Sistema Alumno-Mático: Experiencia en el proceso de desarrollo utilizando UML y JAVA

Universidad Católica del Norte
Depto. de Ing. de Sistemas y Computación
Angamos 0610
Antofagasta, Chile

Juan Bekios Calfa * Equipo de desarrollo†

Resumen

El trabajo pretende mostrar el desarrollo de un sistema de apoyo a la docencia llamado Alumno-Mático, el cual permite a los estudiantes, de diferentes sedes, obtener información relevante sobre su permanencia en la universidad. A través de estas páginas explicaremos los aspectos relevantes del sistema, y principalmente definiremos la notación, metodología y lenguaje de programación seleccionados para el desarrollo: UML y Java. Se explicarán cada una de las etapas definidas en el proyecto, dentro del marco del desarrollo orientado a objetos, a través de la descripción de un módulo del sistema denominado Beneficios.

Palabras Claves: UML, Java, casos de uso, diagramas de casos de uso, diagramas de secuencia, diagramas de colaboración, diagramas de clases, contratos, responsabilidades.

1 Introducción

Con el afán de optimizar la gestión de las unidades administrativas y académicas que requieren contar con información rápida y expedita por parte de alumnado, la Vicerrectoría Académica y el Departamento de Ingeniería de Sistemas, de nuestra casa de estudios, concretó la construcción de un sistema de autoconsulta denominado Alumno-Mático. El sistema permite obtener información relevante para el estudiante, las cuales consideran las siguientes alternativas:

- Módulo Curricular: Orientado a captar los requerimientos curriculares del usuario del sistema, tales como, asignaturas por semestre, malla curricular, horarios, estadísticas del alumno, etc.
- Módulo Beneficios: Este módulo agrupa los requerimientos del usuario referente a beneficios económicos obtenidos.

*e-mail: jbekios@socompa.ucn.cl
†Integrantes: Javier Mandiola, Eric Ross, Victor Lema, Vladimir Moreira, Sara Vega y Diego Urrutia

- Módulo Finanzas: Corresponde a la información asociada a las deudas que el usuario ha contraido con la universidad, sean estas por préstamos y/o créditos.
- Módulo Autentificación: Módulo encargado de validar el ingreso de un usuario al sistema.

El desarrollo del sistema tuvo una duración aproximada de cuatro meses. Esto considera una etapa prelimiar de análisis de requerimientos y de dominio; una segunda etapa de diseño donde se enfatiza la asignación de responsabilidades y el diseño de interacciones; y finalmente la implementación, realizada en Java en forma de aplicación.

En la actividad de desarrollo considera un conjunto de actividades incrementales llamadas iteraciones, a diferencia del desarrollo estructurado. Un ciclo de vida iterativo se basa en el agrandamiento y perfeccionamiento secuencial de un sistema a través de *múltiples* ciclos de desarrollo de análisis, diseño, implementación y pruebas.

Como Java es un lenguaje 100% orientado a objetos fue necesario considerar herramientas de ingeniería de software acordes con dicho paradigma, la herramienta de modelamiento elegida fue UML (*Unified Modeling Language*) y la metodología del proceso de desarrollo se organizó de acuerdo a otras técnicas de desarrollo orientado a objetos, MPR[1][Lar99].

En el presente trabajo trataremos en forma resumida cada una de las etapas de desarrollo utilizando UML y el proceso denominado MPR, como también atenderemos el desarrollo basado sobre internet (TCP/IP) y su implementación sobre el lenguaje Java.

2 Características del Sistema

La universidad cuenta en sus instalaciones con varios sistema de información internos, que considera las distintas instancias administrativas involucradas. Entre los más importantes podemos destacar el sistema curricular, el sistema de finanzas y el sistema de beneficios estudiantiles. Cada uno de estos sistemas tienen asociadas sus respectivas bases de datos en ORACLE, de donde se obtiene la información para el Alumno-Mático.

El acceso a las bases de datos corporativas se realiza a través de la red internet utilizada por la universidad. La conexión del sistema Alumno-Mático se logra por medio de Java a través de los *driver* de conexión JDBC nivel 4 facilitados por la misma empresa ORACLE.

Cada Alumno-Mático está compuesto de un procesador pentium II MMX 333A, 32 Mbytes de memoria RAM, 2.1 Gbytes de disco duro, monitor CRT super VGA 10" 0.28 DPI, *touch screen* resistivo, lector de tarjeta de banda magnética, impresora térmica 40 columnas, etc. Como se puede observar, el Alumno-Mático es un PC convencional en el que se puede cargar cualquier sistema operativo compatible como plataforma de desarrollo. Particularmente se utilizaron dos plataformas de desarrollo una basada en Microsoft Windows'98 y otra en LINUX (Red Hat 6.0).

[1]Modelos y procesos recomendados

3 UML, Lenguaje Unificado para la Construcción de Modelos

UML se define como:

Un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software...[BJR97]

El UML es un estándar para construir modelos orientados a objetos, que actualmente es bastante popular entre los desarrolladores de software. El lenguaje fue creado, en 1994, bajo la iniciativa de Grady Booch y Jim Rumbaugh permitiendo combinar las mejores carácteristicas de sus famosos métodos: el de Booch y el OMT(*Object Modeling Technique*). Más tarde se les unió Ivar Jacobson, creador del método OOSE(*Object-Oriented Software Engineering*). En respuesta a una petición de OMG(*Object Management Group*) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato[Lar99].

Se debe considerar que el UML no es una metodología para el proceso de desarrollo, sino que es un estándar para los artefactos y la notación utilizadas en el modelamiento. Por lo que no es raro encontrar recomendaciones de lo que constituye el proceso adecuado en publicaciones aparte de las dedicadas al UML.

3.1 Análisis y diseño orientado a objetos

El principio básico del análisis orientado a objetos es la investigación del problema a fin de perfilar los aspectos relevantes sobre el dominio del problema, identificando y describiendo objetos desde el punto de vista conceptual. El diseño, en cambio, identifica la solución lógica desde el punto de vista de los objetos lógicos de software, teniendo en claro su próxima participación en la implementación con un lenguaje orientado a objetos.

Dentro del proceso de análisis y diseño podemos resumir las siguientes etapas como las más relevantes:

Análisis y diseño orientado a objetos	Documentos relacionados
Análisis de requerimientos	Casos de uso
Análisis de dominio	Modelo conceptual
Asignación de responsabilidades, diseño de interacciones	Diagrama de diseño de clases, diagramas de colaboración

Tabla 1: Proceso de análisis y diseño

3.2 Alumno-Mático y UML

A continuación se describirá en forma muy sucinta las diferentes etapas que fueron realizadas en el proceso de desarrollo del Alumno-Mático utilizando como herramienta de definición, de artefactos y notación, el UML.

Como se mencionó anteriormente el sistema se descompuso en cuatro módulos: curricular, beneficios, finanzas y autentificación. A manera de ejemplo y con el fin de definir los etapas antes mencionadas, se desarrollará como caso de estudio el módulo de beneficios, específicamente el *caso de uso* **matrícula**.

Análisis de requerimientos

Dentro del proceso de desarrollo lo primero es conocer que se quiere del sistema a través de las especificaciones que entrega el usuario. Una vez realizada esta etapa se desarrolla en forma preliminar los **casos de uso**, los cuales son la descripción de los requerimientos, tomando en cuenta la interacción con cada actor en particular[BJR99].

Una vez obtenidos los casos de uso generales, se realizan los **diagramas de casos de uso** que muestran un conjunto de relaciones entre los *casos de uso* y sus actores, los diagramas de casos de uso direccionan la vista estática del caso de uso de un sistema[BJR99].

La siguiente figura muestra el **diagrama de caso** de uso del módulo de beneficios, figura 1.

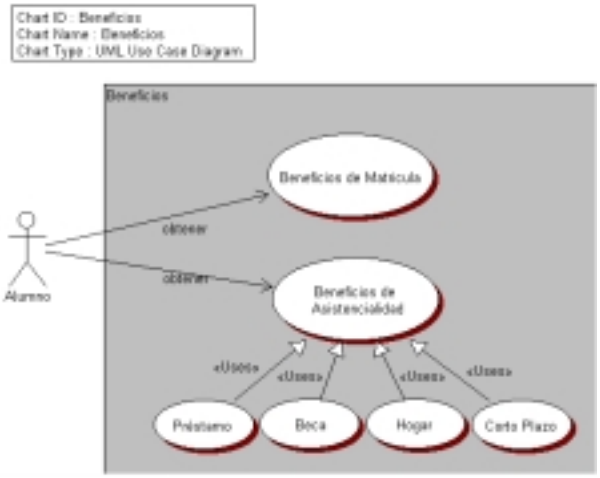

Figura 1: Diagrama de casos de uso, módulo beneficios.

A continación definiremos el **caso de uso** matrícula, se puede observar que la definición se divide en varias partes lo que permite obtener una buena representación del problema describir. En nuestro caso particular las diferentes partes fueron representadas a través de *plantillas* para definir todos los casos de uso del problema, tal como se muestra a continuación:

Caso de Uso: Beneficios de Matrícula (Módulo Beneficios)

Descripción: El alumno desea conocer los beneficios de matrícula obtenidos, sean estos porcentaje de crédito obtenido y/o becas.

Curso:

1. ACTOR: El alumno desea conocer los beneficios de matrícula obtenidos.	2. SISTEMA: El sistema entrega al alumno: 1. Si tiene crédito, el porcentaje asignado, el monto asignado y el estado del pagaré 2. Si tiene beca, el tipo de beca y el monto asignado. 3. El monto total a pagar en el año.
3. ACTOR: El alumno sale del sistema.	

Alternativo:

3. ACTOR: El alumno solicita imprimir.	4. SISTEMA: Realiza impresión de: 1. Si tiene crédito, el porcentaje asignado, el monto asignado y el estado del pagaré 2. Si tiene beca, el tipo de beca y el monto asignado. 3. El monto total a pagar en el año.
5. ACTOR: El alumno retira el papel.	6. SISTEMA: El sistema vuelve a la pantalla inicial.

Nota: Considerar el caso de impresión limitada.

Análisis de dominio

Una vez descritos los **casos de usos**, se debe descomponer el problema en conceptos u objetos individuales: En el proyecto se construyó una tabla con todos los sustantivos, adjetivos y verbos utilizados en las redacción de los **casos de uso**, posteriormente por medio de simples reglas se obtuvo un conjunto de objetos y sus respectivas relaciones, la finalidad de este *modelo conceptual* es de subrayar fuertemente los conceptos de dominio y no las entidades del software.

Una vez obtenido el **modelo conceptual**, a través de un diagrama de clases, se obtienen los **diagramas de secuencia**.

Diagramas de secuencia

La creación de los diagramas de secuencia de un sistema forma parte de la investigación para conocer el sistema; se incluye, dentro del modelo análisis[Lar99]. Estos diagramas permiten explicar gráficamente la interacción entre el actor(usuario) y el sistema. El actor genera un conjunto de eventos dirigidos al sistema. Los sistemas se les trata como cajas negras, separando el curso normal de los eventos de los **casos de uso**. Se consideran eventos alternativos.

En en caso de Beneficios de Matrícula el **diagrama de secuencia** obtenido es el mostrado en la figura 2.

En el **caso de uso** Beneficio de Matrícula se pueden observar los eventos *1:solicitarDeudasActuales()* como primera actividad, y dos eventos alternativos *2:imprimirDeudasActuales* y *2:salirSistema*, estas actividades indican que se puede realizar una o la otra. El número que acompaña a cada evento indica su secuencialidad, los números repetidos indicar eventos alternativos.

Contratos

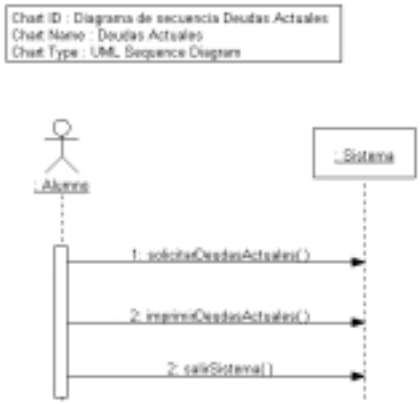

Figura 2: Diagrama de secuencia, caso de uso: Beneficio de Matrícula.

Los contratos contribuyen a definir el comportamiento de un sistema; describen su efecto que sobre él tienen las operaciones.

El lenguaje UML ofrece un soporte para definir los contratos, ya que permite definir las precondiciones y las postcondiciones de las operacionees.

A continuación se presenta la *plantilla* de contrato utilizadas para el **caso de uso** Beneficio de Matrícula.

Beneficios de Matrícula

Parámetros -

Operación **obtenerBeneficioDeMatricula**.

Visibilidad Verdadero.

Parámetro ninguno.

Pre-condiciones Existan instancias de:

- Alumno.
- Carrera.

Post-condiciones -

- Fue creada instancia de Beca.
- Fuen creada instancia de Crédito.
- Fue asociado Alumno y Beca.
- Fue asociado Alumno y Crédito.

Descripción -

Responsabilidad Desplegar porcentaje, monto asignado de crédito y estado del pagaré coorespondiente. Desplegar tipo y monto asigando de Beca.

Excepciones Si no se le asignó o no pidió beca, desplegar mensaje.

Parámetros -

Operación **imprimirBeneficioDeMatricula**

Visibilidad Verdadero

Parámetro Ninguno

Pre-condiciones Existan instancias de :

- Alumno.
- Crédito.
- Beca.

Post-condiciones -

- Fueron destruídas las asociaciones entre Alumno y Crédito.
- Fueron destruídas las asociaciones entre Alumno y Beca.
- Fueron destruídas las instancias entre Beca y Crédito.

Descripción -

Responsabilidad Crear un reporte e imprimir los beneficios de matrícula.

Lo esencial del diseño orientado a objetos es la **asignación de responsabilidades** a los distintos objetos, ya conocidos en la fase de análisis, además de reconocer nuevos objetos que formarán parte de la solución del problema. La **asignación de responsabilidades** se realiza mediante patrones de diseño y gráficamente se representa con diagramas de colaboración.

En esta fase también se crea el modelo de diseño de clases que indica la navegabilidad entre las clases, las operaciones a realizar y la visibilidad de sus atributos.

Arquitectura del sistema

Es parte de la fase de diseño considerar los aspectos relevantes de la implementación en la realización de los elementos de software, para el proyecto se utiliza una arquitectura de tres capas que tiene como finalidad aislar la lógica de la aplicación para convertirla en una capa intermedia bien definida.

En resumen, los elementos de software fueron divididos en tres capas:

- **Capa de Presentación:** Esta capa se encarga de visualizar al usuario la aplicación a través de ventanas y gráficos. Realiza relativamente poco procesamiento en la aplicación; las ventanas envían a la capa intermedia peticiones de trabajo.
- **Capa de Dominio:** Aquí se concentra la lógica de la aplicación, y donde se realizan la gran mayoría de las transacciones.
- **Capa de Servicio:** Todas las consultas a base de datos son implementadas en este nivel. Su misión es obtener el resultados de las consultas, en cualquier base de datos, y transformarlas en estructuras de datos de fácil uso para la *capa de dominio*.

La figura refArqCapas muestra el **diagrama de arquitectura de capas** utilizado en la implementación del Alumno-Mático.

Figura 3: Diagrama de Arquitectura de Capas.

Asignación de responsabilidades

En el trabajo de análisis se propuso dar el soporte suficiente para la fase de diseño orientado a objeto. En esta fase se retoma las definiciones obtenidas del análisis pero se asignan las responsabilidades a los objetos y se realiza el diseño de los **diagramas de colaboración**

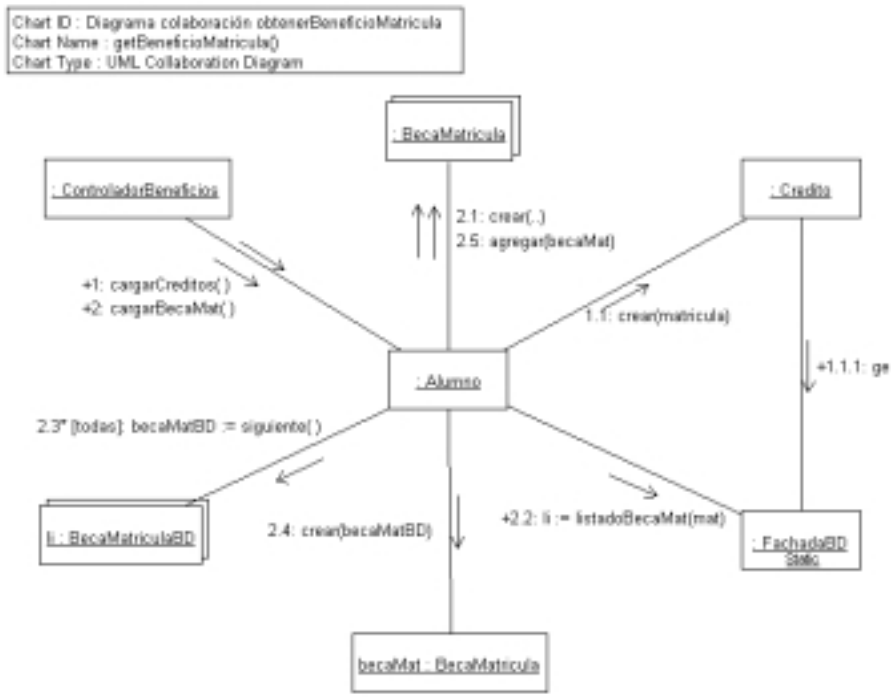

Figura 4: Diagrama de Colaboración Beneficios de Matrícula.

Diagramas de colaboración:

Un diagrama de colaboración es un bosquejo de interacción que enfatiza la estructura organizacional de los objetos que envían y reciben mensajes. Un diagrama de colaboración muestra un conjunto de objetos, relacionados entre sí, y los mensajes enviados y recibidos por ellos. Los objetos pueden ser los típicos o instancias anónimas de las clases, pero también representan instancias de otras cosas, como colaboraciones, componentes y nodos. El uso de los diagramas de colaboración ilustra una vista dinámica del sistema[BJR99].

En el caso particular del sistema Alumno-Mático, el **diagrama de colaboración** del módulo beneficios, **caso de uso** Beneficio de matrícula, es el mostrado en la figura 4 y la asignación de responsabilidades es definida a continuación.

Caso de uso: Beneficios de Matrícula
Responsabilidad: getBeneficioMatricula(), muestra los beneficios de matrícula asignados al alumno, figura 4. *Objeto ControladorBeneficios:* Patrón controlador, objeto creado para cumplir responsabilidades de eventos al sistema.
Responsabilidad: cargarCreditos(), crear y asignar una instancia de credito a alumno, figura 4. *Objeto Alumno:* Patrón creador, objeto que usa al objeto crédito.
Responsabilidad: cargarBecaMat(), crear y asignar instancias de becaMatricula a alumno figura 4. *Objeto Alumno:* Patrón creador, objeto que usa y agrega al objeto becaMatricula.
Responsabilidad: listadoBecaMat(), obtener un listado de las becas de matrícula obtenidas por el alummo, figura 4. *Objeto FachadaBD:* Patrón fachada, objeto creado para soportar servicios heterogéneos para bases de datos.

Una vez obtenidos todos los **diagramas de colaboración**, para cada uno de los módulos, se puede realizar el **diagrama de clases de diseño**.

El **diagrama de clases de diseño** muestra el tipo de atributos de las clases, las operaciones asociadas y la visibilidad de los atributos y operaciones. Además indica la navegabilidad entre las clases. Es en este momento donde ya se han identificado los elementos de software, los cuales serán implementados en el lenguaje que corresponda, en el sistema Alumno-Mático el lenguaje escogido fue Java.

4 Implementación

Como se anticipó anteriormente, la implementación fue realizada integramente en lenguaje Java.

En esta sección profundizaremos la conversión de *componentes de software* a código Java. Para esto se utilizará el ejemplo antes visto del **caso de uso** Beneficios de Matrícula.

Como se estudio anteriormente los *componentes de software* son definidos en la etapa de diseño a través de los **diagramas de colaboración**, tal como se muestra en la figura 4. El **diagrama de colaboración** nos permitirá construir las **clases de implementación**, para esto utilizaremos la secuencia mostrada en dichos diagramas y la realización de la implementación final en Java.

Etapas de codificación

Caso de Uso: Beneficios de Matrícula

Aquí construiremos paso a paso cada uno de los métodos involucrados en los **diagramas de colaboración** utilizando la secuencia mostrada en la figura 4.

Paso 1: Responsabilidad getBeneficioMatricula()

El código Java generado es el siguiente:

```
public class ControladorBeneficios extends Controlador
{
.
.
    public void getBeneficioMatricula()
    {
        getAlumno().cargarCredito(); \\+1:
        getAlumno().cargarBecaMatricula(); \\+2:
    }
.
.
}
```

Como se puede observar en la figura 4, el objeto anónimo *ControladorBeneficios* es un patrón controlador por lo ejecuta la responsabilidad correspondiente a través del método getBeneficioMatricula() que a su vez activa los métodos del objeto anónimo *Alumno*. La instrucción *getAlumno()*

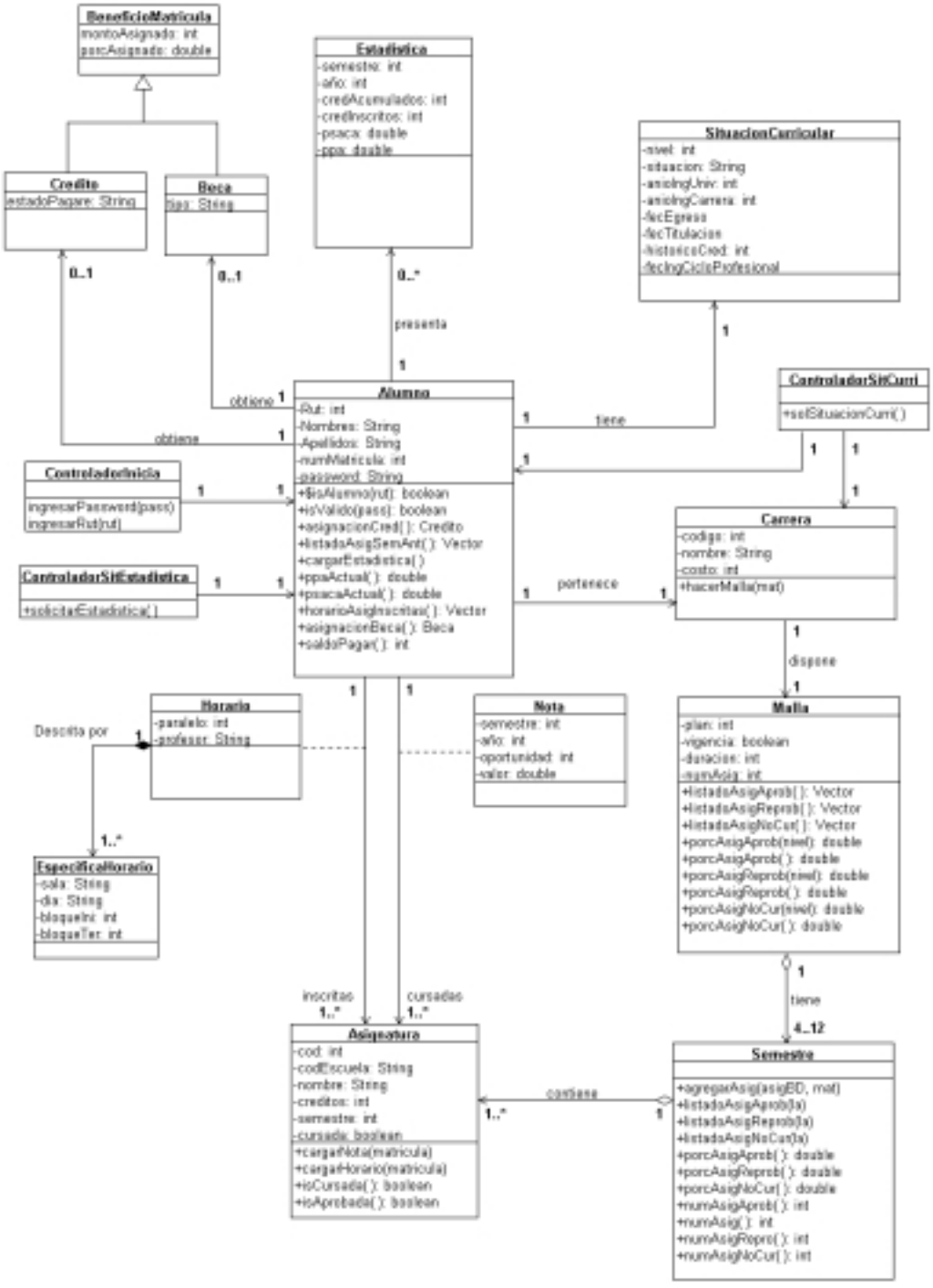

Figura 5: Diagrama de Clase de Diseño.

es un método *heredado* de la clase *Controlador* y que entrega, como resultado, una referencia del objeto *Alumno*.

Esto significa que los métodos *cargarCreditos()* y *cargarBecaMat()* son responsabilidad del objeto *Alumno* tal como se muestra a continuación:

```
public class Alumno
{
...
   public void cargarCredito()
   {
     theCredito= \\1.1:
        new Credito(FachadaBD.getDescCred(matricula));
        \\1.1.1:
   }
   public void cargarBecaMatricula()
   {
     Vector listBecasMat =
        FachadaBD.listadoBecaMat(matricula);
        \\+2.2:
     if(listBecasMat!=null){
        theBecasMatricula =
          new Vector(listBecasMat.size());
          \\2.1:
     for(int i=0;i<listBecasMat.size();i++){
        BecaMatriculaBD becaBD=
            (BecaMatriculaBD)listBecasMat.elementAt(i);
            \\2.3*[todas]: y 2.4:
        BecaMatricula beca=
            new BecaMatricula(becaBD);
        System.out.println(beca);
        theBecasMatricula.addElement(beca);
        \\2.5:
        }\\Fin for
     }\\Fin if
   }\\Fin cargarBecaMatricula
...
}\\Fin clase Alumno
```

Las responsabilidades del objeto anónimo *Alumno*, *cargarCreditos()* y *cargarBecaMat()*, activan un conjunto de acciones que se describen a continuación.

Paso 2: cargarCreditos()

Carga el crédito asignado para un alumno del pago de arancel universitario.

cargaCreditos() indica al objeto anónimo *Alumno* que debe crear una instancia anónima llamada *Credito*; observar el **diagrama de colaboración**, figura 4, y el código de implementación que se representa con el identificador 1.1:. A su vez *Credito* carga del objeto anónimo *FachadaBD* el tipo de descripción de crédito directamente de la base de datos, el cual se representa con el identificador +1.1.1:, tal como se muestra en la figura 4 y el código fuente de la clase *Alumno*.

Paso 3: cargarBecaMat()

Carga los diferentes tipos de Becas obtenidas por el alumno para el pago de sus estudios.

cargarBecaMat() activa en el objeto anónimo *Alumno* la creación de un listado de objetos que contendrán los diferentes tipos de Becas, identificador 2.1:. Luego se consulta al objeto anónimo *FachadaBD* y se obtiene el conjunto de becas desde las bases de datos, identificador +2.2:. Finalmente se recorre el listado de objetos *BecaMatriculaBD*, identificador 2.3, creando y agregando estos al objeto *BecaMatricula*, identificador 2.4 y 2.5 que se observan en el **diagrama de colaboración** de la figura 4 y del código Java para el objeto *Alumno*.

5 Conclusiones

El uso de metodologías en el desarrollo orientado a objetos permite organizar en forma efectiva al grupo de modelado e implementación, reduciendo el costo de comunicación entre ambas partes. En nuestra experiencia, el grupo completo de desarrollo se dividio en dos, un grupo que se encargó de la etapa de análisis y diseño; y otro que trabajo sobre la implementación. Los resultados fueron importantes considerando que cada iteración demoró aproximadamente de dos a tres semanas.

El uso de iteraciones en el proyecto permitió obtener resultados finales en el corto plazo. Esto sin duda mejoró la relación con los usuarios (clientes) ya que cada tres semanas se mostrarón prototipos del sistema, lo cual permitío la revisión y depuración en beneficio del resultado final.

Se observó un ahorro sustancial en lineas de código, comparado con otros paradigmas de desarrollo, además de permitir una fácil administración y mantención del sistema.

El uso de una arquitectura de tres capas, permitío desacoplar el sistema por lo que se pudo trabajar con diferentes tipos de bases de datos. Además la interfaz no depende de la lógica del sistema, esta cualidad nos permitirá, en un futuro próximo reutilizar el código para generar este mismo sistema sobre WEB, a través del uso de *Servlet*.

Toda la implementación del software que soporta el Alumno-Mático fue desarrollado en Java, por lo que no fue necesario utilizar sólo una plataforma de desarrollo. Esto es bastante importante dentro del desarrollo ya que contribuyó a elegir la mejor plataforma de trabajo, además de permitir a distintos tipos de desarrolladores trabajar en la plataforma que más le acomode.

Al construir este sistema, hemos podido constatar que el grupo de desarrollo no sufrio un gran estrés dentro de las diferentes etapas de construcción. Esto destaca los puntos antes señalados ya que este estilo de programación permite ordenar y organizar el proyecto en forma más natural que otros estilos de programación conocidos. Se debe considerar que el grupo de desarrollo estuvo compuesto principalmente por alumnos de pre-grado, sin conocimientos específicos sobre el tema.

Referencias

[BJR97] G. Booch, I. Jacobson, y J. Rumbaugh. *The UML specification documents*. Rational Software Corp., http://www.rational.com, 1997.

[BJR99] G. Booch, I. Jacobson, y J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison Wesley Longman, Inc., Estados Unidos, tercera edición, 1999.

[Lar99] Craig Larman. *UML y Patrones, introducción al análisis y diseño orientado a objetos*. Prentice Hall, México, México, primera edición, 1999.