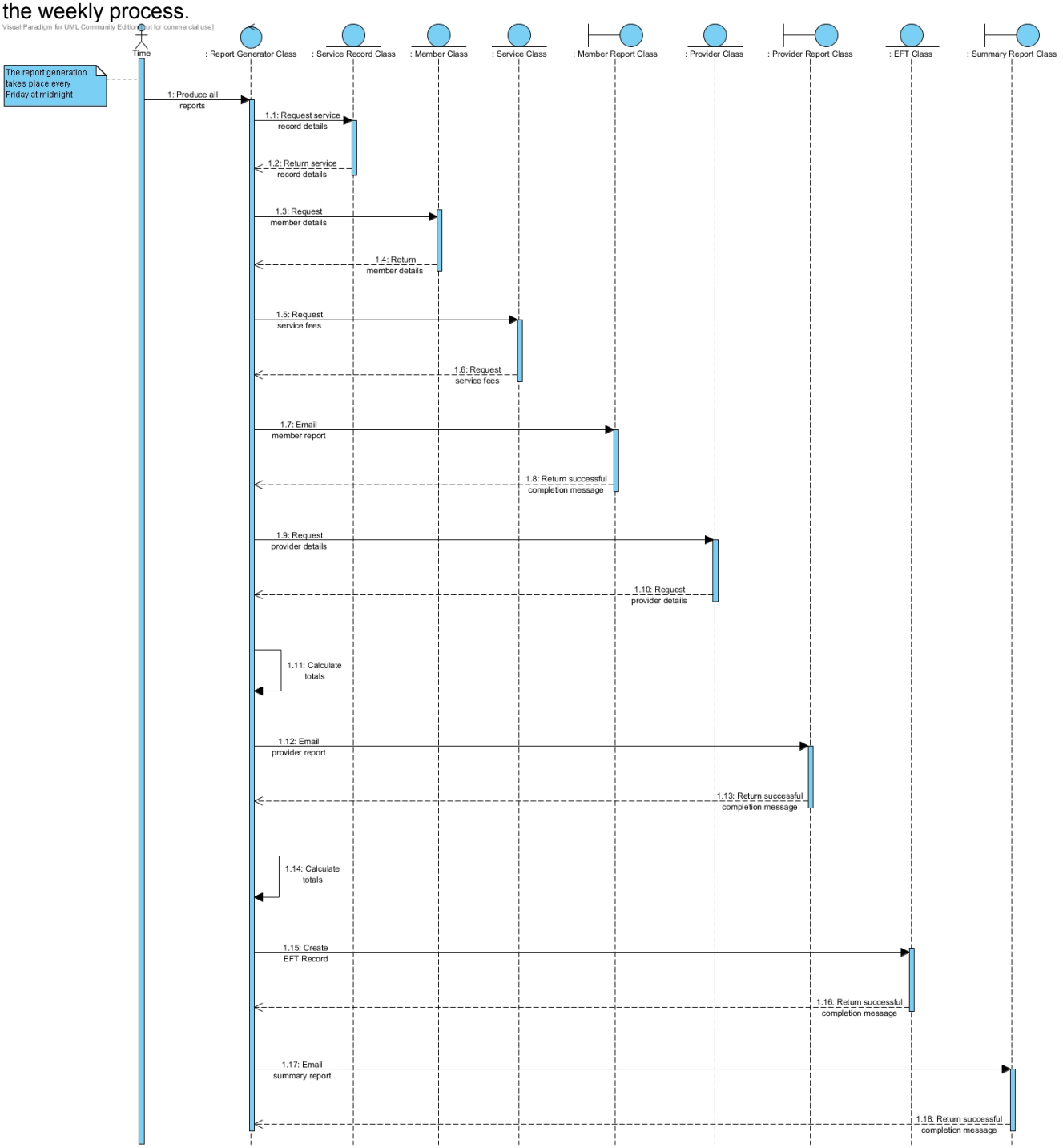


Figure 7.16 – Sequence diagram and description for the *produce a report* use case for all reports generated by the weekly process.



With all diagrams created, we can increment the diagrams to ensure that they are all accurate. We can update the use case diagram (Figure 8) to combine some of the use cases. We can also update the class diagram (Figure 10), as well as create a use case diagram with stereotypes (Figure 9) using the newly defined classes.

Figure 8 – The next iteration of the use case diagram.

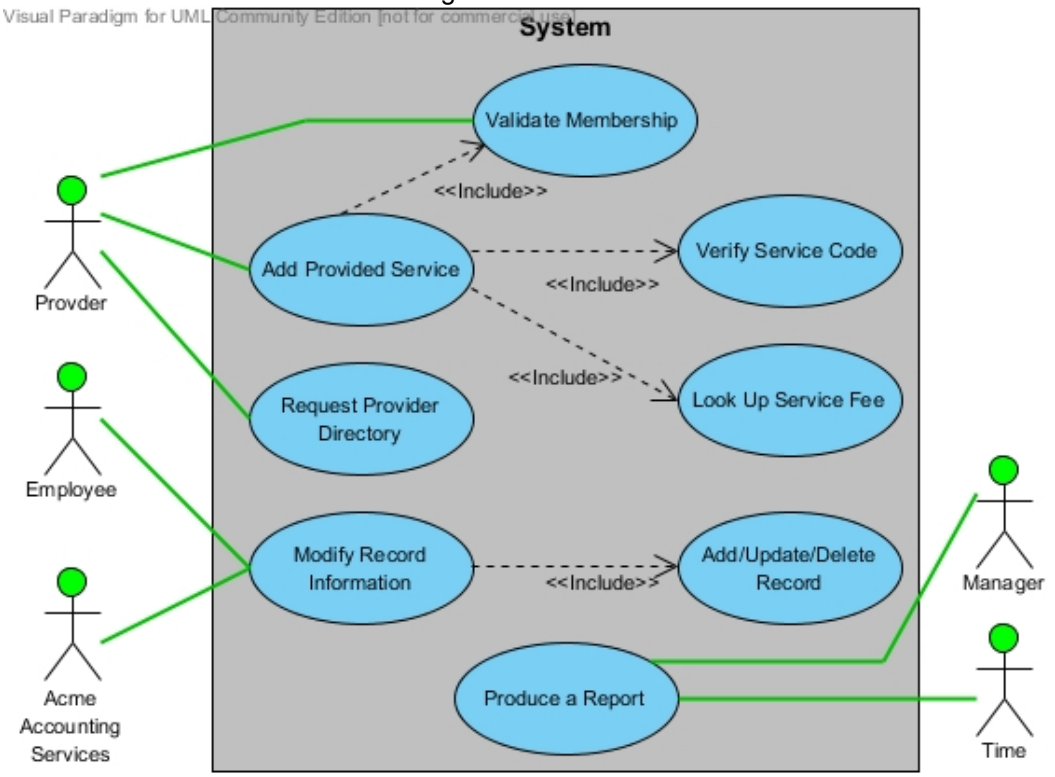


Figure 9 – The class diagram with stereotypes.

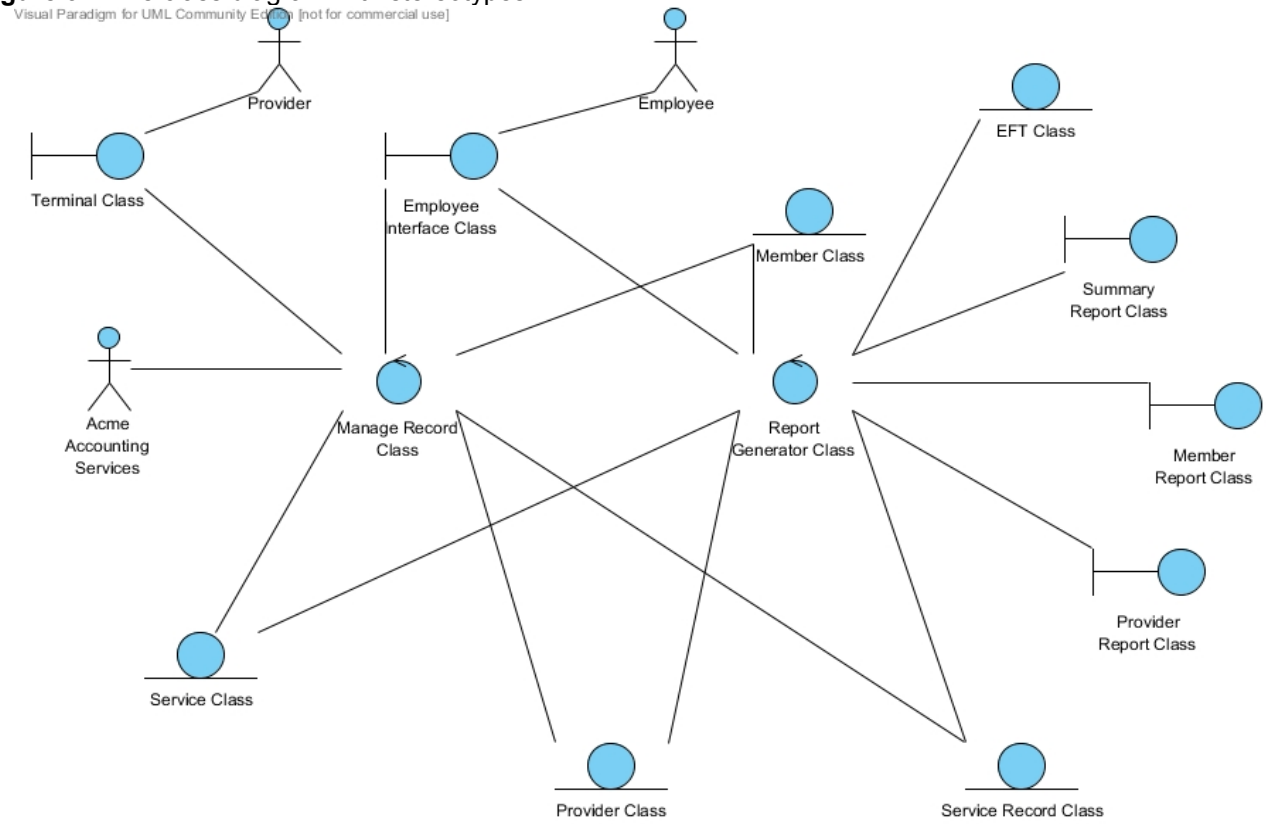
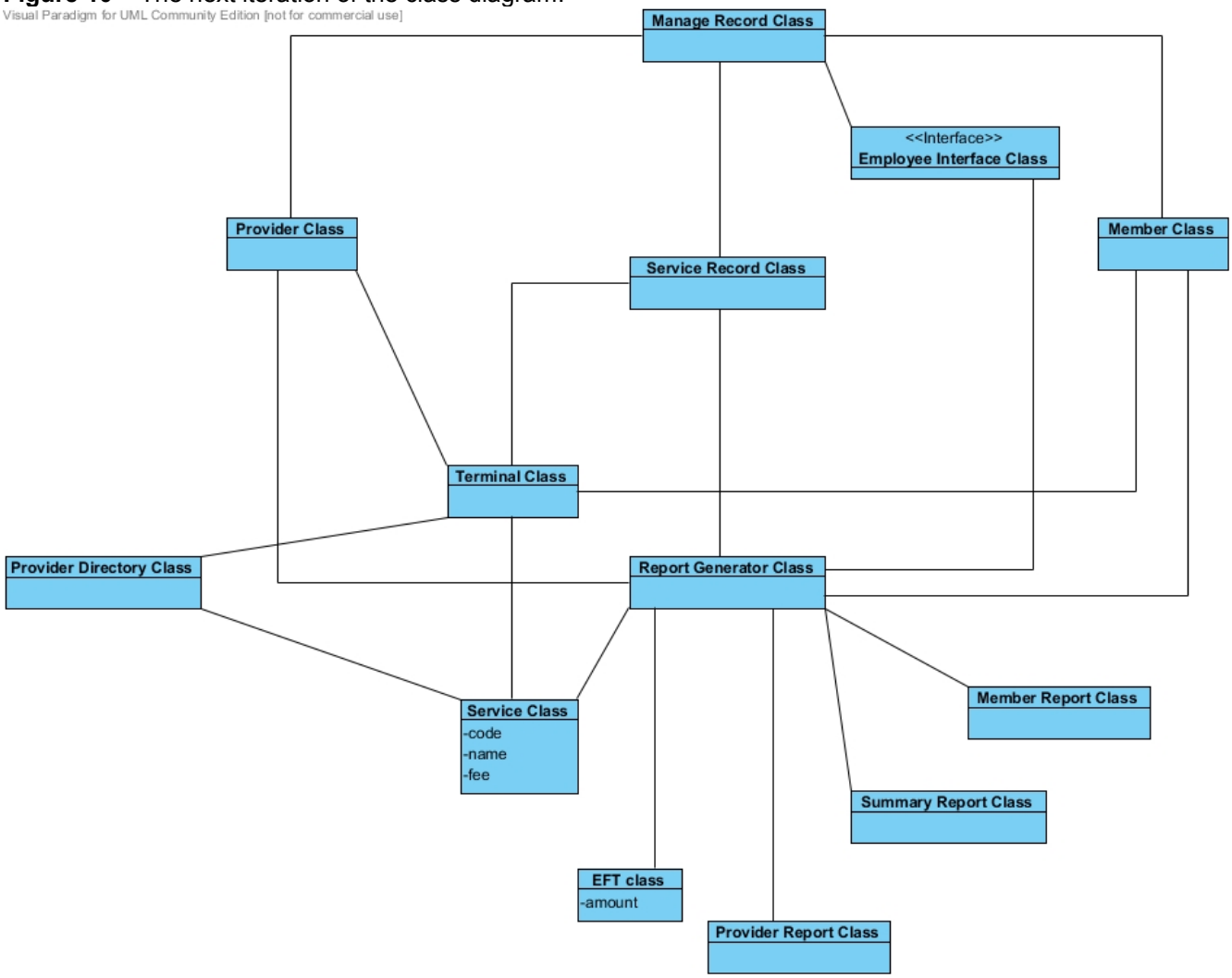


Figure 10 – The next iteration of the class diagram.



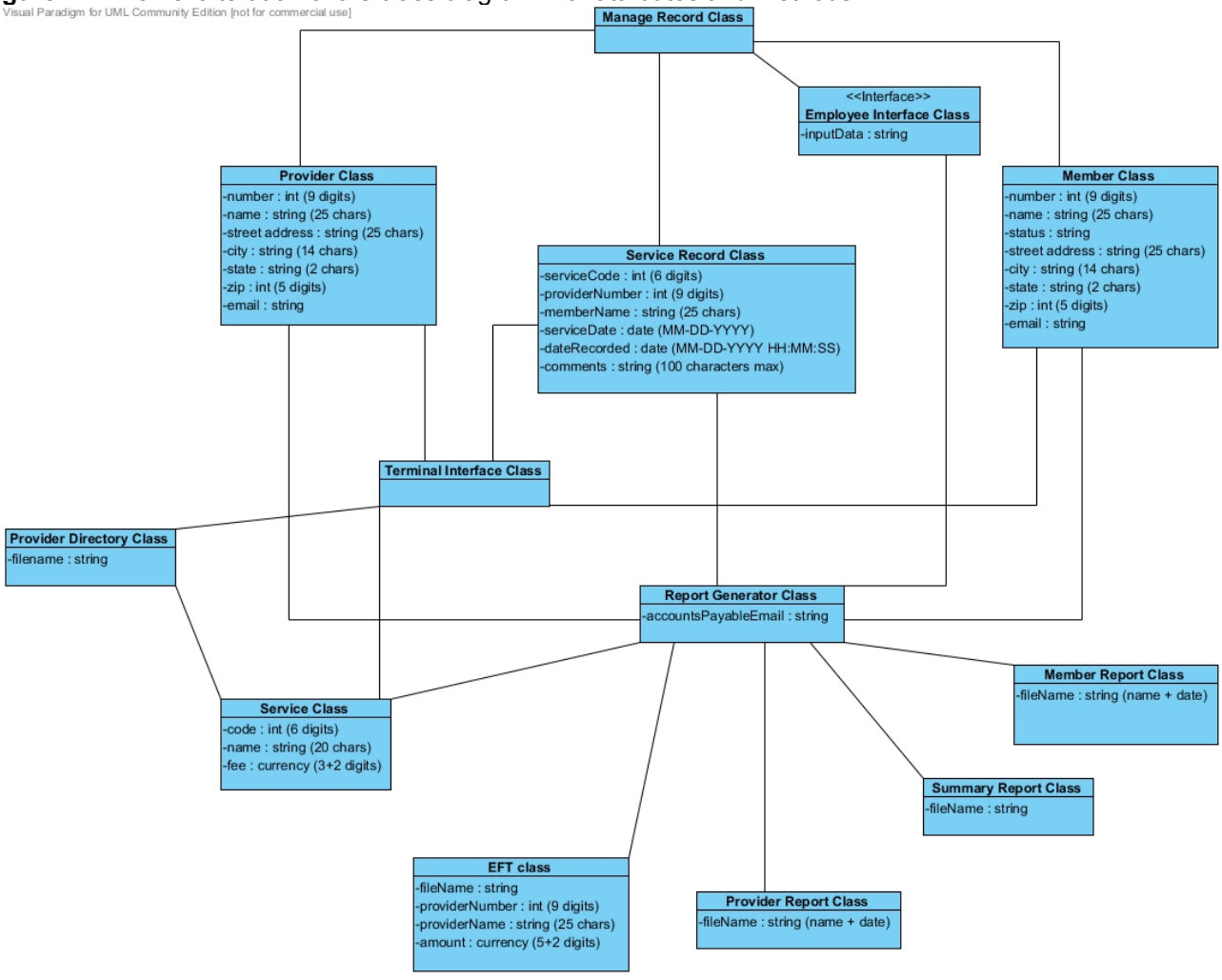
The final step of the analysis workflow is to write the software project management plan (SPMP). However, without information about the developing organization, the development of the SPMP cannot be done. We would need to know the employee hierarchy and policies in place at the development organization. So now, the analysis workflow is complete.

Problem 14.16

Starting with your specifications of Problem 12.20 or 13.22, design the Chocoholics Anonymous product (Appendix A). Use the design technique specified by your instructor (object-oriented design technique).

The specifications of Problem 13.22 will be used as inputs for the object-oriented design workflow of the Chocoholics Anonymous project. The design workflow is intended to allow successful implementation by programmers in the implementation workflow. To achieve this, two steps are required. The first step is to complete the class diagram with appropriate attributes with the aim of achieving high cohesion and low coupling. Figure 1 shows this newly updated class diagram.

Figure 1 – The next iteration of the class diagram with attributes and methods.



With our updated class diagram created, we can move on to the next step. The second step is to perform the detailed design. The goal is to create methods that can be easily implemented when passed to programmers. However, their is no language specification for

the ChocoAn project as of yet. Nevertheless we will still outline the methods using program description language (PDL). The class methods are outlined in Figures 2.1 – 2.5.

Figure 2.1 – Generic outlines for all getter/setter methods.

Method name	**getAttribute**
Return type	*attribute type*
Input arguments	none
Output arguments	none
Error messages	none
Files accessed	none
Files changed	none
Methods called	none
Narrative	This method will return the value of the attribute specified by the specific getter as whatever type is specified by that attribute.

Method name	**setAttribute**
Return type	**void**
Input arguments	*attribute type*
Output arguments	none
Error messages	none
Files accessed	none
Files changed	none
Methods called	none
Narrative	This method will update the value of the attribute specified by the specific setter.

Figure 2.2 – Outline for the Employee Interface class

Method name	**sendRequest**
Return type	**void**
Input arguments	*input data*
Output arguments	none
Error messages	"invalid data", "error retrieving record", "error updating record", "error creating record"
Files accessed	none
Files changed	none
Methods called	**ManageRecord**::**modifyMember**, **modifyProvider**, **addMember**, **addProvider**, **getMember**, **getProvider**, **updateMember**, **updateProvider**, **deleteMember**, **deleteProvider**
Narrative	This method will determine the type of request from the user by the on-screen button pressed. It will validate any input data values needed and return an error message if they are not valid. For the various method calls to the ManageRecord class, the method will print the returned information, confirmation, or error message as required.

Figure 2.3 – Outlines for the Manage Record class. Methods listed that provide information about Member classes have corresponding Provider class methods (not shown here).

Method name	**retrieveMemberInfo**
Return type	*list of Member class attribute types*
Input arguments	none
Output arguments	none
Error messages	none
Files accessed	none
Files changed	none
Methods called	none
Narrative	This method will determine the types of all member attributes and return them as a list.

Method name	**addMember**
Return type	**int**
Input arguments	*list of Member attribute values*
Output arguments	none
Error messages	none
Files accessed	none
Files changed	Member record
Methods called	none
Narrative	This method will create a new member record using the supplied list of member attribute values. If the creation is successful, it will return a value of 1, if the creation fails it will return a value of -1.

Method name	**getMember**
Return type	*list of Member attribute values*
Input arguments	none
Output arguments	none
Error messages	none
Files accessed	Member record
Files changed	none
Methods called	**Member**::**getAttribute**
Narrative	This method will retrieve all Member class values and return them as a list.

Method name	**updateMember()**
Return type	**int**
Input arguments	none
Output arguments	none
Error messages	none
Files accessed	Member record

Files changed	Member record
Methods called	**Member**::**setAttribute**
Narrative	This method will access a member record and change existing values using the supplied list of member attribute values. If the update is successful, it will return a value of 1, if the creation fails it will return a value of -1.

Method name	**deleteMember**
Return type	**int**
Input arguments	**int**
Output arguments	none
Error messages	none
Files accessed	Member record
Files changed	Member record
Methods called	none
Narrative	This method will delete the member record associated with the supplied integer value. If the deletion is successful, it will return a value of 1, if the creation fails it will return a value of -1.

Figure 2.4 – Outlines for the Terminal Interface class

Method name	**verifyMember**
Return type	**String**
Input arguments	**int**
Output arguments	**TerminalInterface**::**createServiceRecord**:String
Error messages	"no record found"
Files accessed	Member record
Files changed	none
Methods called	**Member**::**getAttribute**
Narrative	This method will look up the status and name of a member associated with the input parameter. If the record is found, it will return the member status as a String, otherwise it will return an error message to indicate the record could not be found. The name of the member is stored for use in creating a provided service record.

Method name	**verifyServiceCode**
Return type	**String** (formatted as a currency value)
Input arguments	**int,** *date*
Output arguments	**TerminalInterface**::**createServiceRecord**:int **TerminalInterface**::**createServiceRecord**:date
Error messages	"no record found"
Files accessed	Service record
Files changed	none
Methods called	**Service**::**getAttribute**

Narrative	This method will look up the fee of a Service associated with the integer input parameter. If the record is found, it will return the name of the service. Otherwise, it will return an error message to indicate the record could not be found. The input values are stored for use when creating the provided service record.

Method name	**LookUpServiceFee**
Return type	**String** (formatted as a currency value)
Input arguments	**int**
Output arguments	none
Error messages	none
Files accessed	Service record
Files changed	none
Methods called	**Service**::**getAttribute**
Narrative	This method will look up the fee of a Service associated with the input parameter. It will return the fee value formatted as a currency value.

Method name	**createServiceRecord**
Return type	**String**
Input arguments	**int**, **int**, **String**, *date*, **String**
Output arguments	**TerminalInterface**::**LookUpServiceFee**:int
Error messages	"error creating record"
Files accessed	none
Files changed	ServiceRecord record
Methods called	**TerminalInterface**::**LookUpServiceFee**
Narrative	This method takes the input values and uses the current date and time to create a new ServiceRecord record. If the operation is successful, it returns a confirmation message, if unsuccessful, it returns an error message.

Figure 2.5 – Outlines for the Report Generator class

Method name	**createMemberReport**
Return type	**int**
Input arguments	**int**, *date*
Output arguments	none
Error messages	none
Files accessed	Member record, ServiceRecord records
Files changed	MemberReport file
Methods called	**Member**::getAttribute, **ServiceRecord**::getAttribute
Narrative	This method will use the input parameter. to retrieve Member attributes. It will then search through all ServiceRecord files containing the parameter for the supplied date range. A report file is created with all the retrieved information formatted according to specifications. It is then emailed to the associated member. If the file is created and emailed successfully, it will return a value of 1, otherwise a value of -1.

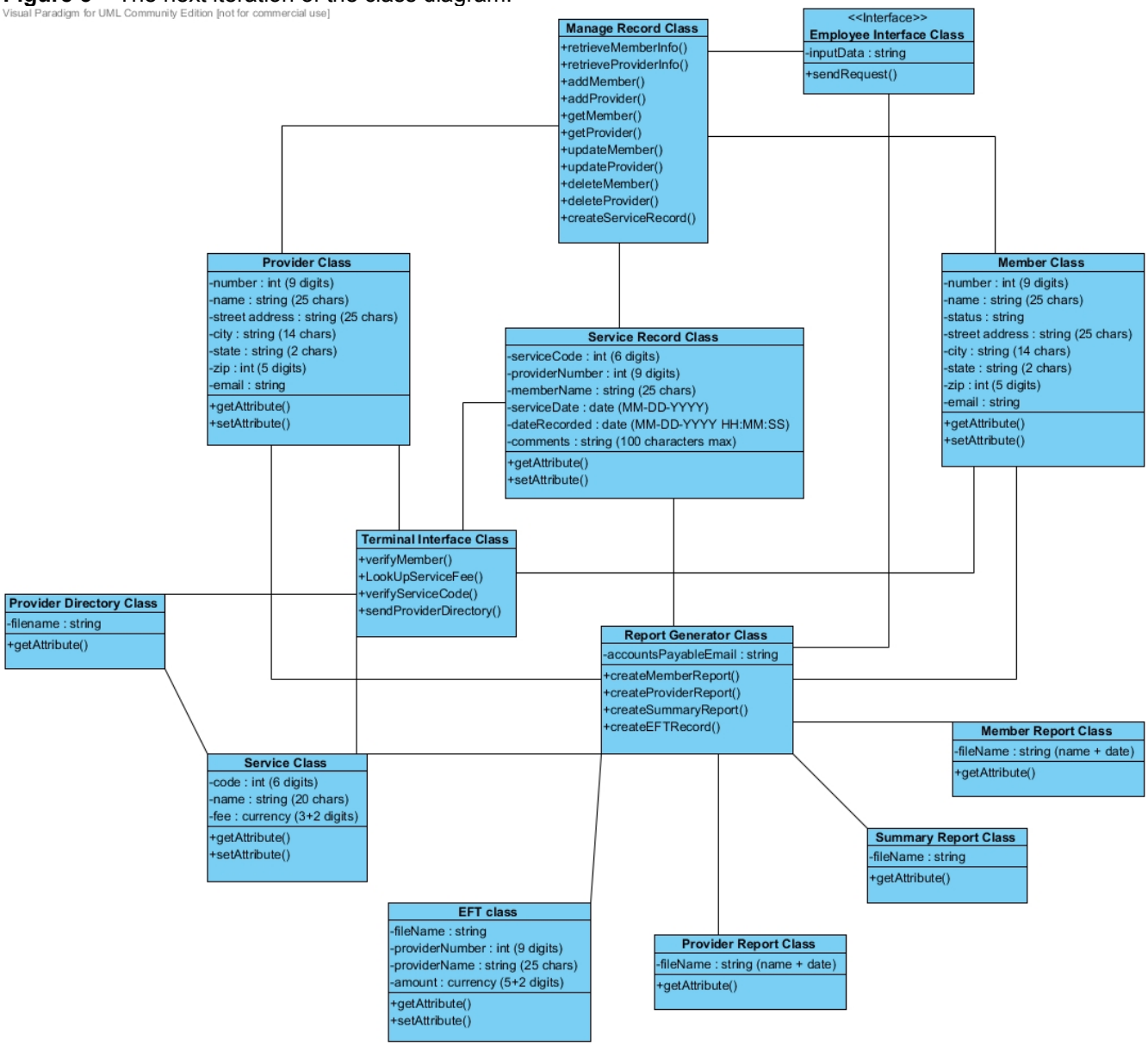
Method name	**createProviderReport**
Return type	**int**
Input arguments	**int**, *date*
Output arguments	**ReportGenerator**::**createEFTRecord**:int **ReportGenerator**::**createEFTRecord**:int **ReportGenerator**::**createEFTRecord**:*currency*
Error messages	none
Files accessed	Provider record, ServiceRecord records
Files changed	ProviderReport file
Methods called	**Provider**::**getAttribute**, **ServiceRecord**::**getAttribute**, **ReportGenerator**::**createEFTRecord**
Narrative	This method will use the input parameter. to search through all ServiceRecord files containing the Provider number for the supplied date range. For each record containing the parameter, the method will keep a running total of the number of records and a sum of all fees. The total of fee is used to create an EFTRecord. The method then retrieves Provider attributes. It also uses the Provider name and number to create an EFTRecord. A report file is created with all the retrieved and calculated information formatted according to specifications. It is then emailed to the associated Provider. If the file is created and emailed successfully, it will return a value of 1, otherwise a value of -1.

Method name	**createEFTRecord**
Return type	**int**
Input arguments	**int**, **String**, *currency*
Output arguments	none
Error messages	none
Files accessed	none
Files changed	EFT record
Methods called	none
Narrative	This method will use the input parameters to create an EFT record. If the record is created successfully, it will return a value of 1, otherwise, a value of -1.

Method name	**createSummaryReport**
Return type	**int**
Input arguments	**int**, *date*
Output arguments	none
Error messages	none
Files accessed	Provider record, ServiceRecord records
Files changed	SummaryReport file
Methods called	**Provider**::**getAttribute**, **ServiceRecord**::**getAttribute**
Narrative	This method will use the input parameter. to retrieve all ServiceRecords for the given date range, keeping a running total of records found and all associated fees. It will then find the information of all Providers associated with the matching ServiceRecords and keep a total of these as well. A report file is created with all the retrieved and calculated information formatted according to specifications. It is then emailed to the ChocoAn manager of accounts payable. If the file is created and emailed successfully, it will return a value of 1, otherwise a value of -1.

With all methods assigned to classes, the next iteration of the class diagram can be seen in Figure 3.

Figure 3 – The next iteration of the class diagram.



With the design steps complete, the project is now ready for implementation.

Problem 15.33

Draw up black-box test cases for the product you specified in Problem 12.20 or 13.22. For each test case, state what is being tested and the expected outcome of that test case.

To create the black-box test cases for the Chocoholics Anonymous product, we begin with test cases derived from equivalence classes and boundary value analysis. See Figure 1 for these test cases.

Figure 1 – Black-box test cases from equivalence classes and boundary analysis.

Provider attributes:	
Equivalence classes for `number`	
1. any digit non-numerical	Error
2. non-integer number	Error
3. negative number	Error
4. < 9 digits	Error
5. 9 digits	Acceptable
6. > 9 digits	Error
Equivalence classes for `name`, `streetAddress`	
1. any character non-alphabetical	Error
2. < 1 characters	Error
3. 1 character	Acceptable
4. between 1 and 25 characters	Acceptable
5. 25 characters	Acceptable
6. > 25 characters	Error
Equivalence classes for `city`	
1. < 1 characters	Error
2. 1 character	Acceptable
3. between 1 and 14 characters	Acceptable
4. 14 characters	Acceptable
5. > 14 characters	Error
Equivalence classes for `state`	
1. any character non-alphabetical	Error
2. 2 letter code not matching state	Error
3. < 1 characters	Error
4. 1 character	Acceptable
5. 2 characters	Acceptable
6. > 2 characters	Error
Equivalence classes for `zip`	
1. any digit non-numerical	Error
2. < 5 digits	Error
3. 5 digits	Acceptable
4. > 5 digits	Error
Equivalence classes for `email`	
1. not in proper email format	Error
2. in proper email format	Acceptable
Member attributes (`name`, `streetAddress`, `city`, `state`, `zip`, and `email` are equivalent to Provider above):	
Equivalence classes for `status`	
1. doesn't contain valid message	Error

2. contains valid message	Acceptable
Service attributes:	
Equivalence classes for `code`	
1. any digit non-numerical	Error
2. non-integer number	Error
3. negative number	Error
4. < 6 digits	Error
5. 6 digits	Acceptable
6. > 6 digits	Error
Equivalence classes for `name`	
1. includes special character	Error
2. < 1 characters	Error
3. 1 character	Acceptable
4. between 1 and 20 characters	Acceptable
5. 20 characters	Acceptable
6. > 20 characters	Error
Equivalence classes for `fee`	
1. any digit non-numerical	Error
2. < 2 digits after decimal	Error
3. 2 digits after decimal	Acceptable
4. > 2 digits after decimal	Error
5. < $0.00	Error
6. $0.00	Acceptable
7. $0.01	Acceptable
8. between $0.01 and $999.97	Acceptable
9. $999.98	Acceptable
10. $999.99	Acceptable
11. $1000.00	Error
12. > $1000.00	Error
ServiceRecord attributes (`serviceCode`, `providerName`, `memberName` correspond to cases above):	
Equivalence classes for `serviceDate`	
1. not in MM-DD-YYYY format	Error
2. before founding of ChocoAn	Error
3. between founding and `dateRecorded`	Acceptable
4. after `dateRecorded`	Error
5. month < 01	Error
6. month = 01	Acceptable
7. month between 1 and 12	Acceptable
8. month = 12	Acceptable
9. month > 12	Error
10. day < 01	Error
11. day = 01	Acceptable
12. day between 1 and month max	Acceptable
13. day = month max	Acceptable
14. day > month max	Error
Equivalence classes for `dateRecorded`	
1. not in MM-DD-YYYY HH:MM:SS format	Error
2. before founding of ChocoAn	Error
3. between founding and current date	Acceptable
4. after current date	Error
5. before `dateRecorded`	Error

6. same as `dateRecorded`	Acceptable
7. after `dateRecorded`	Acceptable
8. month < 01	Error
9. month = 01	Acceptable
10. month between 01 and 12	Acceptable
11. month = 12	Acceptable
12. month > 12	Error
13. day < 01	Error
14. day = 01	Acceptable
15. day between 01 and month max	Acceptable
16. day = month max	Acceptable
17. day > month max	Error
18. hour < 00	Error
19. hour = 00	Acceptable
20. hour between 00 and 24	Acceptable
21. hour = 24	Acceptable
22. hour > 24	Error
23. minute or second < 00	Error
24. minute or second = 00	Acceptable
25. minute or second between 00 and 59	Acceptable
26. minute or second = 59	Acceptable
27. minute or second > 59	Error
Equivalence classes for `comments`	
1. < 1 character	Error
2. 1 character	Acceptable
3. between 1 and 100 characters	Acceptable
4. 100 characters	Acceptable
5. > 100 characters	Error
EFT attributes (`providerNumber`, `providerName`, correspond to cases above):	
Equivalence classes for `amount`	
1. any digit non-numerical	Error
2. < 2 digits after decimal	Error
3. 2 digits after decimal	Acceptable
4. > 2 digits after decimal	Error
5. < $0.00	Error
6. $0.00	Acceptable
7. $0.01	Acceptable
8. between $0.01 and $99,999.97	Acceptable
9. $99,999.98	Acceptable
10. $99,999.99	Acceptable
11. $100,000.00	Error
12. > $100,000.00	Error
Equivalence classes for `filename`	
1. includes special character	Error
2. incorrect file extension	Error
3. meets file name specifications	Acceptable
Attributes for **ProviderDirectory**, **MemberReport**, **ProviderReport**, **SummaryReport**, classes use the `filename` test case above.	

We also need to include the functional testing use cases as seen in Figure 2.

Figure 2 – Functional analysis test cases from equivalence classes and boundary analysis.

The functions outlined for test cases:

1. Verify a membership.
2. Add a provided service.
3. Add a new member.
4. Update a member's info.
5. Delete a member.
6. Add a provider.
7. Update a provider's info.
8. Delete a provider.
9. Create a provider directory.
10. Create a member report.
11. Create a provider report.
12. Create a summary report.

In addition to direct tests it is necessary to perform the following additional tests:

13. Attempt to verify a member without 'Verified' status.
14. Attempt to verify a member not on file.
15. Attempt to add a provided service already on file.
16. Attempt to add a new member already on file.
17. Attempt to update a member not on file.
18. Attempt to update fields of a member twice and check that the second version is stored.
19. Attempt to delete a member not on file.
20. Attempt to delete twice a member already on file.
21. Attempt to add a new provider already on file.
22. Attempt to update a provider not on file.
23. Attempt to update fields of a provider twice and check that the second version is stored.
24. Attempt to delete a provider not on file.
25. Attempt to delete twice a provider already on file.
26. Attempt to create a provider directory sent to a provider not on file.
27. Attempt to create a provider report twice and check that only one EFT record is created.
28. Attempt to create each report for the current week and check it includes only up to the current date.
29. Attempt to create each report twice for a given week range and ensure only one file is written.

Now the black-box test cases can be run by the SQA group to test the implementation code. Once the integration process is complete, product testing and acceptance testing will need to be completed before the software is ready to be officially passed on to the client.

Problem 16.15

Suppose that the product for Chocoholics Anonymous in Appendix A has been implemented exactly as described. Now the product has to be modified to include endocrinologists as providers. In what ways will the existing product have to be changed? Would it be better to discard everything and start again from scratch? Compare your answer to the answer you gave to Problem 1.19.

In my original answer to this question in Problem 1.19, I stated that the project should be started again from scratch. My reasoning was that adding the new type of provider to the system would be very difficult if the project were developed under the classical paradigm. I felt that developing under the object-oriented paradigm would allow this functionality to be easily added to the system and also make it easier to add future functionality. However, after reading the textbook to this point, I have changed my position. The change would have to be analyzed with a cost-benefit analysis to determine the best way to proceed with the implementation of this new functionality.

To determine the cost-benefit analysis, the management at the developing organization will need to consider the changes needed to the product. If implemented under the object-oriented paradigm, a simple attribute change to the Provider class is needed, and extra records for the types of services they provide will need to be added to the database. These perfective maintenance changes should be small and relatively easy to implement, so the cost-benefit analysis outcome is likely to be in favour of modification over development of a completely new product. This is basically the same as my previous answer.

If the product is developed under the classical paradigm, this will have a different effect on the cost-benefit analysis. The addition of endocrinologists will mean that new provider types and services will need to be added to the existing database. Many of the functions for the system will need to incorporate this new provider type. This will require maintenance programmers to significantly change the source code in some areas. Programmers may have to many difficulties in changing the database to accommodate the new design or problems reworking modules to handle new types of variables. Nevertheless, this maintenance is still likely to be less expensive than development of an entirely new product.

However, management will also need to consider the possibility of future changes to the product and the economic impact of these changes. If the product were to be restarted under the object-oriented paradigm, the cost of future maintenance is likely to decrease. This decrease will have to offset the cost of the development of an entirely new product. On the other hand, the cost of developing again from scratch is probably less expensive than the original development because some of the artifacts from the original can be reused in the new version.

Another factor to consider is time. The time it takes to develop the new product means that the current product will still have to be used and maintained. The endocrinologist functionality may need to be added before the new product can be developed. This would mean that the product would be modified and restarted from scratch at the same time, which would end up being much more costly than either solution alone. Even if there is time to develop the new

product before the functionality is included, the maintenance of the original system during this time will add to the cost of development. Considering all these factors, the cost-benefit analysis may determine that it is better to start again from scratch. However, I feel that this is unlikely. In both the classical and object-oriented paradigm versions of development, it is probably better to re-work the existing product rather than start again from scratch.

I have included my original answer here for comparison:

The Chocoholics Anonymous software will have to be revised in a few ways to include the addition of endocrinologists as providers. New provider numbers will have to be issued if the endocrinologists exist as separate providers and not part of an existing provider group. Additionally, new types of services that they provide will need to be added to the system, along with their corresponding names, fees, and codes. This information must also be included in an updated Provider Directory. The rest of the system will need to be able to handle these changes.

If the system were designed *exactly* as described in Appendix A, then these additions would be difficult to implement because the system is tailored to the specific criteria listed. This means the system can only handle dieticians, internists, and exercise experts. Correspondingly, the system will only be able to handle the treatment options available at implementation. This includes the names, fees, and codes of services. If the provider directory is a simple document, this may be able to be edited for changes in a word-processing (or similar) program, but these changes will not be reflected in the system.

As such, I would recommend that the company indeed discard everything and start from scratch. This way, the system can be designed so that post-delivery maintenance of this kind is easily done and can quickly meet client needs. The best way to achieve this result would be to design the system under the object-oriented paradigm. For example, the terminal at the service provider's location sends a message containing the information mentioned. The object representing the account of the provider could then be updated with whatever the message contains, and the same goes for the member object. Internally, the objects can process the information however they wish. Continuing the example, a database might store the list of services and associated information. The objects in question could simply query the database to provide the information to the users. Then to update the system with new information, simply requires an addition to the database.

Therefore, the best solution is to discard everything and redesign the software from scratch under an object-oriented paradigm. This will extend the life cycle of the product because updates to the Chocoholics Anonymous program can be implemented with relative ease to keep the software current and relevant to meet changing client and user needs. This will also reduce overall costs, because maintenance is such a large part of the total cost of software development. And finally, designing the product in this way meets the ethical standards of software engineering by acting in the best interests of the client, and the product will be meeting the highest of professional standards.