UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

LABORATORIO 4

PARADIGMAS DE PROGRAMACIÓN

Vicente I. Ortiz Arancibia

Profesores: Víctor Flores
Roberto González
Luis Celedón
Fecha de Entrega: 07-09-2018

Santiago de Chile

1 - 2018

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

Un paradigma es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento. En el contexto informático, se dice que un paradigma es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales.
Durante este semestre se estudiarán 4 de estos paradigmas, el funcional, lógico, orientado a objetos y a eventos que son los principales paradigmas existentes actuales.

El presente informe se desarrollará bajo el estudio amparado por el paradigma orientado a objetos, y paradigma orientado a eventos.

El paradigma orientado a objetos ve un programa como un conjunto de clases. Las clases son entidades que combinan atributos, que, si lo comparamos con paradigmas como el imperativo, sería sus variables, y un método, serían las funciones. El POO surge para solventar los problemas que planteaban otros paradigmas, como el imperativo, con el objeto de elaborar programas y módulos más fáciles de escribir, mantener y reutilizar.

Por otro lado, el paradigma orientado a eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

El objetivo principal de este informe es desarrollar un chatbot (ver anexo para la definición), cuyo tema en este caso es un asistente de vendedor de preservativos. Este programa emulara de una manera muy básica la inteligencia artificial que manejan actualmente chatbots muy avanzados, y que se han atrevido a desafiar el test de Turing. También se busca demostrar, que, creando un programa en Visual Studio, se pueden mezclar paradigmas como el orientado a objetos, y el orientado a eventos aplicando los conocimientos adquiridos en cátedra y laboratorio sobre estos paradigmas y el lenguaje de programación C#

CAPÍTULO 2. MARCO TEÓRICO

2.1 PARADIGMA ORIENTADO A OBJETOS

La programación orientada a objetos (POO) nace de la necesidad de encontrar un modelo de programación que acelerase la productividad de los desarrolladores de *software*. Para ello, el concepto clave es la reutilización, a través de la herencia y la asociación dinámica. Por otro lado, la complejidad de los programas se hacía cada vez mayor, por lo que existía la necesidad de abstraerse de ciertos conceptos y concentrarse en resolver el problema, lo que derivó en la idea de tipos de datos abstractos.

2.2. CLASE

Un tipo de dato abstracto puede definirse a través de una clase. Una clase relaciona una serie de atributos (variables) y una serie métodos (funciones u operaciones) que operan sobre la clase.

2.3. OBJETO

Es una unidad compacta, que se encuentra en tiempo y que hace las tareas dentro de un programa. En programación, los objetos se utilizan para el modelamiento de entidades y/o de objetos en el mundo real (el objeto lápiz, goma, mesa, etc.). En otras palabras, un objeto es la representación de un concepto dentro del programa, y tiene la información que necesita para su correcta abstracción, es decir, datos que permiten describir sus características (atributos) como sus operaciones que pueden realizarse sobre esta.

2.4. PARADIGMA ORIENTADO A EVENTOS

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por lo sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

2.5. EVENTOS

Un evento es la notificación que algo ha ocurrido, tal como el click que se hace en el botón derecho del mouse o cuando digitamos en una pantalla táctil. Estrictamente hablando, el sistema crea un objeto en tiempo de ejecución en respuesta a una acción.

CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita desarrollar un programa en el lenguaje de programación C# que simule de la mejor manera un chatbot. Este chatbot debe tener características mínimas como:

- Tener un contexto y temática definida.
- Diseño de flujos conversaciones

- Considerar un amplio abanico de respuestas, con el fin de cautivar mediante una conversación activa con el usuario.
- Incorporar respuestas que permitan reaccionar a distintas situaciones.
- Debe responder a mensajes cuya estructura gramatical sea simple, en español y conocido por el chatbot.
- Ante entradas desconocidas, el chatbot debe contar con un repertorio de respuestas que le permitan redirigir la conversación con el usuario.

Además, como parte del diseño orientado a objetos de la solución el enunciado, se debe considerar como mínimo cubrir las siguientes entidades a través de múltiples clases relacionadas:

1. **Chatbot:** Corresponde a la(s) entidad(es) que interactuarán con el usuario respondiendo sus consultas. El chatbot deberá tener personalidades (ej.: coloquial, formal, agresivo, etc.) determinado por un Seed (semilla pseudoaleatoria) para lograr variabilidad de las respuestas del chatbot.
2. **Diseñar un log:** Está formado por los mensajes que emite tanto el chatbot como el usuario. Cada entrada del Log deberá incluir el mensaje, la fecha en que se emitió (timestamp) e indicar su emisor (usuario o chatbot) y deberán ser almacenadas para futura referencia.
3. **Usuario:** Esta entidad está formada por toda la información útil que pueda obtener el chatbot sobre el usuario a medida que conversa con éste.

Finalmente, en la interacción, se debe aplicar un diseño orientado a eventos, donde haya una interfaz gráfica que maneje las interacciones mínimas con el chatbot. En todo momento deben poder cumplirse funcionalidades mínimas, como:

1. **Iniciar una conversación:** Luego de presentarse un texto de bienvenida, mediante un botón, debe ser posible iniciar un nuevo chat con el robot.
2. **Enviar mensajes:** En todo momento el programa debe dejar al usuario enviar mensajes al robot, donde el chatbot debe ser capaz de responder en función a lo que le habla el usuario.
3. **Guardar Log:** El programa debe permitir guardar toda la interacción con el chatbot desde que se inició el chat. Además, debe permitir escoger la ubicación y el nombre donde se guardará el archivo log.
4. **Cargar Log:** Esta función permitirá cargar una conversación almacenada con la función *Guardar Log*. Debe permitir al usuario elegir la ubicación del archivo.
5. **Finalizar conversación**: Esta funcionalidad permite terminar la conversación.
6. **Rate:** Una vez cerrada la conversación, se debe poder evaluar la interacción con el chatbot.

3.1 ANÁLISIS

Como se mencionó anteriormente, una de las ventajas que nos ofrece el paradigma orientado a objetos, es el hecho de que las clases nos permite crear una modularización de la solución que implementaremos, por lo que podemos separar el problema, y luego la solución, en distintas entidades que trabajaran por si solas, pero que se juntarán en algún momento, permitiendo que el programa funcione de manera óptima.

En primer lugar, se debe analizar los aspectos que tienen que ver meramente con el chatbot, y los actores que lo incluyen. A grandes rasgos, podemos identificar 3 grandes actores dentro del flujo conversacional que implica un chatbot, el usuario no-maquina, que sería el que interactúa con la maquina; el usuario-maquina (chatbot), que almacena todas las respuestas configuradas a dar y que puede procesar el mensaje del usuario no-máquina para dar una respuesta acorde; y el mensaje, que sería el actor encargado almacenar toda la información del flujo conversacional.
Si esto lo llevamos a implementación de POO, podríamos decir que dentro de nuestra solución existirán 3 grandes clases, la clase Usuario, la clase Chatbot y la clase Mensaje.

El usuario podrá almacenar toda la información importante que se necesita almacenar respecto a su participación dentro del flujo conversacional, como el ID, el nombre, etc.

El chatbot almacenará las respuestas preconfiguradas que tiene para dar al usuario no-maquina, teniendo un amplio abanico de respuestas en función de la personalidad que este puede tomar. Además el chatbot será capaz analizar el mensaje de entrada del usuario, y procesar internamente cual será la mejor respuesta al mensaje ingresado por el usuario.

Finalmente, el mensaje es el encargado de almacenar todo el flujo conversacional entre el usuario y el chatbot, incluyendo datos importantes como la hora, las evaluaciones que se le pueden hacer al chatbot, entre otra información importante.

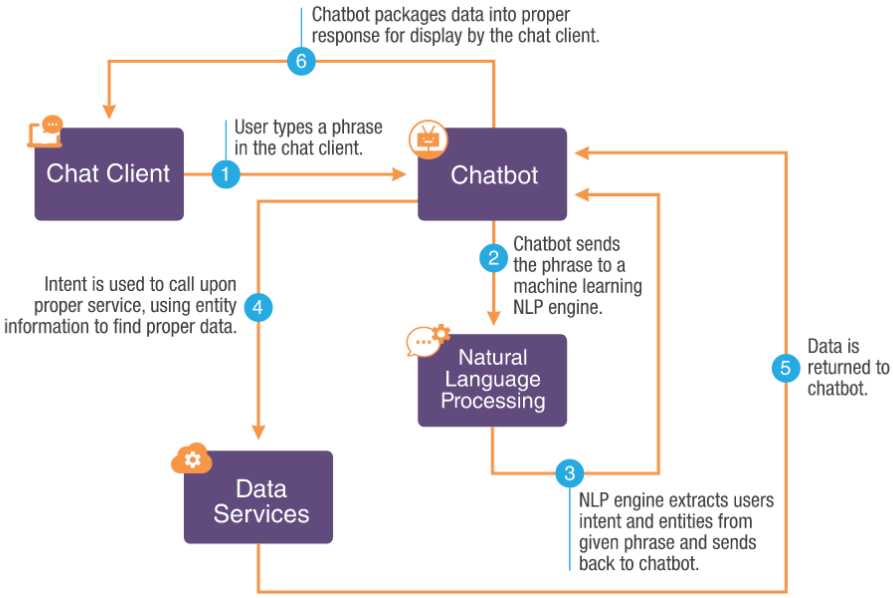

Ilustración 1 .-Diagrama UML de análisis

Ahora, además de identificar las clases existentes en la solución que se presentará, otro gran desafío es el chatbot funcionando en sí. Aquí es donde entra la interfaz gráfica que interactuará con el usuario, acogiendo sus entradas, evaluándolas y dando una respuesta acode a lo que el usuario está solicitando. Si miramos la Ilustración 1, podemos ver que en el flujo conversacional existe un *Chat Client* y un *Data Services*, si lo comparamos con la implementación que se debe

realizar, el *Chat Client* sería la interfaz gráfica o “FrontEnd”, y el *Data Services*, el “BackEnd” que procesa finalmente lo que el chatbot debe ser capaz de hacer.

Luego de este análisis, llevándolo tempranamente a un diagrama UML, quedaría algo así:

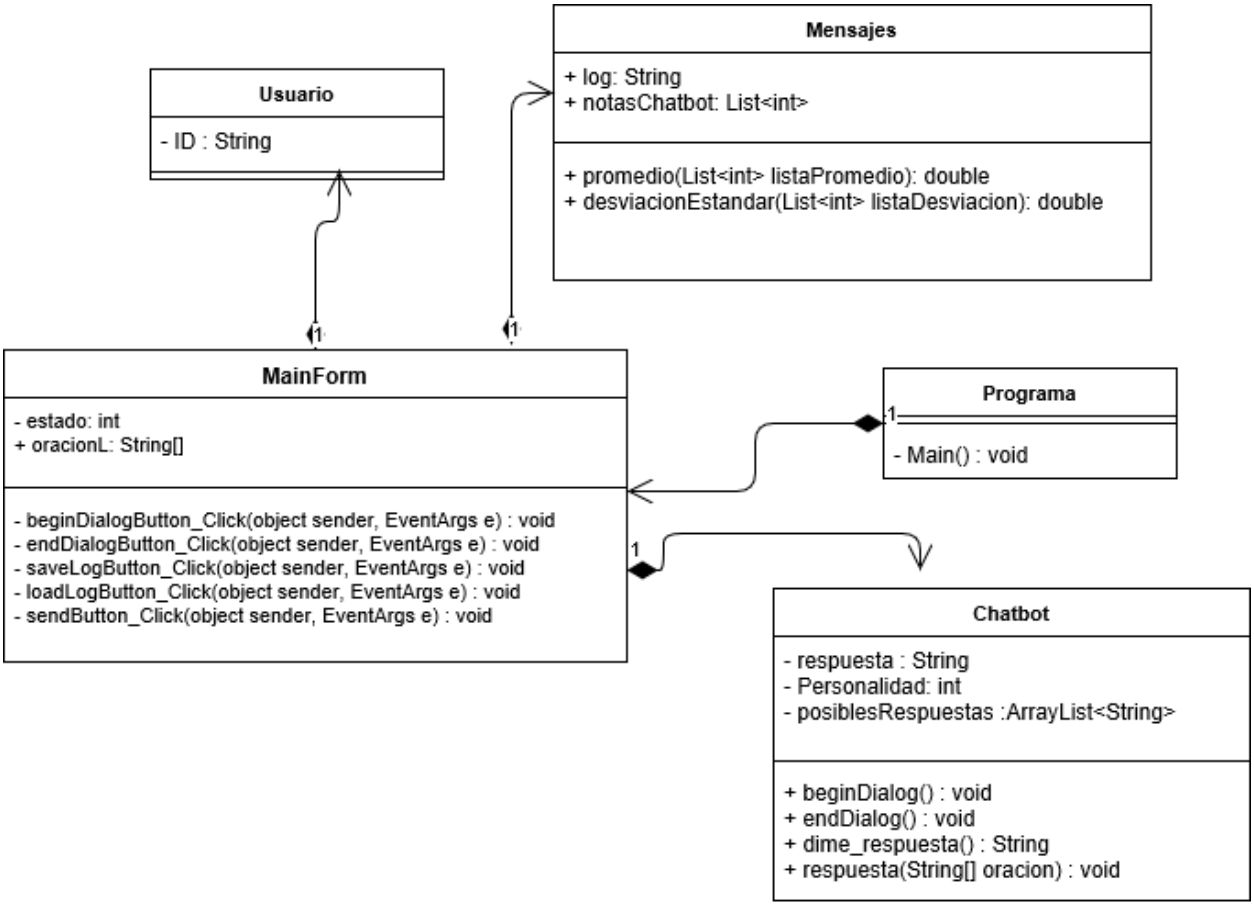

Ilustración 2 .- Diagrama conversacional chatbot

CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Como se mencionó en el análisis del problema, se crearon 3 grandes clases en el proyecto para que el programe funcione, a continuación, se procederá a explicar una por una estas clases.

4.1. CLASE USUARIO

En esta clase se crearon atributos que están asociados a datos importantes del usuario dentro del flujo conversacional, como lo son el ID y el nombre. Además se crearon métodos getter y setter, para poder modificar los atributos de esta clase.

4.2. CLASE CHATBOT

En esta clase finalmente se tuvieron que implementar más atributos de los que se diseñaron en la etapa de análisis. Ya que además de los atributos y métodos que se tenían contemplados, se tuvieron que agregar atributos *String[]* que contuvieran las posibles respuestas del Chatbot a determinado mensaje, además de las palabras que es posible de analizar e identificar el chatbot.

Por ejemplo, si el usuario escribiera por consola “me gustaría encargar 36 preservativos”, el chatbot entraría al arreglo de *Strings* **confirmarCompra** buscando las posibles palabras que puede identificar**,** y al identificar la palabra “encargar”, entraría al arreglo de *Strings* **RCCompra** para entregar una de las respuestas almacenadas en ese arreglo.

```
private String[] confirmarCompra = {"necesito", "encargar", "encargar?"};
private String[] RCCompra = {"Perfecto, dejeme confirmar si hay stock disponible",
                             "Deja revisar si me quedan.",
                             "RESPUESTA INDEFINIDA 3"};
```

Ilustración 3.- Ejemplo de arreglo de Strings que muestra las palabras posibles a identificar, y las posibles respuestas.

En el párrafo anterior, se mencionó implícitamente el proceso que se creó en el método **respuesta**, que es el encargado de analizar la entrada del Usuario, y entregar una respuesta acorde a lo ingresado. Para esto, en la presente clase, como se mencionó se creó este método que a continuación se muestra un extracto:

```
public void respuesta(String[] oracion) {
    int largo = oracion.length;
    int aux = 0;
    date = new Date();

    for(int i=0; i<largo; i++) {
        for(int j=0; j<informacion.length; j++) {
            if(oracion[i].equals(informacion[j])) {
                answer = hourdateFormat.format(date) + "Chatbot> " + Rinformacion[intPersonalidad];
                aux = 1;
            }
        }
    }
```

Ilustración 4.- Extracto del método respuesta de la clase Chatbot.

Como se aprecia, el único parámetro de este método es un arreglo de *String* llamado *oración*. Este parámetro es el mensaje ingresado por el usuario guardado en un arreglo de *Strings* que separa la oración en un espacio. Por lo que si el usuario ingreso “me gustaría encargar 36 preservativos”, el parámetro sería el siguiente arreglo: [“me”,”gustaría”,”encargar”,”36”,”preservativos”]. Este método analiza todos los elementos del arreglo hasta identificar alguna coincidencia con la “base” que existe preconfigurada dentro del chatbot. Si no encuentra nada, le avisa al usuario que no ha entendido su mensaje. Al encontrar una coincidencia, se procede a guardar la respuesta en la variable auxiliar *answer* tomando en cuenta la personalidad adoptada aleatoriamente por el chatbot, guardada en el atributo *intPersonalidad*.

Además de tener el método que hace la labor principal de un chatbot, esta clase también almacena los métodos **beginDialog** y **endDialog**, que son los encargados de dar respuesta a los eventos *beginDialog* y *endDialog*. Estos dan el mensaje de bienvenida y despedida, además de, en el caso de beginDialog entregar el ID al chat, y en conjunto dejar las marcas de tiempo donde se inicializo y finalizó el flujo conversacional.

4.3. CLASE MENSAJE

La clase mensaje, es la que cumple la función de almacenar todos los mensajes e interacciones hechas entre usuario y chatbot. Se creó el atributo **log**, que es del tipo *String*, ya que almacena todas las líneas de texto mostradas por consola en un string, que se usará en caso de que el usuario utilice el botón *saveLog* y se dispare ese evento.
Además, esta clase contiene el atributo **notasChatbot**, que es de tipo *ArrayList<String>* destinados a guardar las notas entregadas por el usuario con relación al chatbot activo en el momento.

4.4. MainForm

La clase MainForm es la encargada de ser el puente entre lo que pasa en el backend, y la interfaz gráfica desarrollada. En esta clase se guardan todos los eventos de los botones existentes en la interfaz gráfica. Existen botones como el beginDialog, endDialog, saveLog, loadLog, entre otros, que el main se encarga que al ser utilizados suceda algo en el programa en respuesta al evento seleccionado. Ej.: En caso de ingresar un texto y presionar el botón *send*, la clase MainForm procede ejecutar el método encargado a darle solución a lo requerido por el usuario.

```
public MainForm()...

private void beginDialogButton_Click(object sender, EventArgs e)...

private void endDialogButton_Click(object sender, EventArgs e)...

private void saveLogButton_Click(object sender, EventArgs e)...

private async void loadLogButton_Click(object sender, EventArgs e)...

private void sendButton_Click(object sender, EventArgs e)...

private void rateFunc()...

private void statistics_Click(object sender, EventArgs e)...
```

Ilustración 5.- Eventos existentes en la clase MainForm

CAPÍTULO 5. RESULTADOS

Debido a que se logró trabajar en armonía tanto el paradigma orientado a objetos, como el orientado a Eventos en Visual Studio, y con relación a los requerimientos mínimos funcionales, se puede mostrar a continuación que se alcanzó un 100% de desarrollo en cada uno de los puntos mencionados en la descripción del problema.

Al inicializar el programa, sale automáticamente el mensaje de bienvenida del chatbot.

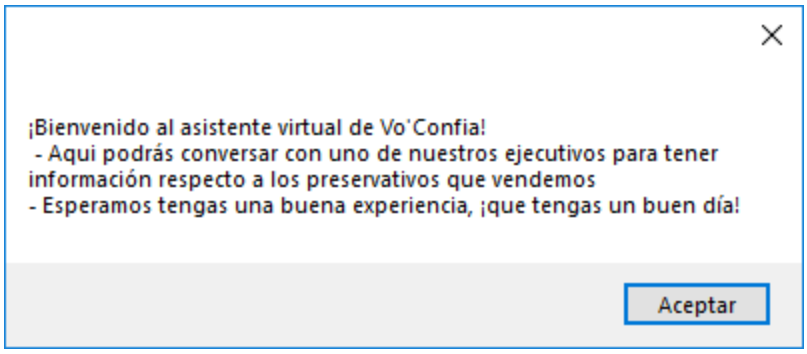

Ilustración 6.- Mensaje de bienvenida

Luego de hacer click en *Aceptar, podemos inicializar la conversación con el botón beginDialog, que hará que el chatbot entregue un mensaje de bienvenida en la conversación, asignará un ID al usuario para poder ligarlo con la conversación, y determinara aleatoriamente la personalidad del Chat.*

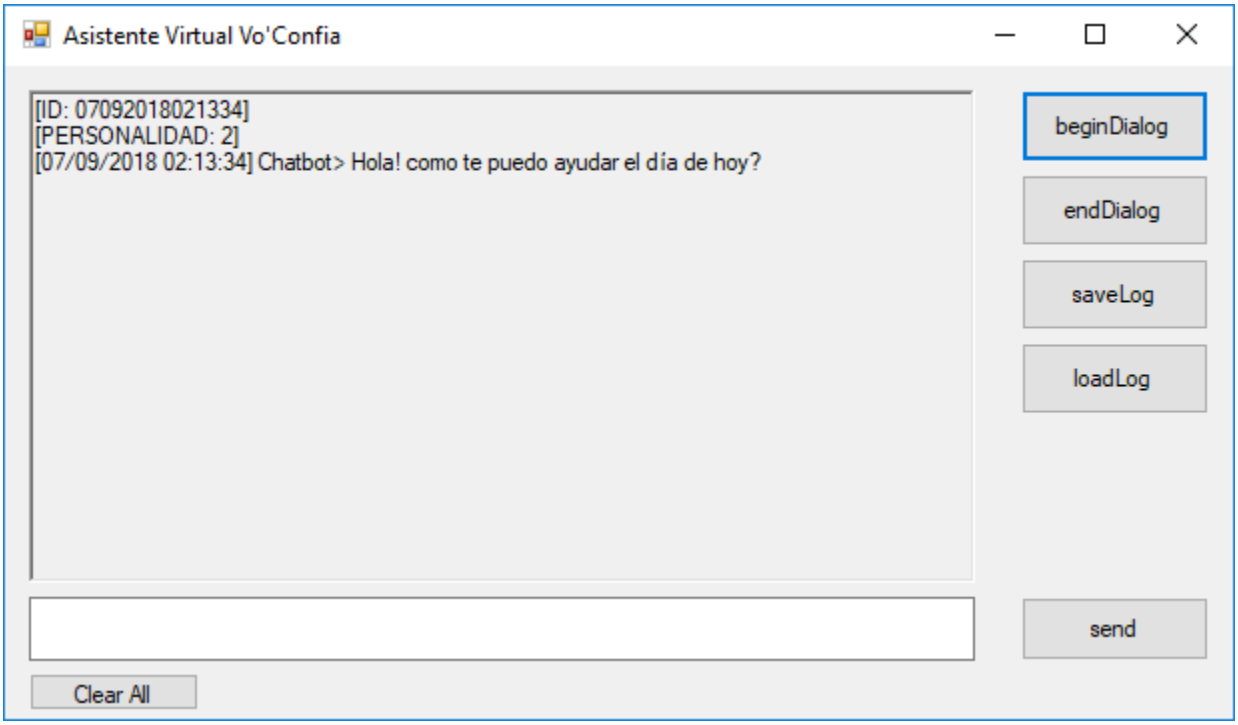

Ilustración 7.- Uso de botón beginDialog

A continuación, podemos proceder a enviar un mensaje con el botón *send*, o si queremos, finalizar la conversación, pero con fin de poder explicar mejor los resultados, procederemos a enviar un mensaje, y luego finalizaremos la conversación.

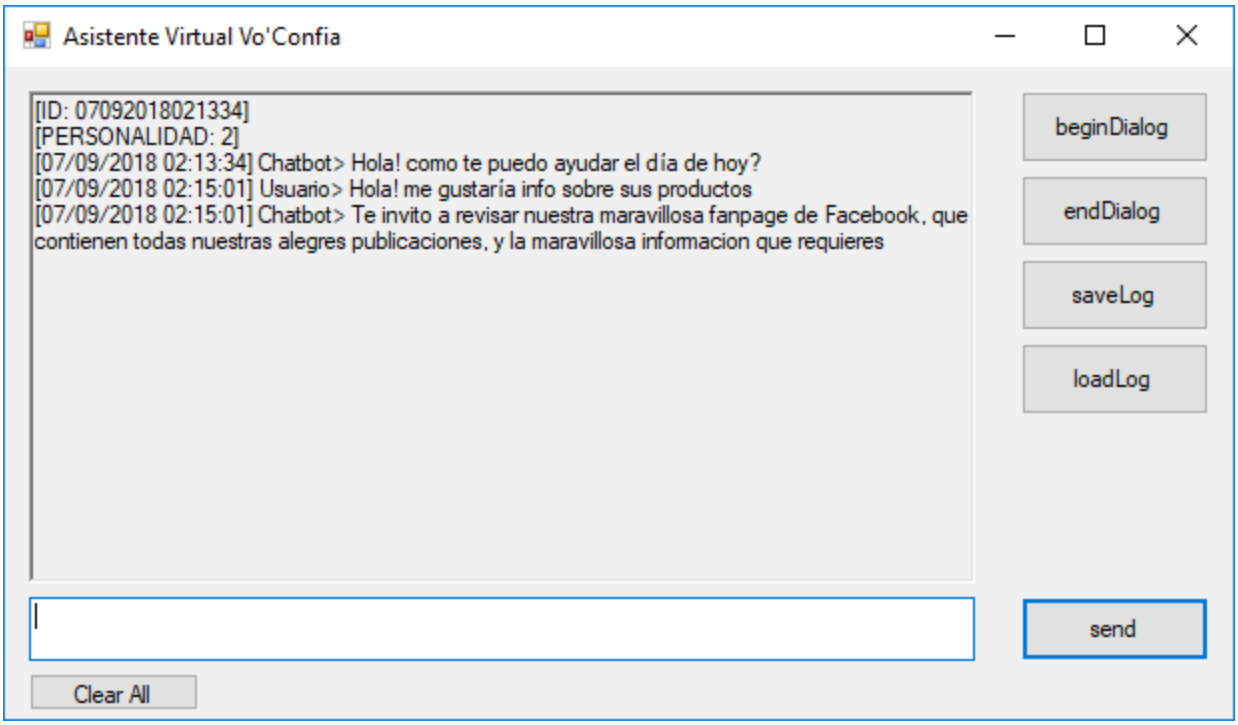

Ilustración 8.- Envió de mensajes

Luego de obtener todo lo que el usuario requiere del chatbot, este puede proceder a finalizar la conversación con el botón **endDialog**, finalizando así la conversación actual, brindándole la hora de termino y un mensaje de despedida.

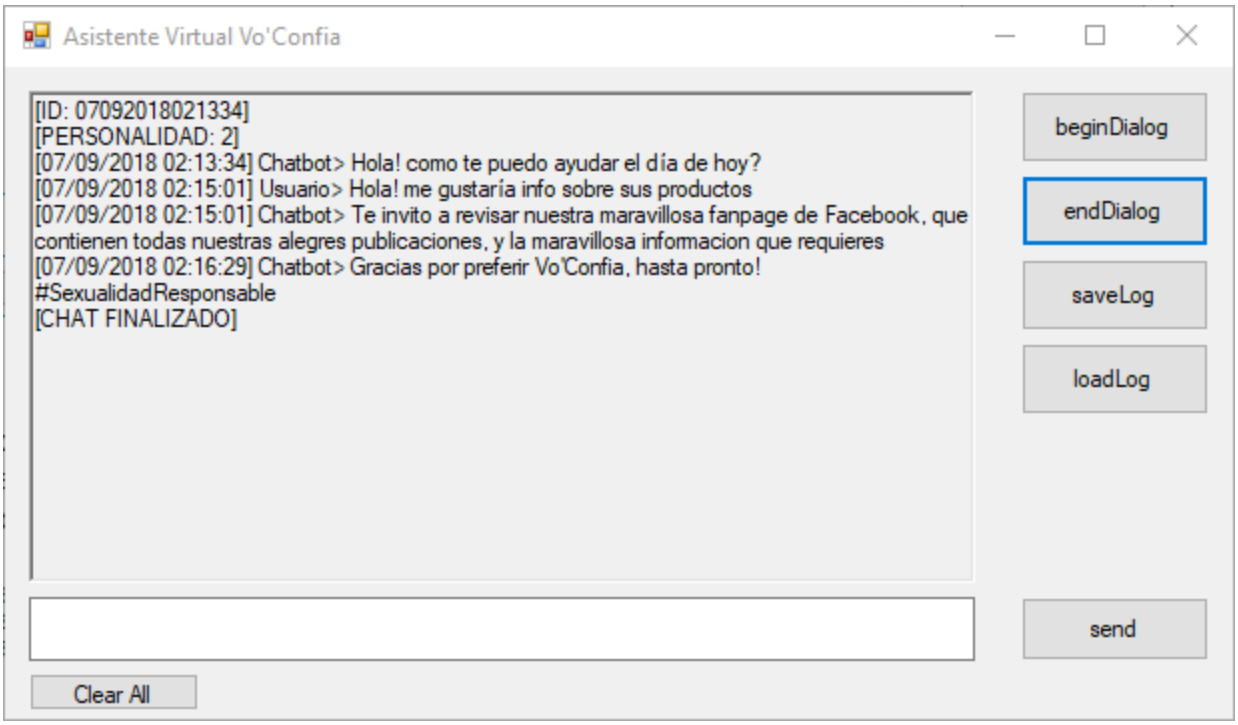

Ilustración 9.- Uso de botón endDialog

Como se ve en la figura anterior, el programa sigue funcionando, ya que el usuario puede seguir interactuando con este, ya sea iniciando una nueva conversación, o también tiene la opción de utilizar las otras opciones que el programa brinda, como el *rate, saveLog* y el *loadLog,* cuyas funcionalidades por pantalla son explicadas más extensamente en el Manual de Usuario.

Además de los resultados funcionales que se acaban de exponer, el resultado final del Diagrama UML fue el siguiente:

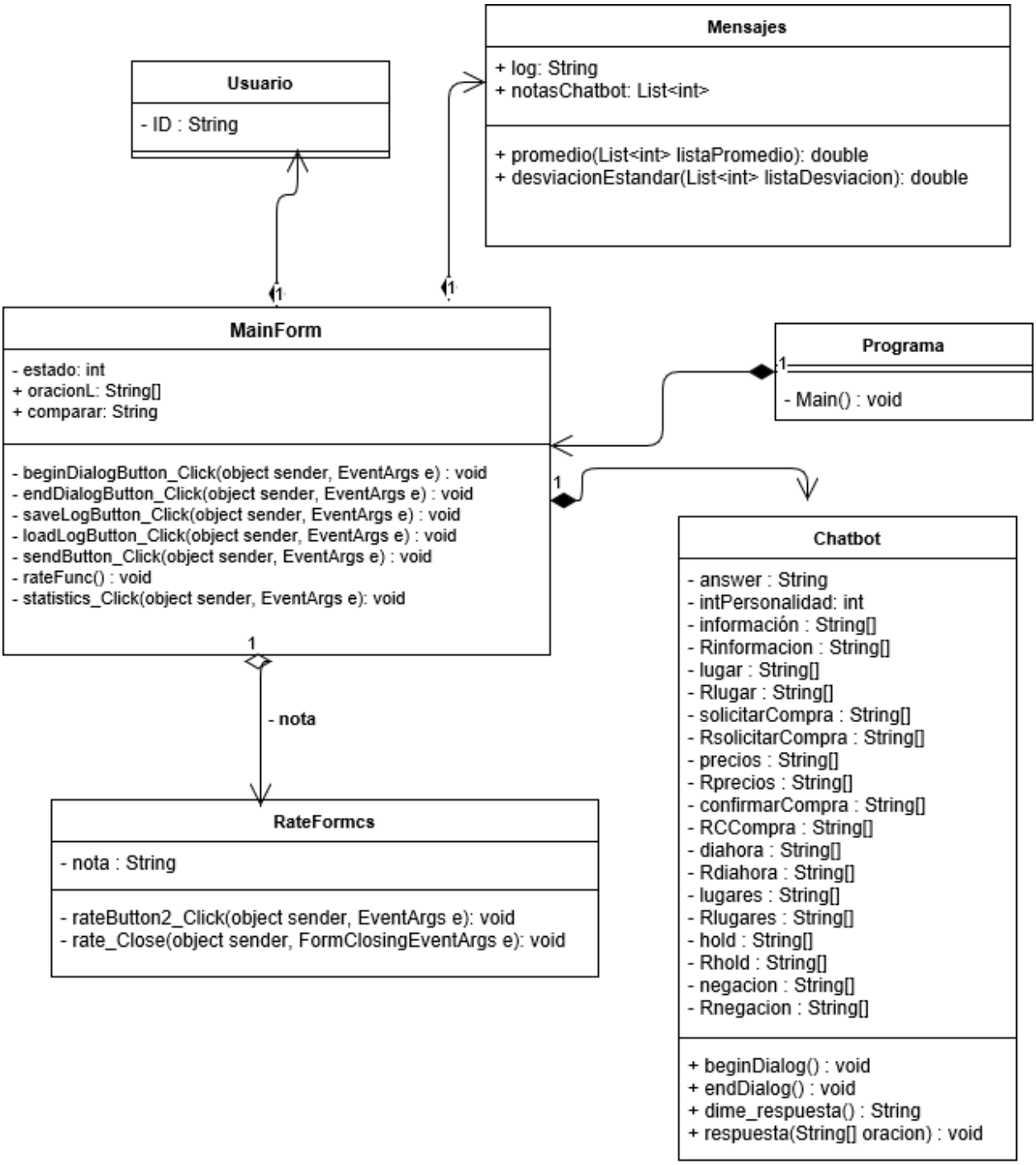

Ilustración 10.- Diagrama UML Final

CAPÍTULO 6. CONCLUSIONES

Dentro de los paradigmas que se estudian en el semestre, el orientado a objetos es quizás el que hace el trabajo más “fácil” el trabajo para un proyecto como este. Ya que los actores dentro de un flujo conversacional están muy bien definidos, por lo que trabajar con la modularización que ofrece trabajar con POO, hace que todo sea más ordenado.

Como lenguaje, C# debe ser de los lenguajes de programación más populares actualmente, por lo que encontrar documentación sobre todo es un trabajo medianamente fácil, el problema es que como popular que es, también tiene muchos conceptos que quizás no se alcanzan a utilizar como herramientas útiles dentro de los proyectos que uno desarrolla. Por ejemplo, para este laboratorio, por temas de tiempo, no se pudo aplicar lo que recientemente se aprendió en teoría como lo son los conceptos de dependencias, herencias, acoplamiento, entre otros. Pero si se ocuparon otras técnicas como las relaciones, los objetos, etc.

Con relación al paradigma orientado a eventos, y comparándolo quizás con el laboratorio N°3, hace que el trabajo sea mucho más ameno ya que ofrece una interfaz gráfica que mejora la experiencia del usuario al momento de ocupar el chatbot programado.

En comparación a la forma de trabajar con paradigmas anteriores, quizás fue levemente más fácil justamente por lo mencionado anteriormente, que es muy fácil encontrar documentación y apoyo al respecto de C# puntualmente, y además que POO y POE hace que este proyecto se haga de manera muy óptima.

Finalmente, se puede afirmar que se desarrolló el chatbot satisfactoriamente y cumpliendo con todos los requerimientos funcionales, además, se puede decir que se ha cumplido a cabalidad el objetivo de este laboratorio que era aplicar y demostrar el conocimiento que se tiene del paradigma orientado a objetos, y del lenguaje C#.

BIBLIOGRAFÍA

- Enunciado laboratorios 1 – 2018. Paradigmas de programación. Profesores varios.
- http://aprendeenlinea.udea.edu.co/lms/men_udea/mod/page/view.php?id=19537
- http://umh1467.edu.umh.es/wp-content/uploads/sites/25/2016/02/Programación-Orientada-a-Eventos.pdf