UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Aplicación de métodos de análisis y diseño orientados a objetos para la construcción de un sistema de localización de objetos celestes

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

Yoatzin Solis Zúñiga
Luis Gilberto Osorio Morales

DIRECTOR DE TESIS

Dr. Jesús Savage Carmona

Ciudad Universitaria, Cd. Mx., 2004

Al soporte de todo lo que soy y existe en mi vida, Dios.

A mi abuelo Joel Zúñiga, un gran hombre y padre. Mi primer contacto con la ingeniería.

A mi madre, la mujer más admirable que conozco, quien me enseñó a desarrollarme como un mejor ser humano, me enseñó cómo conseguir y luchar por lo que quiero. Gracias por tu amor y apoyo incondicional.

A mi hermana, la pequeña constructora de su propia vida y un ejemplo de fuerza, coraje, determinación. Gracias.

A Luis, quien me hace querer ser mejor cada día, quien esta conmigo siempre, quien al hacerme reír hace que todo lo triste desaparezca y solo quedemos él y yo. Porque nos une más que el amor, la lealtad, la amistad y el alma. Gracias.

A mi "vaquita", el mejor amigo, el apoyo incondicional, el entrenador, el *sen-sei*. Porque ya eres una parte imborrable del corazón de Yovi.

A mi familia, a Maru, Montse, Noemí, Joel, Carmen, Josefina, Rubén, Javier.

A mi corazón peludo, mi Hagi, por su amor incondicional y sin reservas. A sus amigos Figo, Chiquita, Scam y Chucky.

A los mejores amigos, porque sé que lo serán siempre. A Claudia, Paco, Manolo, Mau, Sama, Ana, Pablo, Rosi, Vicente, Alejandra, Lalo, Lety y Carlos.

Al Avellaneda, mi nueva ilusión. Gracias Auribel y Paulina.

A la porra mugido: Daniel, Natzin, Rodrigo, Mumu, Mau, Carlos, Karla y Luis.

A los Pumas, campeones y el mejor equipo de fútbol. Al mejor defensa central Joaquín Beltrán.

A los profesores que dejaron más que conocimiento en mi desarrollo profesional: Marcela, Olivia, Julieta, Sergio, Pablo.

A *Sidereus nuncius* y todo la banda de SAFIR, por ser inmejorables amigos y enseñarme que hay que ver al cielo de vez en cuando. A Laura, Tere, Pablito, Tania, Juan Pablo, Araceli, Eduardo, Marilu, Germán, Paola, Bibi, Farah, Fernando, Carlos, Paco y Samantha.

A Enginet un sueño cumplido, gracias por todo Manuel, Oscar, Rodrigo, Toño, Luisillo, Ceci, José Luis y Alejandro.

A la Facultad de Ingeniería.

A mi querida universidad, la Universidad Nacional Autónoma de México.

Yoatzin

A Dios por darme la gran oportunidad.

A Yoatzin, quien es mi motivo y apoyo todos los días para realizar todas las empresas, todas las aventuras, todos los sueños. No imagino día sin ella.

A mis padres Gilberto y Rosario, por que ellos fueron mis mejores maestros, a ellos les debo simplemente todo, absolutamente todo lo que soy. Los admiro y los quiero muchísimo.

A mi abuelita Lupita por que su ternura y cariño me hicieron siempre sentirme seguro.

A toda mi familia, mi hermana Toña, mi tía Beatriz, mi prima Jenevy, mi abuelito José. Por que siempre fueron un apoyo y siempre han creído en mí.

A la familia Pintado: Lety, Carlos, Dany y Chicho. Por acogernos siempre en su casa y compartir con nosotros parte de su vida.

A Lalo, quien ha sido, de mis amigos, el espectador más viejo de esta historia y siempre ha estado pronto a extender la mano y arrancar una risa. Gracias Bro.

A Mumu, por que un amigo como él es una bendición del cielo.

A Mau y Sama por ser los amigos entrañables de la familia.

A Vicente y Gabriel por compartir un año conmigo, por hacerme feliz, por ser su amigo.

A toda la banda *Sideral*: Tere, Laura, Lalo, Marilú, Pillín, Pablito, Tania, Juan Pablo, Araceli, Germán, Paola, Manolo, Oscar, Farah, Bibi, Carlos, Mumu, Mau, Paco. Simplemente han cambiado mi historia, de todos he aprendido muchísimo, de todos estoy muy orgulloso. Completaron con su gran humanismo mi formación universitaria.

A los *Misterios de la Corregidora*: Ale, Carlos, Urko, Alejandro, Lalo. Compañeros de lucha, evolución, revolución e ilusión. Amigos inmejorables.

A Jose Luis, amigo, profesionista y ser humano excepcional que me ha demostrado que la superación constante es posible.

A mis amigos que hicieron posible el sueño de *Enginet*: Manolo, Rodrigo, Oscar, Toño, Yoatzin, Juan Carlos, Ceci y Ale. Gracias por enseñarme la lección más grande de mi vida.

A mis amigos de la prepa: Sergio, Armando, Arturo, Charly. Juntos hemos logrado construir una amistad duradera y de apoyo mutuo. Gracias.

A mi queridísima Facultad de Ingeniería, a todos sus maestros y a los Pumas. Por que ser universitario es algo que queda tatuado en el corazón.

Luis

Agradecimientos

A nuestros padres, por apoyarnos, por alentarnos, por darnos su ejemplo y su inmenso cariño. Por esperar pacientemente este día.

A nuestros profesores, por su gran contribución a nuestro desarrollo profesional. Por darnos en las aulas más que conocimientos.

A SAFIR, por complementar nuestra mirada y enseñarnos qué existe en el Universo. Por ser la idea principal y el porqué de este proyecto.

A Alejandro Farah, Javier Godoy y Jose Luis Iturbide, por su colaboración en la realización del presente trabajo de tesis.

A Rodrigo Morales por su ayuda y dedicación en la corrección de estilo.

A Antonieta Osorio por su cariñosa colaboración en la edición.

Al Dr. Jesús Savage Carmona, por su aliento, por su guía y su apoyo en este proyecto.

A la Facultad de Ingeniería, por darnos conocimientos invaluables, por adoptarnos como hijos por cinco años, por ser una gran institución educativa y enseñarnos como ser mejores profesionistas.

A la Universidad Nacional Autónoma de México, nuestra *alma mater*, porque con su pluralidad, sus grandes escuelas, su gran nombre y espíritu, nos ha tatuado el alma y nos hace estar orgullosos de ser universitarios.

Índice

Ilustraciones

Tablas

1 Introducción

El presente trabajo de tesis pretende exponer una solución al problema de creación de un programa que será utilizado por los miembros de la Sociedad Astronómica de la Facultad de Ingeniería (SAFIR) para controlar el movimiento de un telescopio y realizar observaciones astronómicas. Dicho programa deberá ser un desarrollo a la medida que cumpla con todos los requerimientos necesarios para realizar las tareas de observación y, además, deberá ser de fácil adaptabilidad y escalabilidad, de tal manera que los miembros de la Sociedad (aún aquellos que no pertenecen a las áreas afines a la ingeniería en computación) puedan realizar, de manera relativamente sencilla, modificaciones al mismo para cumplir con nuevos requerimientos; adaptarlo a nuevos dispositivos (telescopios distintos, cámaras CCD, etc.) o bien, expandir sus funciones para nuevos proyectos como el acceso remoto vía la Internet.

La solución propuesta se centra en dos puntos: 1) El análisis y diseño orientado a objetos utilizando como herramienta para modelado el UML para lograr la máxima reutilización de código; y 2) la construcción utilizando un lenguaje orientado a objetos interpretado para reducir la curva de aprendizaje y facilitar el mantenimiento.

Los resultados que se esperan al finalizar el proyecto son: que se cuente con un diseño de programa que cumpla con las metodologías orientadas a objetos que sirvan de guía para el desarrollo de proyectos similares; tener la documentación necesaria escrita en un leguaje común para que pueda ser modificado por cualquier persona interesada; aportar componentes de *software* que sirvan para ser utilizados en otros desarrollos y fácilmente modificables para interactuar con otros dispositivos y, por último, esperamos brindar una solución basada en el lenguaje interpretado *Python* para aquellos desarrollos de proyectos estudiantiles que requieren una herramienta de desarrollo potente, portable y de fácil aprendizaje.

1.1 Antecedentes

1.1.1 La Sociedad Astronómica de la Facultad de Ingeniería, SAFIR

La Sociedad Astronómica de la Facultad de Ingeniería (SAFIR) nace de la inquietud de un grupo de alumnos de la Facultad de Ingeniería por instalar un observatorio para uso de la comunidad, en la cúpula de la torre B del edificio principal. Históricamente, era ahí dónde se impartían clases de observación astronómica a los futuros ingenieros y, por tanto, era el lugar idóneo por tradición para alojar la nueva generación de ingenieros con afición a la astronomía.

La idea fue llevada al entonces Director de la Facultad, Ing. José Manuel Covarrubias Solís, quien, preocupado tal vez por la seriedad de la propuesta, pidió a los alumnos que demostraran la autenticidad de su interés organizándose para realizar alguna actividad que sirviera para reunir a miembros de la comunidad que compartieran la misma afición. Entonces nació la revista de divulgación astronómica *Sidereus nuncius*, nombrada así en honor al célebre Galileo Galilei quién nombró de la misma manera a la gaceta donde diera a conocer sus descubrimientos, revolucionarios para su época, acerca de las lunas de Júpiter.

La revista fue ganando popularidad y entonces SAFIR vio su nacimiento un viernes 26 de septiembre de 1997 en el marco de la X Semana SEFI. La Sociedad fue formalmente constituida por el Ing. José Manuel Covarrubias Solís y fungiendo como Secretario de servicios Académicos el Ing. Pablo García y Colomé.

La sociedad llegó a reunir en los primeros meses a más de 800 asociados, convirtiéndose así en la sociedad astronómica con mayor número de miembros de México.

Pronto comenzó a ser identificada por los miembros de la comunidad quienes participaban activamente en las actividades que se ofrecían, tales como: el taller de construcción de telescopios, las clases de observación, las visitas al observatorio de Tonanzintla, entre muchas otras.

Además, SAFIR supo ganar la confianza y el agrado de reconocidos astrónomos e ingenieros como Julieta Fierro y José de la Herrán, (entre muchos otros) quienes reconocieron el trabajo de divulgación que se venía realizando.

Se había gestado de esta manera un nacimiento exitoso y comenzaba una larga carrera de luchas constantes para ver su objetivo primario realizado: La instalación de un observatorio en la vieja cúpula de la Facultad de Ingeniería.

1.1.2 El Proyecto para instalar el observatorio

Como se mencionó en la sección anterior, el principal objetivo de la naciente SAFIR era la instalación de un observatorio en la cúpula de la torre B del edificio principal. La primera fase del proyecto fue liderada por uno de los miembros fundadores, Fernando Quirós, quien con la colaboración de otros miembros fundadores como Alejandro Farah y Pablo Álvarez concibieron al proyecto de la siguiente manera:

"Uno de los proyectos de la Sociedad Astronómica de la Facultad de Ingeniería de la UNAM (SAFIR) es poner en funcionamiento un observatorio astronómico en las instalaciones de la Facultad de Ingeniería en Ciudad Universitaria. Dicho observatorio permitirá a los miembros de SAFIR realizar prácticas de observación: seguimiento de planetas, identificación de objetos astronómicos importantes, estudio de la superficie lunar, observación de cometas, astrografía, etc.

El telescopio que se utilizará es de tipo newtoniano de seis pulgadas de diámetro, colocado en una montura ecuatorial, los movimientos de posición y de seguimiento de la bóveda celeste serán efectuados por motores y controlados por una PC, a través de un software diseñado para la comunicación del sistema con el usuario. La PC almacenará una base de datos con las coordenadas de objetos observables.

El proyecto se llevará a cabo siguiendo dos directrices: la primera es que el sistema instalado será modular, de tal forma que podrán agregarse nuevas funciones y dispositivos más complejos posteriores a la puesta en funcionamiento inicial; la segunda es que la realización del proyecto debe mostrar la aplicación de las diferentes ramas de la ingeniería en el desarrollo de la astronomía. Por ésta razón, el trabajo global se divide en proyectos individuales definidos en función del área de la ingeniería a la que corresponden..."

Los equipos que se formaron para resolver el problema de la instalación del observatorio fueron los siguientes:

- Equipo de diseño mecánico. Cuyo objetivo era adaptar la montura ecuatorial existente al control dirigido por computadora, instalando los motores y sensores en las articulaciones de la montura, además de instalar la montura modificada en la cúpula de observación.

- Equipo de electrónica. Que se encargaría de los mecanismos de posicionamiento y control utilizando un microprocesador HC11.

- Equipo de computación. Que sería el encargado de diseñar el programa de computadora para obtener las coordenadas del objeto celeste a partir de la posición absoluta alojada en la base de datos, la interfaz de la computadora con la tarjeta controladora y la calibración de la posición inicial del telescopio.

El telescopio que se pensó utilizar entonces, fue prestado por el Dr. Jesús Savage, Jefe del Departamento de Ingeniería en Computación. Dicho telescopio era reflector tipo newtoniano marca MEADE, con espejo primario de 6 pulgadas de diámetro (152.4 mm.), distancia focal de 1220 mm y relación focal F8, lo que le permitía alcanzar hasta 350 aumentos con el lente objetivo indicado. La montura era de tipo ecuatorial y contaba con un motor para el seguimiento de la bóveda celeste.

El proyecto tal como fue concebido estaba listo para empezar, sólo faltaba algún patrocinio para poder adquirir los dispositivos necesarios.

En 1999, se encontró una opción para llevar a cabo el proyecto. La IEEE en conjunto con AT&T organizó y lanzó una convocatoria para proyectos estudiantiles; los cuales apoyaría económicamente con hasta USD1000.

El proyecto para la instalación del observatorio de SAFIR fue inscrito tal y como se concibió en un principio, con los tres equipos: mecánico, electrónico y computación. Y para el mismo año de 1999 los miembros de la Sociedad y los participantes activos del proyecto recibieron con agrado la noticia de que el proyecto “cúpula”, como algunos lo llamaban, había sido seleccionado para recibir el patrocinio.

El equipo de electrónica concluye algo que será definitorio para las decisiones posteriores. Después de haber estudiado las opciones para la compra de codificadores y circuitos para controlar los motores, se dieron cuenta que se necesitaba un presupuesto muy alto sólo para comprar éstos dispositivos.

Entonces el equipo de trabajo tomó una decisión importante: El dinero para el patrocinio del proyecto sería invertido en un telescopio para poder realizarle las adecuaciones necesarias y ser instalado en la cúpula de observación.

Mientras todo esto sucedía, algunos miembros fundadores analizaban las estructuras del edificio B de la Facultad de Ingeniería y gestionaban para que la dirección realizara las adecuaciones necesarias a fin de tener la cúpula en condiciones óptimas para comenzar a utilizarla. Sin embargo, aún cuando los trabajos de mantenimiento ya habían comenzado, el equipo de trabajo recibió una noticia desalentadora: la torre B no albergaría al proyecto, ya que, la cúpula ahí instalada estaba a cargo de la división de Ingeniería Civil, y justo en aquel momento estaban pensando en reacondicionarla para llevar a cabo uno de sus proyectos.

Era una época que parecía tornarse muy complicada, no sólo para SAFIR sino para la comunidad universitaria en general. Para el mes de abril la UNAM veía cerradas repentinamente sus instalaciones y el paro estudiantil más largo en su historia comenzaba.

Para el proyecto fue un periodo de inactividad total, no se contaba con instalaciones para trabajar; varios miembros del equipo de electrónica habían concluido sus estudios universitarios y se incorporaban ahora a la vida laboral o a realizar estudios de posgrado en el extranjero; no había telescopio para las pruebas. En fin, el proyecto parecía haber terminado.

Sin embargo, durante la huelga, las actividades universitarias encontraron sedes alternas en donde, parcialmente, realizarían su trabajo. Y también los miembros de SAFIR y los participantes del proyecto cúpula que quedaban reanudaron su lucha. Durante la huelga se visitó al nuevo Director de la Facultad de Ingeniería, M. en C. Gerardo Ferrando Bravo para hacer de nueva cuenta la petición de la cúpula; un comité formado por Yoatzin Solís, Pablo Álvarez, Francisco Tovar, Luis Osorio y el Dr. Jesús Savage (quién en calidad de docente respaldaba académicamente el proyecto), se entrevistó en distintas ocasiones con el Director y el jefe de la división de Ingeniería Civil. Después de algunas juntas para tratar de conciliar a las partes, dónde incluso se planteó la posibilidad de compartir la cúpula, el director tuvo que tomar una decisión: La cúpula del edificio B seguiría a cargo de la división de Ingeniería Civil y, una vez que se recuperaran las instalaciones, la dirección haría los arreglos necesarios para reubicar al futuro observatorio de SAFIR.

La llamada huelga estudiantil terminó en febrero del 2000 y para marzo del mismo año los miembros de SAFIR y su nueva mesa directiva se presentaban de nueva cuenta ante el M. en C. Gerardo Ferrando para acordar en qué lugar se instalaría el observatorio. El Director autorizó entonces que se iniciaran los trabajos necesarios para que el edificio A albergara el observatorio de SAFIR. Tal y como se había acordado, el observatorio era reubicado.

Para septiembre del 2000, Javier Godoy, miembro fundador de SAFIR, es el encargado de comprar el telescopio que sea más adecuado para el observatorio. El telescopio que se adquirió, gracias al premio de la IEEE y de algunas contribuciones personales de miembros fundadores, fue un *Celestron* de la línea *Celestar* que contaba con una óptica Schmidt-Cassegrain y cuyo espejo tenía un diámetro de 8 pulgadas. Además, el *Celestar 8* contaba con un motor integrado de fábrica para el seguimiento y podían acoplarse algunos otros dispositivos como el motor de declinación y un control para ajuste fino.

El equipo de computación, con ayuda de Alejandro Farah y Francisco Tovar, reanudó sus trabajos y comenzó a realizar la caracterización del telescopio que se utilizaría. Además, ya que no se contaba con equipo de electrónica, se decidió incluir la parte del control dentro de los objetivos del equipo de computación. Se volvió a plantear el problema y se decidió no utilizar una tarjeta controladora y en su lugar utilizar la misma computadora para controlar los motores.

Los trabajos para adecuar la nueva cúpula duraron cerca de un año. Se limpió y aceitó la cúpula; se realizó la obra civil para abrir el techo del edificio A e instalar la cúpula así como la instalación de escaleras, barandales, alumbrado etc. Todos los trabajos de acondicionamiento corrieron a cargo de la Facultad; sin embargo, los miembros de SAFIR siempre participaron con entusiasmo en cualquier tarea en la que fuera necesaria. Para febrero del 2001 los trabajos estaban casi terminados y el día de la instalación del observatorio se veía muy cerca. La ilusión de los miembros fundadores concebida en 1997 y heredada a los nuevos miembros, parecía por fin volverse realidad.

En el marco de la 1ª Jornada Astronómica que se llevó a cabo del 7 al 9 de marzo, se inauguró el Observatorio de SAFIR, con el apoyo de la Facultad de Ingeniería. La ceremonia se efectuó el viernes 9 de marzo en la Sala del Consejo Técnico, y estuvo presidida por el director de la Facultad, el M. en C. Gerardo Ferrando Bravo. En el acto también estuvieron presentes el Dr. Arcadio Poveda Ricalde, la Dra. Silvia Torres, el Dr. Dante Morán Zenteno; los ingenieros José R. de la Herrán y Pablo García y Colomé, asesores y miembros de la Sociedad; y Francisco Javier Tovar López, presidente de la mesa directiva SAFIR.

El objetivo parecía haberse logrado y todo estaba listo para iniciar el desarrollo del programa que permitiría a los usuarios del observatorio controlar los movimientos del telescopio y sentar las bases para futuros desarrollos.

2 Formulación de la problemática

La Sociedad Astronómica de la Facultad de Ingeniería (SAFIR), decidió instalar un observatorio para uso de sus miembros y de la comunidad de la Facultad de Ingeniería en general, donde se pudieran realizar prácticas de observación y algunos estudios básicos de astronomía.

En el observatorio de SAFIR se instalaría un telescopio que realizaría los movimientos de posición y seguimiento de manera automática, mediante motores conectados a una computadora personal que serían controlados a través de un programa. El programa serviría de interfase entre el usuario y el sistema de control, además de permitir la consulta a una base de datos que almacenaría los datos característicos de los objetos observables en la bóveda celeste.

El problema en general, el de mover el telescopio del observatorio de SAFIR de manera automática, fue concebido en tres partes: una parte mecánica encargada de la montura, su acoplamiento y su ajuste; otra electrónica encargada del control de los motores; y una tercera, de computación, encargada del diseño y desarrollo del sistema de cómputo para realizar los cálculos, almacenar los datos y comunicarse con el sistema de control.

En lo particular, la presente tesis trata de resolver el problema de la construcción del programa de cómputo para el movimiento del telescopio, siguiendo las directrices con las que se concibió el proyecto desde su inicio: 1) que el programa instalado sea modular, de tal manera que puedan agregarse nuevas funciones y dispositivos; y 2) la aplicación de diferentes ramas de la ingeniería en el desarrollo de la astronomía.

2.1 Definición de la problemática

Como ya dijimos, el problema a resolver en la presente tesis es la construcción de un programa de cómputo que controle la posición del telescopio a través de motores y sensores conectados a la computadora, que maneje una base de datos de los principales objetos celestes, y que calcule la posición de un objeto en la bóveda celeste en una fecha y localidad determinadas.

Ya que estamos definiendo el problema, vale la pena definir también a qué le llamamos *software* o programa de cómputo. El software se define como todas las instrucciones y datos que son ingresados en una computadora y que causa su funcionamiento de algún modo. Esto incluye sistemas operativos, compiladores, rutinas de prueba, programas de aplicación, etc. Los documentos utilizados para definir y describir los programas están también incluidos en la definición[1].

Ahora bien, ¿qué tipo de software es el que se pretende desarrollar?.

La IEEE define dos grandes tipos de software: 1) el aplicativo o aplicaciones y 2) el de sistema. El primero se define como *el software diseñado para cumplir enteramente con las necesidades específicas de un usuario; Por ejemplo, programas de cómputo para la navegación; procesos de control, etc.*[2] La segunda se define como *El software diseñado para facilitar la operación y mantenimiento de un sistema de cómputo y sus programas asociados; Por ejemplo, sistemas operativos, ensambladores y utilerías*[3].

Por tanto, el software o programa a construir es uno del tipo aplicativo o aplicación.

Pasando ahora a los puntos particulares, este programa debe ser diseñado de tal forma que cumpla con los requerimientos iniciales concebidos por los miembros de la sociedad, según lo expresado en el documento definitorio que dice:

"... El proyecto se llevará a cabo siguiendo dos directrices: la primera es que el sistema instalado será modular, de tal forma que podrán agregarse nuevas funciones y dispositivos más complejos posteriores a la puesta en funcionamiento inicial; la segunda es que la realización del proyecto debe mostrar la aplicación de las diferentes ramas de la ingeniería en el desarrollo de la astronomía. Por ésta razón, el trabajo global se divide en proyectos individuales definidos en función del área de la ingeniería a la que corresponden..."

La primera "directriz" marca la pauta para la propuesta central de desarrollo. Se refiere de manera textual a un "sistema modular" al que pueda agregarse "nuevas funciones y dispositivos más complejos". Aquí encontramos dos conceptos que vale la pena definir: "Modular" y, al hablar de nuevas funciones y dispositivos, "modificable".

[1] SIGFRIED, Stefan. *Understanding object-oriented software engineering* …
[2] Ídem
[3] Ídem

El concepto de modularidad puede ser descrito en términos de los cinco criterios y los cinco principios propuestos por Mayer[4].

Los cinco criterios para hablar de modularidad según Mayer son los siguientes:

1. Decomponibilidad
2. Componibilidad
3. Entendibilidad
4. Continuidad
5. Protección

La *decomponibilidad* se refiere al requerimiento de la ingeniería de software y de la administración de proyectos, de que los sistemas se descompongan en pedazos administrables, los cuales puedan ser modificados mucho más fácilmente.

La *componibilidad* se refiere a la propiedad de los módulos de ser libremente combinados, aún en los sistemas para los cuales no han sido desarrollados. Esto es fundamental en la reutilización de software.

La *entendibilidad* ayuda a las personas a comprender un sistema revisando sus partes en vez de revisar el todo.

La *continuidad* en un sistema implica dos cosas: un pequeño cambio realizado al sistema resultará en sólo un pequeño cambio en su comportamiento y, pequeños cambios en la especificación requieren sólo pequeños cambios en algunos cuantos módulos.

Por último, el criterio de la *protección* insiste en que las condiciones de error o excepción deben permanecer confinadas al módulo en el que ocurren o propagarse sólo a unos cuantos módulos relacionados.

En cuanto a los principios requeridos para asegurar los criterios de modularidad, se presenta la lista a continuación.

1. Unidades lingüísticas modulares
2. Pocas Interfaces
3. Interfaces Pequeñas
4. Interfaces Explícitas
5. Ocultamiento de la información

[4] GRAHAM, Ian. *Object Oriented Methods*

El concepto de *unidades lingüísticas modulares* se refiere a la necesidad de correspondencia entre los módulos de un sistema y los tipos primitivos de datos o unidades sintácticas proveídas por un lenguaje o método utilizado para resolver un problema.

El concepto *pocas interfaces* se refiere a que cada módulo deberá comunicarse con otros pocos. Tan pocos como sean posibles.

El principio de *interfaces pequeñas* se refiere a que las interfaces deben pasar una cantidad tan pequeña de datos como sea posible.

La necesidad de *interfaces explícitas* viene de la necesidad de utilizar objetos vía su especificación y no de su implementación. Es decir, la utilización de cierto componente de software debería ser de lo más sencilla entendiendo su definición sin necesidad de entrar a revisar el código que lo implementa.

Por último, el principio de *ocultamiento de la información* dice que los módulos deben ocultar las decisiones de diseño utilizadas en su construcción.

En resumen, al hablar de modularidad hablamos de criterios a verificar y principios a seguir propuestos por Mayer, y que darán la pauta para las decisiones de análisis, diseño y desarrollo del software.

El problema con las modificaciones puede ser expresado como un problema de mantenimiento.

El *mantenimiento de software* es el proceso de modificar un software, sistema o componente después de liberación para corregir fallas, mejorar el rendimiento, o adaptar a un ambiente distinto[5].

Sin duda, una de los problemas centrales de la ingeniería de sistemas es precisamente éste, lograr la creación de software de fácil mantenimiento.

Philippe Krutchen señala que a pesar de que los proyectos de desarrollo de software fracasan por múltiples motivos, se puede identificar un número común de síntomas que caracterizan a la mayoría de proyectos fallidos[6]. Y entre los puntos citados menciona al “software que es muy difícil de mantener o extender”.

Obviamente, un software de difícil mantenimiento requiere de mucho mayor esfuerzo de los profesionales para agregar nuevas funcionalidades, corregirlas o extenderlas. En los procesos comerciales de desarrollo es ésta la principal causa de pérdidas económicas. En nuestro caso es parte de los atributos del sistema, justificado por el hecho de que al tratarse de un proyecto estudiantil se espera que los miembros de la sociedad amplíen los alcances del proyecto en curso o simplemente ideen nuevos. Por ejemplo, algunos proyectos a futuro basados en el sistema de control de telescopio es el acceso vía Internet para que miembros

[5] SIGFRIED, Stefan. *Understanding object-oriented software engineering*. IEEE Press & IEEE Computer Society Press. 1996. NY, EUA. Pag. 12.
[6] KRUCHTEN, Philippe. *The Rational Unified Process an Introduction*. Addison-Wesley. 2nd edition. 2001. EUA. Pág. 3-4

de la sociedad que no tengan acceso físico al observatorio puedan realizar peticiones de observación como se realiza en algunos observatorios de universidades en otros países. Extender el proyecto original a éste nuevo implicaría extender la funcionalidad del software hacia el Internet e instalar nuevos dispositivos como una cámara CCD que sería la encargada de capturar la imagen del objeto para ser enviada más tarde al usuario.

Los dos problemas planteados: Modularidad y fácil mantenimiento, son problemas de diseño de software y mantenimiento del mismo y competen a la rama de la ingeniería llamada *Ingeniería de programación* también llamada *Ingeniería de sistemas* o *de software*, la cual es definida como:

La aplicación sistemática, disciplinada y cuantificable de enfoques de desarrollo, operación y mantenimiento de software.

En conclusión, el problema de construcción de un programa para el control de telescopio dentro del ámbito de la ingeniería de programación es un problema de diseño de software y mantenimiento del mismo, que será resuelto mediante las metodologías, técnicas y tecnologías propuestas en el siguiente capítulo para asegurar que se cumplan los criterios de Mayer que permitan la modularidad y que soporten un fácil mantenimiento.

2.2 Justificación

Existen muchas razones por las que se requiere controlar un telescopio utilizando una microcomputadora, pero tal vez la razón principal es la del costo/beneficio en términos de incrementar la eficiencia del trabajo de los científicos en la adquisición de datos, en cuyo caso el costo de los mecanismos de control queda perfectamente justificado. Sin embargo, antes de desarrollar esta razón primaria, repasemos las otras múltiples razones por la que un control de telescopios con la ayuda de una computadora podría ser justificado.

2.2.1 Difícil acceso a lugares de observación

En algunas situaciones el control de telescopios puede ser de gran ayuda o incluso esencial. Un buen ejemplo podría ser el telescopio automático del Polo Sur donde nadie esta ahí para operarlo.[7]

Dicho telescopio inició operaciones en enero de 1995. Debido al frío excepcional y la atmósfera seca, el Polo Sur es el mejor lugar para un telescopio terrestre que realice observaciones astronómicas en longitudes de onda milimétricas. [8] El sistema entero del observatorio está altamente automatizado para reducir al mínimo la necesidad de la intervención humana.[9]

[7] TRUEBLOOD, Mark & GENET, Russell. 1985. *Microcomputer control of telescopes.* Willmann-Bell, Inc. EUA, pag. 5
[8] 2003, Antarctic Submillimeter Telescope and Remote Observatory: http://cfa-www.harvard.edu/~adair/AST_RO/
[9] 2003, Antarctic Submillimeter Telescope and Remote Observatory: http://cfa-www.harvard.edu/~adair/AST_RO/info.html

Gracias al control por computadora las adversidades climatológicas que existen no son obstáculo para la operación del telescopio y las ventajas de observación pueden ser completamente aprovechadas.

Un ejemplo más sería el telescopio *Hubble* el cual fue diseñado en los años setenta y lanzado al espacio en 1990. Hasta el día de hoy se encuentra en operación, orbitando a más de 600 kilómetros de la tierra. Utiliza un sistema de posicionamiento y de control único que lo hace extraordinariamente preciso y estable[10].

2.2.2 Precisión para apuntar el telescopio

En algunas otras ocasiones el control de telescopios a través de la computadora se vuelve más que necesario para apuntar el telescopio en condiciones no favorables con gran precisión.

Un ejemplo serían los telescopios infrarrojos (IR), los cuales son operados durante el día y se requiere de gran precisión para apuntarlos. O los telescopios de espejos múltiples, para los cuales el control humano para mantener correctamente alineados los espejos es simplemente insuficiente[11].

Las microcomputadoras pueden ser utilizadas para controlar o asistir en una amplia gama de funciones de observación.

Atrás de una gran eficiencia en la captura de datos se encuentra siempre un control de telescopio por microcomputadora y una instrumentación que incrementa la rentabilidad de dicha adquisición de datos. Una captura manual de dichos datos después de una sesión de observación que se ha extendido hasta altas horas de la noche es propicia a errores. El astrónomo que es ayudado por sistemas de posicionamiento de telescopios tiene más tiempo libre para pensar en astronomía y olvidarse del manejo del telescopio lo cual constituye, además, un trabajo monótono y no crítico para sus investigaciones.[12]

2.2.3 Razones de costo/beneficio

En muchos problemas de ingeniería, un nuevo proyecto no sólo se justifica en términos de tecnología que ayude a realizar mejor una tarea; en la mayoría de los casos, nos enfrentaremos al problema de justificar en términos económicos la inversión, lo cual intentaremos realizar en los siguientes párrafos.

Muchos sistemas computarizados han sido desarrollados como respuesta a la expectativa de que los costos de desarrollo y mantenimiento del sistema van a ser compensados por el incremento en la productividad o una mayor eficiencia en la utilización de recursos.

[10] 2003, NASA, The Hubble project: http://hubble.nasa.gov/overview/
[11] TRUEBLOOD, Mark & GENET, Russell. 1985. *Microcomputer control of telescopes.* Willmann-Bell, Inc. EUA, Pág. 5
[12] Ídem

En el caso de un sistema de control de telescopios existe el problema de obtener más tiempo de utilización de telescopio que el que está disponible. Algunos científicos[13] argumentan que el tiempo de *no-observación* (tiempo gastado en posicionar el telescopio, poner a punto el o los instrumentos, etc.) que puede ser reducido mediante la computarización, es una fracción muy pequeña del tiempo total de uso del telescopio, por lo cual, la reducción en éste no aumenta el rendimiento de procesamiento de datos científicos lo suficiente como para justificar el costo.

Sin embrago, este punto de vista puede ser muy discutible y depende mucho del tipo de observación a realizarse y del observatorio en donde se quiera implementar la solución de computarización de telescopio. Por ejemplo, en el caso de un observatorio de fotometría se tiene que los tiempos de no-observación, son muy largos debido a la dedicación que exige calibrar y poner a punto los instrumentos y apuntar el telescopio; en ambas tareas este tiempo podría ser dramáticamente reducido mejorando la precisión, utilizando una computadora.

Aunque la computarización podría pensarse más justificable en los grandes telescopios dónde los ahorros en los costos también son mayores, el abaratamiento del *hardware* en nuestros días hace posible que la inversión sea justificada en los pequeños observatorios y de esta manera gocen de los beneficios de la computarización.

Para concluir con este apartado de justificaciones, diremos que existen muchas otras razones por las cuales computarizar la operación de un telescopio. Sin embrago, como escriben Truebold y Genet, entre las muchas razones que existan *en muchas ocasiones (tal vez la mayoría), la razón real será que se puede hacerlo* (los astrónomos) *y verlo como una cosa divertida para hacer.*

[13] Ídem, Pág. 8

3 Metodologías, Tecnologías y Técnicas a aplicar

Como vimos en el capítulo anterior, nuestro problema es el de diseñar un programa aplicativo con fines astronómicos que cumpla con los principios de modularidad de Mayer y, además, sea de fácil mantenimiento.

Estas dos características parecen resumir los últimos años de desarrollo e investigación en ingeniería de programación, que revisaremos brevemente en este capítulo.

En este capítulo describiremos el marco de referencia que servirá para plantear una solución al problema de diseño de la aplicación para asegurar la modularidad y el fácil mantenimiento.

Con respecto a la modularidad, vimos en el capitulo anterior que Mayer lista cinco principios para asegurarla. Dichos principios pueden ser fácilmente mapeados con las características intrínsecas del paradigma Orientado a Objetos (OO) y los lenguajes de programación que lo implementan.

Con respecto al mantenimiento, observaremos que las metodologías de análisis y diseño orientado a objetos, apoyados por una herramienta de modelado robusta; así como los procesos de desarrollo iterativos, pueden ayudar a disminuir los tiempos de desarrollo aumentando la probabilidad de éxito y por ende, disminuyendo el esfuerzo durante el mantenimiento.

Por último, trataremos de asegurar la implementación de los paradigmas OO utilizando un lenguaje OO cuyas características ayuden a implementar la modularidad y el fácil mantenimiento.

3.1 Metodologías Orientadas a Objetos

Como vimos en el capítulo anterior, el problema que compete a la presente tesis es el de lograr un programa que sea modular y de fácil mantenimiento. Dicho problema se encuentra dentro del marco de la ingeniería de software o ingeniería de sistemas

Estos atributos son para los autores el problema raíz de la ingeniería de software. Las inquietudes por desarrollar sistemas de cómputo de manera cada vez más rápida y eficiente a costos menores se basan en la reutilización de código logrados por la modularidad de sus componentes, que al mismo tiempo permiten un fácil mantenimiento.

El rápido crecimiento de los sistemas computacionales ha traído consigo grandes beneficios: velocidades de procesamiento que hace apenas unos cuantos años no podríamos haber imaginado; capacidades de almacenamiento inconcebibles para apenas una generación anterior; dispositivos móviles cada vez más poderosos y pequeños. En síntesis, mayor capacidad de manejo de información en menor tiempo.

Sin embargo, hardware y software (partes inherentes a cualquier sistema de cómputo) no han crecido a la misma tasa de desempeño/precio, dando origen a lo que se conoce como "la crisis del software". Históricamente, hemos observado cómo el valor que nos ofrece el hardware con relación a su precio ha ido aumentando dramáticamente, mientras que por cada unidad monetaria gastada en desarrollo de software no obtenemos, por mucho, un beneficio apenas equiparable. Además, mucho más preocupante que la cantidad de software obtenida, es la calidad de éste[14]. Esta crisis aumenta si pensamos en el hecho comprobado, de que su demanda ha crecido rápidamente[15].

La economía mundial depende del incremento de software. Los tipos de software que la tecnología hace posible y la sociedad demanda han aumentado en tamaño, complejidad, distribución e importancia, llevando al límite el conocimiento que tenemos acerca del desarrollo de software. Tratar de realizar sistemas que utilicen la más avanzada tecnología trae consigo un conjunto de problemas técnicos y organizacionales. Complementando el problema, está el hecho de que el negocio continúa demandando el incremento de productividad y la mejora de la calidad con un rápido desarrollo e implementación. Sumado a esto, la provisión del personal calificado para desarrollo no va a la par con la demanda[16].

El resultado es que la construcción y mantenimiento de software se vuelve cada vez más complejo; y la construcción de software de calidad de manera predecible y repetible es aún más complejo[17].

[14] SIGFRIED, Stefan. *Understanding object-oriented software engineering*. IEEE Press & IEEE Computer Society Press. 1996. NY, EUA. Pág. 3-4.
[15] Ídem
[16] KRUCHTEN, Philippe. *The Rational Unified Process and Introduction (Second Edition).* Addison-Wesley. 5ª impresión 2001. EUA. Pág. 4.
[17] Ídem

El paso de más de 30 años de crisis del software llevó a las mentes más prominentes de la ingeniería de programación a una revolución, a un nuevo paradigma conocido como la "orientación a objetos", término que más allá de estar de moda y representar la modernidad tecnológica, representa toda una filosofía de desarrollo de sistemas.

La orientación a objetos, o mejor dicho, los métodos de análisis y desarrollo orientados a objetos, están enfocados a disminuir la complejidad y el mantenimiento del proceso de desarrollo mediante el modelado de la realidad de una manera sencilla pero robusta, una abstracción de la realidad en los términos más familiares a la percepción humana: conceptos, o lo que es lo mismo, objetos.

Muchas metodologías de análisis orientado a objetos (OOA por sus siglas en inglés) y de diseño orientado a objetos han nacido desde la década de los 70's que se reconoce como la década donde comenzaron a desarrollarse tecnologías orientadas a objetos. A continuación revisaremos de manera breve la historia de ésta metodología.

3.1.1 Breve historia del arte

La mayoría de los autores coinciden en que el nacimiento de la programación orientada a objetos inició con el lenguaje de simulación de eventos discretos desarrollado por Kristen Nygaard y Ole-Johan Dahl llamado *Simula,* en Noruega, en el año de 1967 y continuó con el desarrollo de un lenguaje que casi se convierte en el fetiche de la notación del *objeto*: *Smalltalk* en los 70's.

El modelado de una simulación es un problema particularmente difícil para los programadores convencionales. El problema requiere que el programador adapte el flujo funcional de control normal a la mayoría de lenguajes a un flujo funcional que es más naturalmente descrito en términos de complejos objetos los cuales cambian de estado y de eventos a cada momento. En la programación orientada a objetos, es un enfoque totalmente natural desde el hecho de que cada estructura de los programas refleja directamente la estructura del problema. Además, es mucho más claro qué es un objeto en este tipo de problemas: carros en la calle, máquinas en la línea de producción, etc. Son usualmente cosas del "mundo real" más que abstracciones y mucho más fáciles de identificar.

El término *programación orientada a objetos* finalmente llegó junto con el advenimiento del lenguaje Smalltalk. Smalltalk fue largamente desarrollado en el centro de investigaciones Xerox en Palo Alto (PARC), pero tuvo sus orígenes no sólo en Simula sino también en el trabajo doctoral de Alan Kay quien, al igual que Rentsch, se basó en la visión de una pequeña pero potente computadora personal capaz de manejar cualquier tipo de información administrando el problema y capaz de ser utilizada por cualquier tipo de persona. La primera versión de dicha máquina fue la llamada *Flex* la cual en el PARC fue llamada *Dynabook*. Smalltalk fue el software esencial de la Dynabook y fue fuertemente influenciado por la notación de clases y la herencia de Simula y las características estructurales de LISP[18].

[18] LISt Processing. Fue un lenguaje desarrollado por John McCarthy alrededor de 1958, que fue seleccionado como el lenguaje básico para los trabajos iniciales en IA.

La siguiente fase de la orientación a objetos, en la difícil época de los 80's, se caracterizó por un "boom" en el interés por las interfaces de usuario (UI por sus siglas en inglés). Los pioneros más conocidos como Xerox y después *Apple* dieron al mundo el omnipresente WIMP (Siglas de *Windows, Icons, Mice and Pointers* que se refieren al estilo de Interfase Gráfica de Usuario) y muchas de las ideas de SmallTalk, incomprendidas por muchos, fueron atadas fuertemente a estos desarrollos. De hecho, se considera que una de las razones que contribuyeron al éxito de la programación orientada a objetos fue la complejidad de estas interfaces y su alto costo asociado. Sin la inherente reusabilidad del código orientado a objetos se ha pensado que éstas interfaces no hubieran sido realizadas a gran escala. Por ejemplo, se ha estimado que el *Apple Lisa*, precursor de *Machintosh*, representa cerca de 200 años/hombre de esfuerzo, haciendo la mayor aportación el desarrollo del GUI.

Desde la mitad de la década de los 70's y en los años posteriores, existió una contribución cruzada importante entre la programación orientada a objetos y la investigación en el campo de la Inteligencia Artificial (IA), ampliando muchos lenguajes con ideas de objetos, como fue el caso de LISP, cuyas extensiones: *KEE* y *ART* se vieron muy influenciadas por la tecnología de objetos.

A su vez, la tecnología orientada a objetos se vio beneficiada al absorber las sofisticadas teorías relativas a la herencia, desarrolladas por la IA.

Otra corriente de investigación de la IA, conjunto con la investigación en la computación de la época, condujo al concepto de *actores*. Los sistemas Actores, como cajas negras, procura modelar conjuntos de trabajadores o expertos. Un actor es una noción más antropomórfica que la de objeto y ha definido responsabilidades, necesidades y conocimiento acerca de colaboradores.

Al aplicarse los primeros lenguajes orientados a objetos a otro tipo de desarrollos, distintos a las GUI's, surgieron problemas que llevaron al desarrollo de nuevos lenguajes tales como *Eiffel* y extensiones de los lenguajes convencionales existentes que habían demostrado su efectividad, tales como *C* y *Pascal*.

Además, el desarrollo de interfaces gráficas no plantea ningún problema significante de manejo de datos, comparado con las aplicaciones comerciales. Esto significaba que los lenguajes de programación orientada a objetos tampoco ofrecían las facilidades suficientes para manejar los objetos persistentes, concurrencia, etc. Esto condujo a la investigación y desarrollo de bases de datos orientadas a objetos.

Al tiempo que fue madurando la programación orientada a objetos, el interés se fue concentrando en los métodos de diseño orientados a objetos (DOO u *OOD* por sus siglas en inglés) y en los métodos de análisis orientados a objetos (AOO u *OOA*). Los beneficios de la reusabilidad y la extensibilidad podrían ser aplicados a diseños y especificaciones así como al código.

Comenzaron entonces importantes planteamientos al respecto, como si un diseño orientado a objetos debe ser implementado en cierto lenguaje o si los métodos actuales de diseño están ligados a lenguajes específicos.

Para la década de los 90's, el interés se concentró ahora en la estandarización. La tecnología de objetos podía ser exitosa sobre la inercia de las prácticas existentes sólo si los usuarios podían moverse del punto actual a los requeridos sistemas abiertos. Si las aplicaciones orientadas a objetos son incompatibles, si las bases de datos no pueden trabajar interconectadas unas con otras y con bases de datos relacionales, y si no existen notaciones y términos estándar para el análisis y diseño orientados a objetos; había poca esperanza para lograrlo. El principal protagonista en el área de estandarización, ha sido la organización llamada *Object Management Group* (*OMG)*. La OMG es un grupo grande y en crecimiento de compañías influyentes en tecnología, comprometidas en establecer acuerdos entre vendedores de terminología de orientación a objetos y el estilo e interfase estándar de lenguajes de cuarta generación (*4GL*, por sus siglas en inglés).

La OMG tiene la tarea de publicar de manera rápida los estándares acordados. Entre sus publicaciones más sobresalientes se encuentra una guía de arquitectura de software, un estándar para la arquitectura común para el envío de peticiones de objetos, mejor conocido como CORBA (*Common Object Request Broker Architecture)* una arquitectura acotada para la interoperación de aplicaciones orientadas a objetos.

A mitad de los 90's, las aplicaciones se comenzaron a enfocar más al Internet. Y fue entonces que surgió uno de los lenguajes de programación orientada a objetos más importantes y que seguramente influyó en gran medida al desarrollo de la tecnología y de nuevos estándares: El lenguaje de programación *Java* desarrollado por *Sun Microsystems*.

En ésta misma época nace otro lenguaje que revolucionaria la tecnología de orientación a objetos: *El Lenguaje Unificado de Modelado* (*UML*, por sus siglas en inglés) que comenzó a finales de 1994 cuando *Grady Booch* y *Jim Rumbaugh* de *Rational Software Corporation* empezaron a unificar sus métodos. En 1997 UML 1.1 fue aprobada por la OMG convirtiéndose en la notación estándar *de facto* para el análisis y el diseño orientado a objetos.

Para el año 2000 Microsoft anuncia el lanzamiento del lenguaje de programación denominado C#, basado en C y que adiciona nuevas características para competir con Java, además de una plataforma de desarrollo que ambiciona con aglutinar y unificar en una solo ambiente a todos los lenguajes orientados a objetos, permitiendo que en una aplicación se encuentren módulos escritos en distintos lenguajes: la plataforma *.NET*. Son más de 20 lenguajes los que están en proceso de migrarse a la plataforma .NET, entre ellos *Python.*

La fase actual de la historia de la orientación a objetos se caracteriza por cambiar el énfasis de la programación al diseño y el análisis y por un conocimiento de la aplicación de sistemas abiertos y de estándares.

3.1.2 Terminología e ideas básicas de la tecnología Orientada a Objetos

A lo largo de los siguientes capítulos, conforme vayamos desarrollando la idea principal de la presente tesis, se harán referencia a múltiples ideas y conceptos propios de la tecnología OO que vale la pena en éste momento definir.

- **Modelo.** Un modelo es una representación de un proceso del mundo real, dispositivo, o concepto. Las representaciones (gráficas o narrativas) que representan a nuestro sistema es lo que llamamos modelo. Un modelo de alto nivel muestra la mayoría de las partes del sistema, cómo interactúa y cómo se relaciona con su ambiente.

- **Objeto**. Las unidades básicas de construcción, usadas para conceptualización, diseño o programación, son instancias organizadas en clases con características en común. Las características comprenden atributos y procedimientos, llamados operaciones o métodos.

  Un objeto debe tener relevancia al dominio del problema, debe existir independientemente, es decir, debe poseer identidad, comportamiento y características. Un objeto representa conceptos de la realidad utilizando la abstracción.

- **Atributo.** Un atributo es un descriptor de una instancia. Nos dice algo importante y significante acerca de la naturaleza de una instancia.

- **Encapsulación**. Las estructuras de los datos y los detalles de implementación de un objeto deben de esconderse de otros objetos del sistema. La única forma para acceder al estado del objeto es mandando un mensaje que cause que uno de los métodos se ejecute.

- **Mensajes**. Los objetos, clases y sus instancias se comunican a través de mensajes. Los mensajes se implementan como llamadas a métodos o funciones.

- **Herencia**. Las instancias heredan usualmente todas y sólo las características de las clases de las cuales provienen, pero es posible en un sistema orientado a objetos que las clases hereden características de clases más generales llamadas superclases. En este caso la herencia puede ser sobrescrita y nuevas características pueden ser agregadas a las instancias.

- **Poliformismo**. Es la habilidad de usar con la misma expresión diferentes operaciones. El poliformismo es usualmente implementado como un ligado dinámico.

Las principales características de la orientación a objetos son la herencia y la encapsulación, también conocida como abstracción[19].

Abstracción es el concepto más difícil de la orientación a objetos y podríamos definirlo de la siguiente manera: “ Representación de las características esenciales de algo sin incluir el trasfondo o los detalles innecesarios“. En programación esto significa que los objetos deben abstraer y encapsular todos los datos y procesos esenciales, es decir, no sólo sus características sino su comportamiento. Y para representar su comportamiento o los procedimientos permitidos sobre sus atributos, existen los *métodos*.

[19] GRAHAM, Ian. *Object Oriented Methods*. 2ª edición. 1994, Addison Wesley, 9Pág.

Un método es un procedimiento o función que altera el estado de un objeto o causa que el objeto mande un mensaje, es decir, regrese valores. La descripción de un método es llamada *operación*. Las operaciones definen cuáles de los mensajes recibidos deben ser procesados por los objetos.

Una *clase* es una colección de objetos que comparten características y métodos comunes. También puede considerarse como una plantilla para crear instancias de objetos, y puede ser llamada Tipo de objeto, pensando estrictamente que la clase es la implementación del mismo. Los atributos y métodos de un Tipo de objeto son también conocidos como sus características y responsabilidades. Un atributo representa la responsabilidad de conocer algo y un método la responsabilidad de hacer algo. Un tipo de dato abstracto (*ADT* por sus siglas en ingles) es una abstracción similar a una clase, que describe un conjunto de objetos en términos de su estructura de encapsulación y operaciones. ADT es lo contrario a los tipos de datos primitivos de algunos lenguajes, posiblemente definidos por el usuario o diseñador del sistema. La diferencia entre los ADT y las clases son sus métodos, debido a que las clases describen especificaciones que serán compartidas más tarde por colecciones de objetos u otras clases y no describen sólo objetos de un tipo especifico como los ADT. Aunque desde el punto de vista del analista son lo mismo.

Con lo anterior debemos concluir que un objeto debe ser entendido como una clase o un ADT o una instancia de un tipo. Sin embargo en las *Clases Abstractas*, no tenemos instancias, sus hijos son otras clases, entonces el término de objeto no lo usaremos, pues la deferencia entre clase e instancia es importante[20]. Un objeto tiene dos aspectos principales, el interno que describe el estado, implementación e inicialización del objeto; y el externo que sólo muestra los nombres de sus métodos y los tipos de sus parámetros, es decir, nos muestra qué puede hacer el objeto

Los atributos de un objeto, se van a identificar con dos métodos estándar conocidos como *get* y *put*. Los atributos de las instancias serán a veces conocidos como *variables de instancia*, y los correspondientes a las clases como *variables de clase*. Así como distinguiremos también los *métodos de clase* y los *métodos de instancia*.

Encapsulación es equivalente a ocultamiento de información, refiriendo a la práctica de incluir dentro del objeto todo lo que necesita, y de manera que ningún otro objeto pueda tener acceso a su estructura interna. Lo mismo aplica para los métodos, su implementación es privada, sólo su comportamiento es visible a los demás objetos.

Los datos de los objetos en un sistema orientado a objetos son sólo obtenidos a través de los mensajes. Un mensaje consiste en una dirección (a qué objeto u objetos se va a enviar) y una instrucción, expresada como el nombre del método con cero o más parámetros. Si la dirección (objeto) tiene el método enviado, devolverá entonces un valor al objeto que envió el mensaje. Al conjunto de mensajes al que un objeto es sensible a responder, se le conoce como *protocolo*.

Otros términos utilizados son poliformismo y sobrecarga de operadores, ambos representan conceptos muy similares: la habilidad de usar el mismo símbolo para diferentes propósitos.

[20] GRAHAM, Ian. *Object Oriented Methods*. 2ª edición. 1994, Addison Wesley, Pág 12

La definición formal de poliformismo (muchas formas) significa la habilidad de una variable o función de tomar diferentes formas en tiempo de ejecución, o la habilidad de referirse a instancias de varias clases. Sobrecarga es el caso especial donde dos o más diferentes operaciones comparten el mismo nombre.

Generalización es la habilidad de definir módulos parametrizables, como ejemplo tenemos a las *Listas*. Listas de enteros, de nombre, de objetos. Usualmente los parámetros son tipos de datos. La generalización y la herencia son técnicas alternativas para la reutilización y extensibilidad de código.

La generalización identifica y define los atributos y operaciones comunes en una colección de objetos. Este proceso ayuda a identificar las clases, reduce la redundancia y promueve la reutilización.

Un concepto muy importante en los sistemas orientados a objetos, es la manera de manejar las relaciones entre instancias y clases y eliminar la redundancia en el almacenamiento de datos o procedimientos. La llave para resolver lo anterior es la *herencia* o la clasificación jerárquica. La herencia es una forma de establecer generalización, especialización y clasificación entre los objetos. Vamos a considerar dos tipos principales de herencia la que se integra por la generalización y clasificación y la que se integra por composición y agregación. En los sistemas orientados a objetos, se heredan los atributos y los métodos de la clase superior, pero no los valores de los atributos, además de agregar sus propios atributos o métodos o sobrecargar los heredados.

Podemos ver a la herencia como una estructura de árbol, moviéndonos a través del árbol hacia abajo las clases se vuelven mas especializadas, y por el contrario moviéndonos hacia arriba del árbol las clases son más generales.

Existe un caso especial donde un objeto o clase puede tener dos o más padres, a esta habilidad se le conoce como *herencia múltiple*. La dificultad con la herencia múltiple, es que puede darse el caso de que las propiedades heredadas de los padres, sean directa o parcialmente contradictorias o muy similares. Los principales puntos de conflicto sobre las propiedades heredadas (atributos o métodos) son el nombre o el valor. La solución más común para los métodos heredados es utilizar la sobrecarga.

Existen otros conceptos importantes en la Orientación a objetos, como el termino *Actor* del sistema y *delegación* de la responsabilidad. La delegación de responsabilidades en una forma de herencia sin clases permite que los objetos deleguen a otros objetos permisos para ejecutar operaciones sobre sí mismos. Esto permite a los objetos transformar el comportamiento de otros sin estar atados por la herencia de clases.

Otro concepto interesante es la recursividad a sí mismo o auto referencia, esto quiere decir que un objeto puede mandarse mensajes a sus propios métodos de manera recursiva, o mandarse mensajes así mismo.

Para terminar describiremos la diferencia entre un sistema basado en clases, en objetos y un sistema orientado a objetos. Un sistema *basado en objetos* soporta la encapsulación y la definición de los objetos como entidades únicas, no utiliza la abstracción y por lo tanto ni las

clases y la herencia. Un lenguaje basado en clases incluye la noción de abstracción y de instancias de clases, soporta las características de un sistema basado en objetos y la herencia. Sin embargo no permite el concepto de clase abstracta debido a que no puede tener instancias. Por todo lo anterior, un sistema orientado a objetos es aquel que une al basado en objetos y al basado en clases, permitiéndose agregar la completa herencia entre las instancias y las clases, y las clases y las clases[21]. Es decir, tanto instancias como clases heredan atributos y métodos de sus clases padres, además de permitir la recursividad.

3.1.3 El Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (*UML, Unified Modeling Language*) fue creado por Grady Booch, Jim Rumbaugh e Ivar Jacobson y es el sucesor de la oleada de métodos de análisis y diseño orientado a objetos que surgió a finales de la década de 1980 y principios de la siguiente. Es un lenguaje de modelado y no un método. Es una notación (principalmente gráfica) de que se valen los métodos para expresar los diseños.

Breve historia del arte

En la década de 1980 *Smalltalk* se estabilizó y nació C++. Los principales desarrollos de *software* estaban guiados por los lenguajes de programación, y debido a la popularidad de los métodos de diseño aplicados a la industria en las décadas de 1970 y 1980, se pensó aplicarlos en el desarrollo orientado a objetos. Los principales estudios de análisis orientado a objetos y los métodos de diseño se desarrollaron entre 1988 y 1992. Durante este tiempo se publicaron una serie de libros que influyeron y contribuyeron a la creación de UML. Sally Shlaer y Steve Mellor escribieron un par de libros (1989 y 1990) sobre análisis y diseño; Peter Coad y Ed Yourdon escribieron libros en los que desarrollaron enfoques hacia los métodos orientados a objetos (1991); la comunidad *Smalltalk* aporto el diseño guiado por responsabilidad (Responsibility-Driven Desing, 1990) y las tarjetas clase – responsabilidad – colaboración (CRC, 1989); Grady Booch trabajaba con *Rational Software*, creando su metodología Booch'93; Jim Rumbaugh creó, al trabajar en los laboratorios de *General Electric*, un método llamado técnica de modelado de objetos (OMT, 1991) e Ivar Jacobson escribió con base en su experiencia en *Ericsson* el concepto de casos de uso.

Los métodos que se habían desarrollado por todos estos autores eran muy similares, sin embargo tenían una gran cantidad de diferencias menores y los mismos conceptos tenían denominaciones diferentes. El número de metodologías orientadas a objetos creció de menos de 10 a más de 50 entre 1989 y 1995, por lo que solo quedaba un camino, la estandarización. Así, en 1994 Jim Rumbaugh empieza a trabajar en *Rational* y junto con Grady Booch trabajan en la unificación de sus métodos (OMT y Booch). Para 1995 Grady y Jim habían preparado la primera descripción pública de su método integrado: la versión 0.8 del Método Unificado (*Unified Metod*).

Mientras tanto Ivar Jacobson, que entonces trabajaba en *Objetory*, trataba de estandarizar su método OOSE (*Object Oriented Software Engineering*), pero fue hasta 1995 que *Rational* compro *Objetory* y Jacobson se unió a Booch y Rumbaugh en el proceso de unificación. Así durante 1996, los tres Amigos, como se les conoce desde entonces, construyeron su método

[21] GRAHAM, Ian. *Object Oriented Methods*. 2ª edición. 1994. Addison Wesley. pág. 26.

y lo nombraron Unified Modeling Language (UML), lenguaje unificado de modelado. Éste fue enviado a la OMG (*Object Management Group*) para llevar a cabo la estandarización, los encargados de este proyecto fueron Mary Loomis, y después Jim Odell. En enero de 1997, *Rational* liberó la versión 1.0 de la documentación de UML, así mismo se conformó un conjunto de empresas (*IBM, HP, Rational, Oracle, Microsoft, Unisys*, entre otras) para apoyar la unificación, y así en Noviembre de 1997 con todo el apoyo de la industria y la OMG, la versión 1.1 de UML fue adoptada dentro de la OMA (*Object Management Architecture*).

Se liberaron algunas versiones de UML, pasando por la 1.2 y la 1.5 tratando de estandarizar más el lenguaje. Actualmente la ultima versión de UML es la 2.0 fue liberada a mediados del 2001.

Ahora que conocemos un poco más sobre cómo surgió UML, debemos de empezar a entender como aplicarlo en el análisis y diseño de software. Es muy importante señalar que para lograr un adecuado y exitoso desarrollo de software, debemos tener una correcta comunicación con el usuario del sistema, y esto lo lograremos aplicando las herramientas de UML, como lo son los casos de uso y los diagramas de clases. Al aplicar UML debemos enfocarnos a los conceptos para hablar el mismo lenguaje que los usuarios.

UML es un lenguaje para modelar, no un método, por lo que es independiente del proceso. El método orientado a objetos se encuentra en desarrollo.

Casos de uso

Un caso de uso es, en esencia, una interacción típica entre un usuario y un sistema de cómputo[22].

Los casos de uso captan alguna función visible para el usuario, pueden ser pequeños o grandes, logran un objetivo discreto para el usuario y se describen a través de diagramas de casos de uso y ocasionalmente diagramas de actividad.

Elementos de un diagrama de casos de uso:

? *Sistema*

Como parte del modelado de casos de uso, los límites del sistema desarrollado están definidos. Nótese que un sistema no necesariamente tiene que ser un sistema de software; puede ser un negocio o una máquina. Definir las fronteras y la responsabilidad total del sistema no siempre es fácil, por que no siempre es obvio cuáles tareas deben ser automatizadas por el sistema y cuáles deben ser llevadas manualmente o por otro sistema. Otra consideración es qué tan grande debe ser el sistema en su primera versión. Es tentador ser ambicioso, pero esto puede llevar a un sistema grande y un tiempo de entrega muy largo. Una mejor idea es identificar la funcionalidad básica y concentrarse en definir un sistema estable y bien planeado en su arquitectura de modo que en el futuro pueden agregarse nuevos módulos fácilmente.

[22] FOWLER, Martin, KENDALL, Scott. *UML Gota a Gota*. Adison Wesley. 1997. México. Pág. 49.

Un sistema es descrito en diagramas de casos de uso como una caja; el nombre aparece sobre o dentro de la caja.

? *Actores*

Empleamos el término actor para llamar así al usuario, cuando desempeña ese papel con respecto al sistema. Un actor es alguien o algo que interactúa con el sistema; es quién o qué usa el sistema[23].

Cuando se trata con actores, conviene pensar en los papeles, no en las personas ni en los títulos de sus puestos. Dicho de otra manera, un actor es un rol, no un usuario individual del sistema. De hecho, una misma persona puede ser varios actores, dependiendo de su rol dentro del sistema.

Los actores son quienes llevan a cabo los casos de uso dentro del sistema. Un actor puede realizar varios casos de uso; a la inversa, un caso de uso puede ser ejecutado por varios actores.

El actor se comunica con el sistema enviando y recibiendo mensajes, los cuales son similares a la POO. Un caso de uso siempre es iniciado por una actor que envía un mensaje. Esto es llamado estímulo. Cuando un caso de uso es ejecutado puede mandar un mensaje a uno o más actores, los cuales pueden ser diferentes a los que iniciaron el caso de uso.

Los actores pueden categorizarse. Un actor es primario si utiliza las funciones primarias del sistema, tal como la función principal. Los actores secundarios se encargan del manejo de la funcionalidad secundaria, tal como el manejo de las bases de datos, respaldos y otras tareas de administración. Los actores también pueden definirse como activos (los que inician los casos de uso) y los pasivos (aquellos que no inician casos de uso, pero participan de alguna manera)

Algunas preguntas que pueden ayudar en la búsqueda de actores para el sistema, son las siguientes:

- ¿Quién usará la funcionalidad primaria del sistema?
- ¿Quién necesitará soporte del sistema para realizar sus funciones diarias?
- ¿Quién necesitará mantener, administrar y guardar el trabajo del sistema (actores secundarios)?
- ¿Qué dispositivos de hardware necesitará manejar el sistema?
- ¿Con qué otros sistemas interactúa el sistema?

Los actores en UML pueden representarse de dos formas:

[23] FOWLER, Martin, KENDALL, Scott. *UML Gota a Gota*. Adison Wesley. 1997. México. Pág. 52

Como clases del estereotipo <<*actor*>>*,* con el nombre del actor como nombre de la clase; o con el icono característico del *stickman*, con el nombre del actor debajo de la figura.

Un caso de uso en UML es entonces definido como una secuencia de acciones que desarrolla un sistema que deriva en un resultado observable de algún valor para un actor en particular. Las acciones pueden involucrar comunicación con un número de actores (usuarios y otros sistemas), así como desarrollar cálculos y trabajo dentro del sistema. Las características de una caso de uso son:

Un caso de uso siempre es iniciado por un actor, directa o indirectamente. Ocasionalmente, el actor puede no percatarse de iniciar un caso de uso.

Un caso de uso proporciona algún valor a un actor: Un caso de uso debe de entregar algún tipo de valor tangible al usuario.

Un caso de uso es completo: debe completar una descripción. Un error común es dividir un caso de uso en otros más pequeños que se llaman entre sí, cual funciones en un lenguaje de programación. Un caso de uso no está completo hasta que el valor final se produce, aún si se llevan a cabo comunicaciones durante el desarrollo del mismo.

Un caso de uso es una clase y no una instancia. Describen la funcionalidad como un todo, incluyendo alternativas posibles, errores y excepciones que pueden darse durante el caso de uso. Una instancia de un caso de uso se conoce como *escenario* y representa un uso específico del sistema.

Algunas preguntas que pueden ayudar a encontrar los casos de uso del sistema son:

- ¿Qué funciones requiere el actor del sistema? ¿Qué necesita hacer el actor?
- ¿El actor necesita leer, crear, modificar o almacenar algún tipo de información en el sistema?
- ¿El actor tiene que ser notificado de eventos en el sistema o tiene que notificar algo?
- ¿Pueden simplificarse o hacerse más eficientes los actores diarios a través de una nueva funcionalidad en el sistema?
- ¿Qué entrada/salida necesita el sistema? ¿De dónde viene y a dónde va esta entrada/salida?
- ¿Cuáles son los mayores problemas con la actual implementación del sistema (en caso de existir)?

Una buena fuente para identificar los casos de uso son los eventos externos. Piense en todos los eventos del mundo exterior ante los cuales queremos reaccionar. Un evento dado puede provocar una reacción en el sistema en el que no intervenga el usuario, o puede

causar una reacción de los eventos ante los que se necesitará reaccionar será de ayuda en la identificación de los casos de uso.

Un caso de uso se representa en UML por una elipse, conteniendo el nombre del caso de uso o con el nombre del caso de uso debajo de la elipse.

A continuación presentamos el ejemplo de representación en UML de los elementos de caso de uso descritos anteriormente:

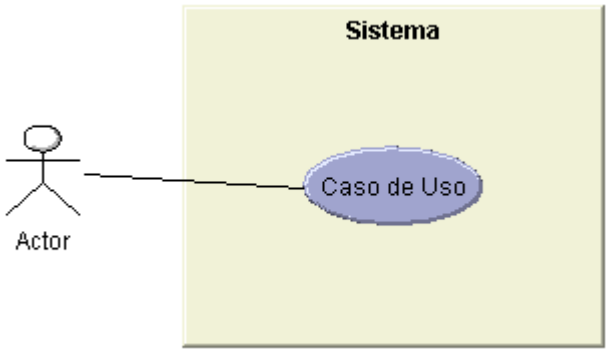


Fig. I Ejemplo de diagrama de Casos de Uso

Relaciones entre casos de uso

Existen tres tipos de relación entre casos de uso: Extiende (*extends*), usa (*uses*) y agrupa (*grouping*). Las relaciones extiende y usa, son dos tipos de herencia. Agrupa es un modo de colocar casos de uso relacionados en un paquete.

? Relación "*extiende*"

Se usa la relación *extiende* cuando se tiene un caso de uso que es similar a otro, pero que hace un poco más.

Es una relación de generalización donde un caso de uso extiende de otro, agregando acciones a un caso general. El caso de uso que extiende puede incluir comportamiento del caso de uso que es extendido, dependiendo de las condiciones de extensión.

Entonces podemos entender que aplicamos la relación *extends* cuando encontramos una variación en el caso de uso. También puede suceder que el caso de uso se tenga que dividir por su complejidad.

? Relación “usa”

Las relaciones *usa* ocurren cuando se tiene una porción de comportamiento que es similar en más de un caso de uso y no se quiere copiar la descripción de tal conducta. Es una relación de generalización donde cada caso de uso utiliza otro caso de uso, indicando que es parte del caso de uso especializado. El comportamiento del caso de uso general deberá ser incluido por completo.

Debemos poner énfasis en la diferencia entre *uses* y *extends*. Ambos implican la factorización de comportamientos comunes, dejando un caso de uso como principal o común, sin embargo la interacción con los actores es la que cambia. En la relación *extends* los actores tienen que ver con todos los casos de uso, el común y los extendidos, y en la relación *uses* puede o no haber un actor asociado a los casos de uso.

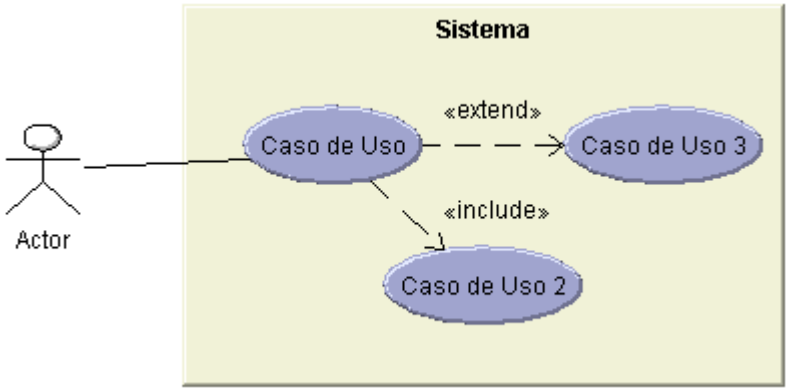


Fig. II Ejemplo de relaciones utilizadas en los diagramas de Casos de Uso

Escenario

En UML se refiere a una sola ruta a través de un caso de uso, una ruta que muestra una particular combinación de condiciones dentro de dicho caso de uso. Puede generar un éxito o un fracaso[24].

Diagramas de Clase

El diagrama de clase describe los tipos de objeto que hay en el sistema y las relaciones estáticas que existen entre ellos. Hay dos tipos principales de relaciones estáticas: las asociaciones y los subtipos[25].

Los diagramas de clase también muestran los atributos y operaciones de una clase.

[24] FOWLER, Martin, KENDALL, Scott. *UML Gota a Gota*. Adison Wesley. 1997. México. Pág. 57
[25] Ídem.

Podemos tener tres perspectivas al dibujar los diagramas de clase, la conceptual, de especificación y la de implementación. La primera representa conceptos del dominio y debe de ser independiente al lenguaje de programación, es decir, se consideran independientes del lenguaje, se utiliza en la etapa de análisis del sistema. Desde el punto de vista de especificación estamos viendo las interfaces del software, por lo que vemos tipos y no clases. Un tipo representa una interfaz que puede tener muchas implementaciones distintas, de acuerdo al ambiente de implementación, esto toma importancia cuando dependiendo del tipo se delegan responsabilidades en la parte del diseño. En la perspectiva de implementación tenemos clases y se apegan totalmente a la implementación escogida.

Es importante que cuando dibujemos un diagrama de clases lo hagamos desde una sola perspectiva, y que cuando lo leamos, utilicemos la perspectiva con la que se dibujó.

Las asociaciones representan relaciones entre instancias de clase. Cada asociación tiene dos papeles, cada papel en una dirección en la asociación. Un papel tiene también una multiplicidad, la cual es la indicación de la cantidad de objetos que participarán en la relación. En general la multiplicidad indica los límites inferior y superior de los objetos participantes.

Las asociaciones las identificamos con el enunciado “ usa un..”, los objetos involucrados son independientes y son transitorias por un periodo de tiempo.

Dentro de la perspectiva de especificación, las asociaciones representan responsabilidades. Podemos agregar flechas a las líneas de asociación para indicar la navegabilidad. La navegabilidad es importante en las perspectivas de especificación e implementación.

En los diagramas de clases también se representan los atributos y operaciones, los primeros con características de los objetos y tienen un valor; los segundos son los procesos que una clase sabe llevar a cabo. Debemos distinguir entre método y operación, una operación es algo que se invoca sobre un objeto, y un método es el cuerpo del procedimiento.

Dependiendo del detalle del diagrama la notación para un atributo en UML es: *visibilidad nombre: tipo=valor por omisión*. Para una operación es *visibilidad nombre (lista de argumentos): valor de retorno*, donde la visibilidad es + (*public*), # (*protected*), o – (*private*).

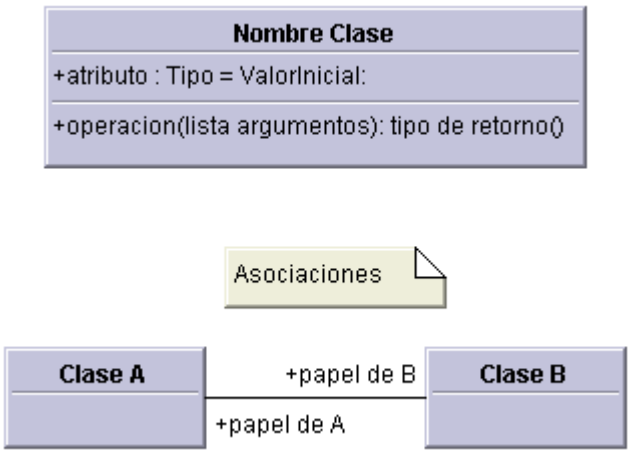


Fig. III Ejemplo de diagrama de clases

La generalización es otro aspecto que debemos de tomar en cuenta cuando realicemos los diagramas de clases, dependiendo de la perspectiva del diagrama. Si es de tipo conceptual, la generalización representa subtipos y todo lo que digamos del tipo es válido para el subtipo; sí es a nivel de especificación significa que la interfaz del subtipo debe de incluir todos los elementos de la interfaz del supertipo, y a nivel de implementación estamos hablando de herencia, es decir, la subclase hereda todos los métodos y campos de la súper clase.

Los atributos, las asociaciones y la generalización nos permiten especificar las condiciones importantes de nuestro problema, sin embargo, si queremos especificar mas restricciones, podemos escribirlas brevemente en el diagrama entre corchetes{ }.

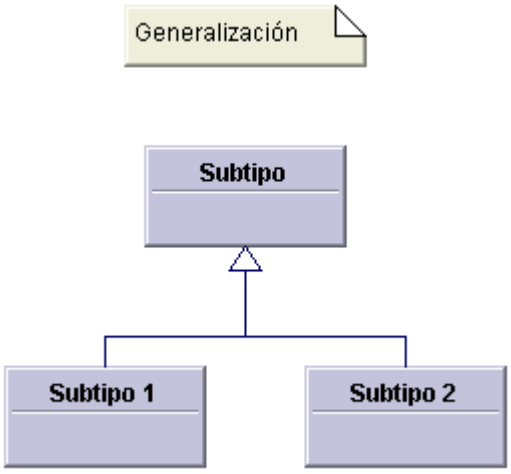


Fig. IV Ejemplo de generalización en los diagramas de clases

EL UML ha adoptado un concepto de OO llamado *estereotipo* para sugerir responsabilidades en las clases. Para esto vamos a clasificar las clases del sistema en tres tipos: objetos de interfaz, objetos de control y objetos de entidad; esta clasificación nos define las clases que llamaremos estereotipos. Un estereotipo se representa con comillas francesas <<>>.

Con lo anterior ya podemos realizar diagramas de clases bastante completos, sin embargo existen otros conceptos avanzados para realizar los diagramas de clases detallados, entre los que destacan la clasificación múltiple, composición, agregación, clase abstracta, entre otras. A continuación describiremos sólo las más importantes.

La *clasificación* se refiere a la relación entre un objeto y su tipo. En la clasificación simple un objeto pertenece a un solo tipo y en la múltiple, un objeto puede ser descrito por varios tipos que no están conectados necesariamente por medio de la herencia. Es importante señalar que la clasificación múltiple y la herencia múltiple son diferentes, debido a que la herencia múltiple plantea que un tipo puede tener muchos súper tipos, pero debe definir un sólo tipo por cada objeto. La clasificación múltiple en cambio permite varios tipos para un objeto sin definir un tipo específico para tal fin.

Al utilizar la clasificación múltiple debemos de tener cuidado en respetar las combinaciones legales entre los tipos, para esto etiquetaremos la línea de generalización con un discriminador, el cual indicará la subtipificación. La convención nos indica que todas las subclases que emplean un discriminador desemboquen en un triángulo.

La agregación es la relación de componente, es decir, un objeto es componente o parte de otro objeto, como un motor es componente a un auto. Se identifica por el enunciado “tiene un..”, los objetos componentes son intercambiables, la relación es más fuerte, deben de existir las partes para formar un todo. Se representan con un rombo vacío, que llega al todo.

La composición es un tipo de agregación donde el objeto parte puede pertenecer a un todo único, y se espera que las partes vivan y mueran con el todo. Se identifica con el enunciado “tiene un..”, los objetos componentes no son intercambiables. La relación es aún mas fuerte, debe existir un todo para que existan las partes. Se representa con un rombo lleno, que llega al todo.

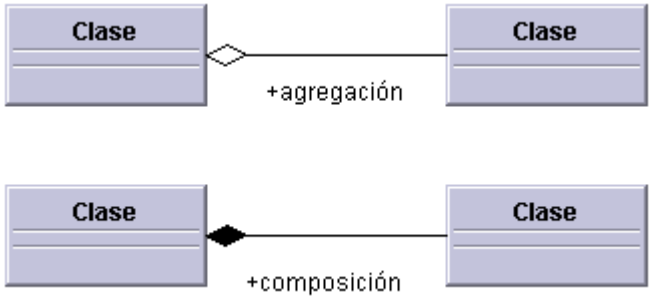


Fig. V Ejemplos de agregación y composición en los diagramas de clases

Diagramas de interacción

Son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento[26].

Un diagrama de interacción capta el comportamiento de los objetos para un sólo caso de uso. Este diagrama muestra ejemplos de los objetos y los mensajes que se pasan entre los objetos en el caso de uso particular. Podemos decir que estamos viajando a través de un escenario de un caso de uso.

Hay dos tipos de diagramas de interacción los de secuencia y los de colaboración.

- *Diagramas de Secuencia*

  En los diagramas de secuencia, los objetos se muestran como cajas en la parte superior de líneas verticales punteadas, una por objeto. A esta línea se le conoce como línea de la vida del objeto y representa la vida del objeto durante la interacción.

  Cada mensaje se representa mediante flechas entre las líneas de la vida de dos objetos. El orden en que se dan los mensajes transcurre de arriba hacia abajo. Cada mensaje debe de ser etiquetado, y puede darse el caso de la auto delegación, cuando un objeto se envía un mensaje a sí mismo.

  Debemos destacar que en estos diagramas se representan las condiciones, siempre verdaderas, que indican cuando se envían los mensajes. Y los marcadores de iteración que se indican con *, y la condición entre corchetes [], cuando un mensaje se envía muchas veces a varios objetos.

  En este diagrama también se incluye un regreso, el cual indica el regreso de un mensaje, no un nuevo mensaje. Los regresos se representan con líneas punteadas. No es recomendable utilizar muchos regresos, pues puede saturar el diagrama.

  Los diagramas de secuencia son valiosos también para ejemplificar los procesos concurrentes, donde tenemos procesos que son llamados asincrónicamente y operan en paralelo. Para estos procesos deben de existir nuevos elementos en el diagrama, como las activaciones, que aparecen explícitamente cuando un método esta activo, éstas se representan por rectángulos vacíos en la línea de vida del objeto.

  Los mensajes asíncronos los representamos con medias cabezas de flecha, estos mensajes no bloquean al invocador y pueden crear un nuevo proceso, crear un nuevo objeto o comunicarse con un proceso que ya está operando.

  El último elemento de los diagramas de secuencia recurrentes es el borrado de un objeto que se muestra con una X grande.

[26] FOWLER, Martin, KENDALL, Scott. *UML Gota a Gota*. Adison Wesley. 1997. México. Pág. 115

También tenemos que tener en cuenta el tipo de relación existente entre los objetos del diagrama de secuencia, pues si tenemos una relación de asociación, las relaciones son temporales, pero si estamos hablando de una composición, un todo y sus partes, todos los objetos se crean al mismo tiempo y comparten la línea de la vida.

? *Diagramas de Colaboración.*

Los diagramas de colaboración son la segunda forma de los diagramas de iteración. En este caso particular los objetos son representados como íconos, las flechas indican los mensajes enviados en un caso de uso dado, sin embargo la secuencia se indica numerando los mensajes.

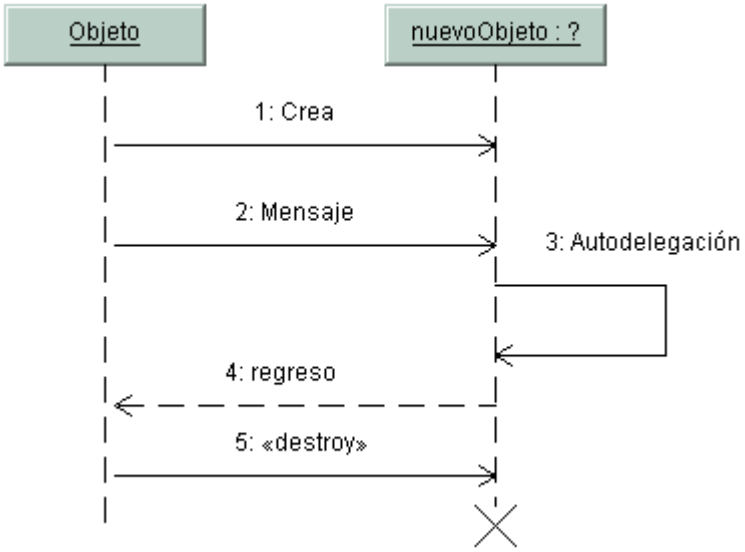


Fig. VI Ejemplo de diagrama de secuencia

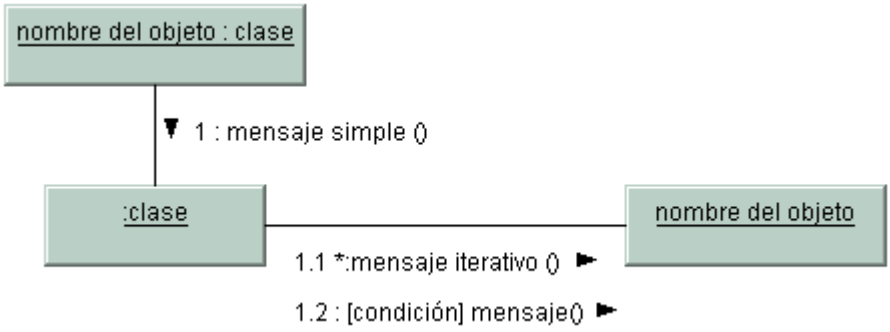


Fig. VII Ejemplo de diagrama de colaboración

Diagramas de Paquetes

Al agrupamiento de las clases en unidades de nivel más alto se le conoce como *paquete*. En UML emplearemos el término de diagrama de paquetes para indicar un diagrama que muestra los paquetes de clases y las dependencias entre ellos.

Decimos que existe dependencia entre dos elementos cuando los cambios a la definición de un elemento pueden causar cambios en el otro. La dependencia entre las clases puede ser porque: una clase envía un mensaje a otra clase; una clase tiene a otra como parte de sus datos; una clase es parámetro de otra en una operación.

Existe una dependencia entre dos paquetes si existe algún tipo de dependencia entre dos clases cualquiera en los paquetes. Existe similitud entre la dependencia de paquetes y dependencia de compilación, pero las dependencias en los paquetes no son transitivas.

Otro aspecto importante entre los paquetes es la visibilidad de las clases contenidas en ellos. Definiremos entonces dos tipos de visibilidad la opaca y la transparente. Decimos que tenemos visibilidad opaca cuando no podemos ver los paquetes anidados, sólo el paquete contenedor y transparente cuando podemos ver todos los paquetes incluyendo los anidados. Esto también depende del lenguaje con el que se desarrolle el sistema (Java, C++, etc.).

Para mostrar el contenido de un paquete debemos de colocar el nombre del paquete en la parte superior izquierda con una etiqueta y el contenido dentro del cuadro principal. El contenido puede ser la lista de clases, otro diagrama de paquetes, o un diagrama de clases.

En los diagramas de paquetes podemos aplicar la generalización, y por lo tanto algunas notaciones de los diagramas de clases. La generalización nos llevará a que exista dependencia entre el paquete supertipo y el paquete subtipo.

Los diagramas de paquetes sirven para tener un mejor control del sistema, nos ayudan a identificar las dependencias y con esto podemos reducirlas para tener un bajo acoplamiento.

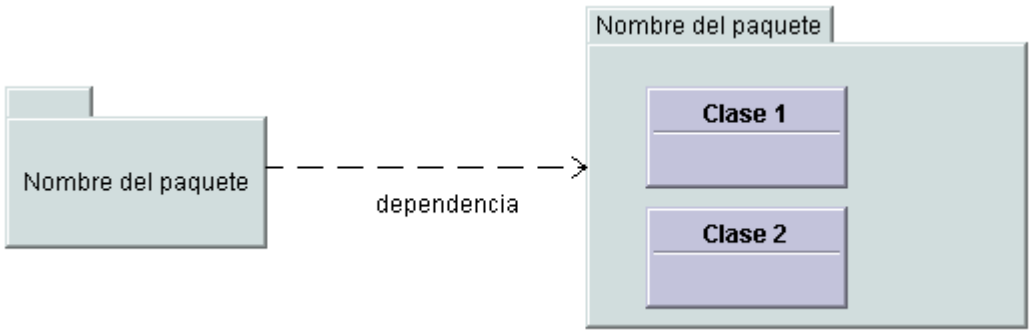


Fig. VIII Ejemplo de diagrama de paquetes

Diagramas de Estados

Los diagramas de estados nos muestran el comportamiento del sistema. Describen todos los estados posibles en los que puede estar un objeto específico y como cambia de estado durante su ciclo completo de vida.

El diagrama empieza en un punto de partida, y se traslada a los diversos estados, representados por rectángulos con las puntas redondeadas, conectados a través de flechas que nos indican la navegabilidad. Las flechas que identificaremos como la transición de un estado a otro, deben de estar etiquetadas siguiendo la sintaxis *Evento[Guardia]/Acción*, las partes de la etiqueta son optativas, dependiendo del caso.

Si los estados tienen actividad asociada, ésta se coloca dentro del rectángulo del estado, etiquetando la actividad con la sintaxis *hace/actividad*.

Las acciones se asocian con las transiciones y se consideran como procesos que se ejecutan con rapidez y no son interrumpibles. Las actividades se asocian con los estados, pueden ser interrumpidas y durar más. Ambas son métodos.

Un guardia es una condición lógica que sólo devuelve verdadero o falso, por lo que una transición de guardia solo se resuelve como verdadero. Los guardias deben ser mútuamente excluyentes para cualquier evento, así solo se puede tomar una transición de un estado dado.

En estos diagramas también podemos aplicar lo ya visto anteriormente, la generalización y por lo tanto tener superestados, donde los subestados heredan todas las transiciones del superestado.

Si un estado responde a un evento con una acción que no produce una transición, dicha condición se pone dentro del cuadro de estado con la sintaxis *nombreEvento/nombreAccion.*

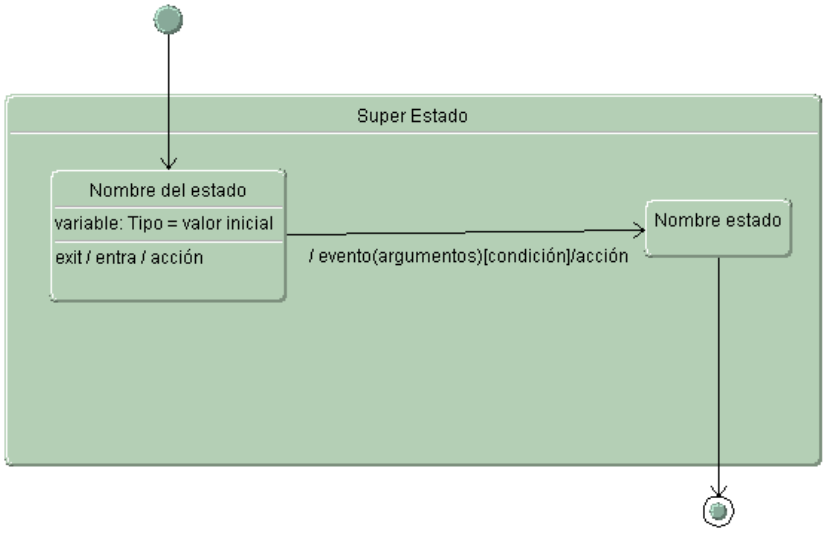


Fig. IX Ejemplo de Diagrama de estado

Existe un caso particular donde una transición vuelve al mismo estado que la produjo, a esto le llamaremos auto transición.

Cuando un objeto puede estar en dos estados diferentes, cada uno representado por un diagrama, decimos que podemos utilizar un diagrama de estados concurrentes. Así en este tipo de diagramas podemos combinar los comportamientos de un objeto. Sin embargo debemos de tener cuidado en que los objetos no tengan muchos conjuntos de comportamientos concurrentes, si es así sería mejor dividir el objeto.

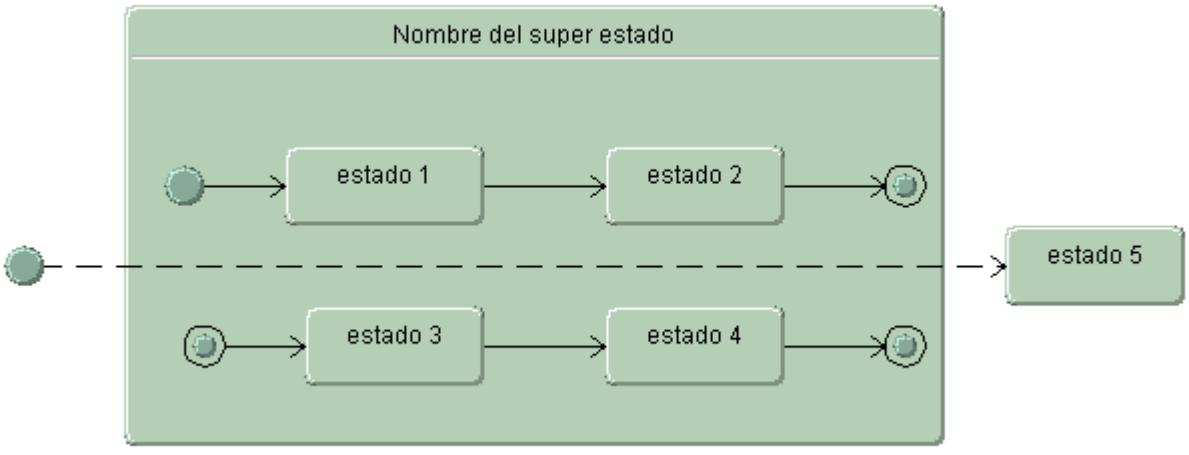


Fig. X Ejemplo de diagramas de estados concurrentes

Es importante no perder de vista que los diagramas de estado nos muestran el comportamiento de los objetos a través de varios casos de uso, sin embargo debemos de tomar la decisión de sólo hacer diagramas de estados de las clases que tengan un comportamiento interesante o importante en nuestro desarrollo y evitar diagramas de todos los objetos.

Diagramas de actividades.

Los diagramas de actividad pretenden mostrar la secuencia general de las acciones de varios objetos y casos de uso. En este tipo de diagramas, el concepto clave es la actividad, el cual dependerá de la perspectiva que tomemos para realizar el diagrama. Si interpretamos el diagrama de manera conceptual, una actividad es cierta tarea que debe ser llevada a cabo. Si lo vemos desde el punto de vista de especificación o implementación, una actividad es un método de una clase.

Este diagrama, nos presentará actividades seguidas de actividades y podríamos verlo como un diagrama de flujo, sin embargo cuenta con elementos que lo diferencian y enriquecen. Entre estos elementos vamos a encontrar los rombos de decisión, los guardias, que son expresiones lógicas que se evalúan como verdadero o falso; barras de sincronización, las cuales indican actividades realizadas en paralelo, intercaladas o simultáneas, no importando el orden de ejecución. Las anteriores son las principales diferencias con un diagrama de flujo, los diagramas de actividades indican las reglas de secuenciación y son útiles cuando tenemos procesos concurrentes o necesitamos sincronizar procesos.

Cuando representemos casos de uso con los diagramas de actividad, vamos a tener elementos adicionales como son un disparador de inicio, representado por un círculo relleno.

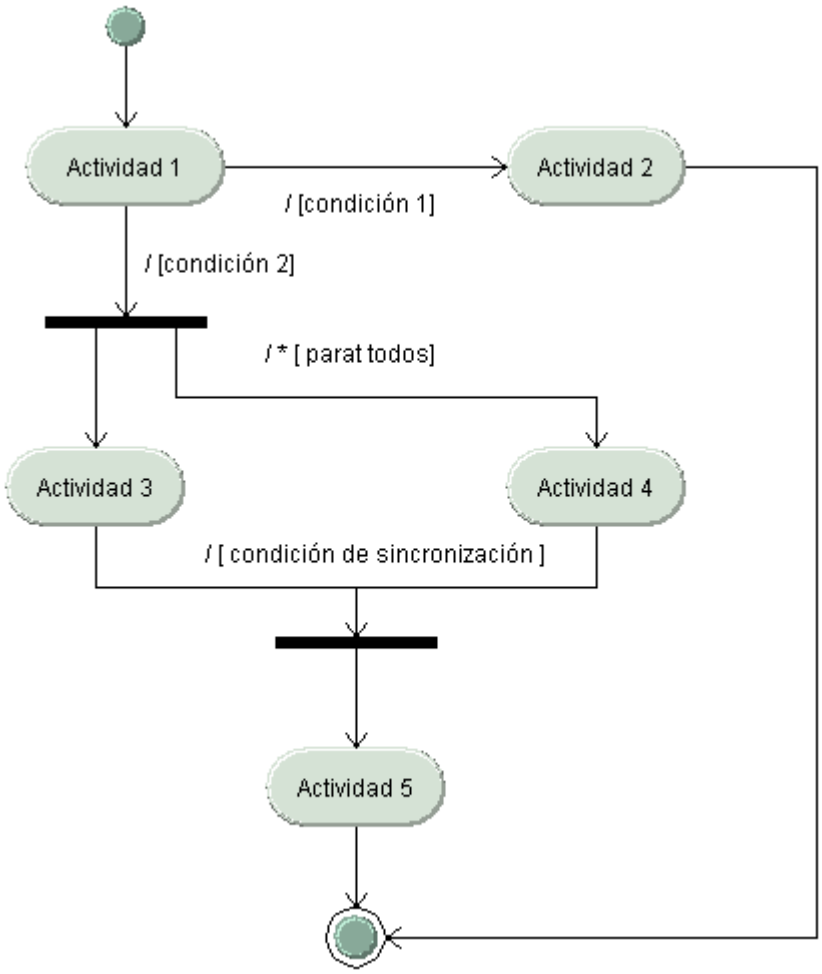


Fig. XI Ejemplo de Diagrama de Actividades

3.1.4 Patrones para asignar responsabilidades

Un sistema orientado a objetos se compone de objetos que envían mensajes a otros objetos para que lleven a cabo las operaciones. En los contratos se incluye una conjetura inicial óptima sobre las responsabilidades y poscondiciones de las operaciones. Los diagramas de interacción describen gráficamente la solución (a partir de los objetos en interacción o en colaboración) que estas responsabilidades y poscondiciones satisfacen.

La calidad de diseño de la interacción de objetos y la asignación de responsabilidades presenta gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en principios que rigen un buen diseño. En los patrones se codifican algunos de éstos principios, que se aplican para crear diagramas de interacción, cuando se asignan responsabilidades.

Responsabilidades y métodos

Una “responsabilidad” se define como “un contrato u obligación de un tipo de o clase”. Las responsabilidades se relacionan con las obligaciones de un objeto respecto a su comportamiento.

La asignación de responsabilidades a menudo se asignan en el momento de preparar los diagramas de colaboración.

Las responsabilidades se implementan usando métodos que operan solos o con otros métodos y objetos.

Patrones

Los diseñadores expertos en orientación a objetos van formando un amplio repertorio de principios generales y de expresiones que lo guían al crear software. A unos y a otras se puede asignar el nombre de patrones si se codifican en un formato estructurado que describe el problema y su solución, y si se le asigna un nombre.

Resumiendo lo anterior, un patrón es una pareja de problema / solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.

Entre los patrones más importantes y utilizados se encuentran:

- Experto
- Creador
- Alta cohesión
- Bajo acoplamiento
- Controlador

A continuación se presentan cuadros sinópticos que explican cada uno de los patrones anteriormente citados y que serán utilizados durante el presente trabajo para la asignación de responsabilidades:

Experto	
Solución	Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir con la responsabilidad.
Problema	¿Cuál es el principio fundamental en virtud del cuál se asignan las responsabilidades en el diseño orientado a objetos?

Explicación	Experto es el patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en OO. Con él no se pretende asignar una idea oscura ni extraña; simplemente la intuición de que los objetos hacen cosas relacionadas a la información que poseen. El patrón experto da origen al diseño donde el objeto de software realiza las operaciones que normalmente se aplican a la cosa que representa. El patrón experto ofrece una analogía del mundo real (como otras tantas cosas en la tecnología orientada a objetos). Acostumbramos asignar responsabilidades a individuos que disponen de la información necesaria para llevar a cabo una tarea. Por ejemplo, en una empresa ¿Quién sería el encargado de preparar un informe de pérdidas y ganancias?. El empleado que tiene acceso a toda la información, tal vez, el director financiero.
Beneficios	Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer los que se les pide.
Otras formas de designarlo	“Juntar responsabilidades y la información”, “lo hace el que conoce”, “animación”, “Lo hago yo mismo”

TABLA I. Patrón experto

Creador	
Solución	Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos: B *agrega* los objetos A B *contiene* los objetos A B *registra* las instancias de los objetos A B *utiliza* específicamente los objetos A B *tiene los datos de inicialización* que serán transmitidos a A cuando éste objeto sea creado(así que B es un experto en cuanto a la creación de A)
Problema	¿Quién debería ser responsable de crear una nueva instancia de la clase A?
Explicación	El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El patrón creador indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada, desde luego se trata de una directriz. En ocasiones encontramos un patrón creador buscando la clase con los datos de inicialización que serán transferidos durante la creación. Éste es en realidad un ejemplo del patrón experto.
Beneficios	Se brinda soporte a un bajo acoplamiento
Patrones conexos	Bajo Acoplamiento Parte-Todo

TABLA II. Patrón Creador

Bajo acoplamiento	
Solución	Asignar una responsabilidad para mantener bajo acoplamiento
Problema	¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? El acoplamiento es una medida de fuerza con que una clase está conectada a otras clases, con las que conoce y con las que recurre a ellas. Una clase con bajo o débil acoplamiento no depende de muchas otras. Una clase con alto acoplamiento recurre a muchas otras, estas clases no son convenientes por que: Los cambios de las clases afines ocasionan cambios locales Son más difíciles de entender cuando están aisladas Son más difíciles de reutilizar por que se requiere de la presencia de las clases de las que dependen
Explicación	El bajo acoplamiento es un principio que debemos recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. El bajo acoplamiento no debe considerarse en forma independiente de otros patrones como Experto o Creador, sino más bien, ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades. No existe una medida absoluta de cuándo el acoplamiento es excesivo. Lo importante es que el diseñador pueda determinar el grado actual de acoplamiento y si surgirán problemas en caso de incrementarlo. En términos generales, han de tener escaso acoplamiento las clases muy genéricas y con grandes probabilidades de reutilización. El caso extremo de Bajo Acoplamiento ocurre cuando las clases tienen poco o nulo acoplamiento lo que da origen a un diseño deficiente donde las clases son incoherentes, atiborradas y complejas hacen todo el trabajo, con muchos otros objetos muy pasivos y de acoplamiento cero que funcionan como meros depósitos de datos.
Beneficios	No se afectan por cambios de otros componentes Fáciles de entender por separado

	Fáciles de reutilizar
Otras formas de designarlo	

TABLA III. Patrón bajo acoplamiento

Alta Cohesión	
Solución	Asignar una responsabilidad de modo que la cohesión siga siendo alta.
Problema	¿Cómo mantener la complejidad dentro de límites manejables? En la perspectiva del diseño orientado a objetos, la cohesión, (o más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas: Son difíciles de comprender Son difíciles de reutilizar Son difíciles de conservar Son delicadas: las afectan constantemente los cambios
Explicación	Como el patrón de bajo acoplamiento, también la alta cohesión es un principio que debe tenerse en cuenta para decisiones de diseño. Es un patrón evaluativo que el diseñador aplica para al valorar sus decisiones. A continuación se mencionan algunos escenarios que ejemplifican los diversos grados de la cohesión funcional: *Muy baja cohesión*. Una clase es la única responsable de muchas cosas en áreas funcionales muy heterogéneas. *Baja Cohesión*. Una clase tiene la responsabilidad exclusiva de una tarea

	compleja de un área funcional. *Alta cohesión.* Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. *Cohesión moderada*. Una clase tiene un peso ligero y responsabilidades exclusivas de unas cuantas áreas que están relacionadas lógicamente con el concepto de la clase, pero no entre ellas.
Beneficios	Mejoran la claridad y la facilidad con que se entiende el diseño Se simplifican el mantenimiento y las mejoras en funcionalidad. A menudo se genera un bajo acoplamiento. La ventaja de una gran funcionalidad soporta mayor capacidad de reutilización por que una clase muy cohesiva puede asignarse a un trabajo muy específico.
Otras formas de designarlo	

TABLA IV. Patrón alta cohesión.

Controlador	
Solución	Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones: El “sistema” global (controlador de fachada) La empresa u organización global (controlador de fachada) Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y en que pueda participar en la tarea (controlador de tareas) Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados “Manejador<Nombre CasodeUso> (controlador de casos de uso) *Corolario:* Nótese que en esta lista no figuran las clases “ventana”, “aplicación”, “vista”, ni “documento”. Estas clases *no* deberían ejecutar las tareas asociadas a los eventos del sistema; generalmente las reciben y las delegan al controlador.
Problema	¿Quién debería encargarse de entender un evento del sistema? Un evento de sistema es un evento de alto nivel generado por actor externo; es un evento de entrada externa. Se asocia a operaciones del sistema: las que emite en respuesta a los eventos del sistema.
Explicación	La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (GUI) operado por una persona. Otros medios de entrada son los mensajes externos, entre ellos un conmutador de telecomunicaciones para procesar llamadas, o las señales procedentes de sensores como sucede en los sistemas de control de procesos. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. La primera categoría de controlador es un controlador de fachada que representa al “sistema” global. Los controladores de fachada son adecuados cuando el sistema sólo tiene unos cuantos eventos o cuando es imposible redirigir los mensajes de los eventos del sistema a otros controladores,

	como sucede en un sistema de procesamiento de mensajes. Si se recurre a la cuarta categoría de controlador (un "manejador artificial de casos de uso"), habrá entonces un controlador para cada caso. Adviértase que éste no es un objeto del dominio; es un concepto artificial cuyo fin es dar soporte al sistema (una *fabricación pura*, en términos de patrones de GRASP). ¿Cuándo deberíamos escoger un controlador de casos de uso? Es una alternativa que debe tenerse en cuenta si el hecho de asignar las responsabilidades es en cualquiera de las otras opciones de controlador genera diseños de baja cohesión o alto acoplamiento.
Beneficios	*Mayor potencial de los componentes reutilizables*. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de objetos de dominio y no por la de interfaz. *Reflexionar sobre el estado de los casos de uso*. A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones del caso de uso subyacente.
Otras formas de designarlo	

TABLA V. Patrón controlador

3.1.5 El proceso de desarrollo

Llamamos proceso de desarrollo de software al método de organizar las actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de software.

Así como los paradigmas de programación han evolucionado; el proceso de desarrollo de software lo ha hecho también.

En un principio se tenía un enfoque muy razonable: el proceso secuencial. Muchos problemas en ingeniería son resueltos utilizando un proceso secuencial, el cual típicamente consta de los siguientes cinco pasos:

1. Entender completamente el problema por resolver, sus requerimientos y sus reglas. Capturarlos en un texto y mostrarlo a todas las partes interesadas para que estén de acuerdo con la descripción de sus necesidades.

2. Diseñar una solución que satisfaga todos los requerimientos y reglas. Examinar este diseño cuidadosamente y estar seguro que todas las partes interesadas están de acuerdo en que se trata de la mejor solución.

3. Implementar la solución utilizando las mejores técnicas de ingeniería.

4. Verificar que la implementación satisface los requerimientos.

5. Entrega o Liberación.

En este enfoque hay características importantes en las que debemos hacer un alto y reflexionar acerca de ellas.

En primer lugar, es la forma en las que son concebidas las *fases* (el proceso de desarrollo de software tradicionalmente es realizado en una serie de pasos o, como son comúnmente conocidos, “fases”). Estas fases son un conjunto de actividades que usan documentos como entrada y producen otros nuevos y, muy probablemente, diferentes documentos como salida.

A pesar de que el enfoque secuencial parece ser lógico, muchos proyectos de desarrollo de software que lo han seguido han fracasado. ¿Por qué?, parece haber algunas razones clave:

- Hemos asumido algunos hechos erróneos:

    1. Notemos que en el paso 1 del enfoque secuencial asumimos que se puede captar la esencia del problema desde el principio, no hay nada más falso en el desarrollo de sistemas. Los requerimientos no se “congelan” a lo largo del proceso, los requerimientos cambian constantemente, cambian a la par que los usuarios cambian, que el problema evoluciona, que la tecnología cambia y que los mercados cambian. Además, los requerimientos cambian por el hecho comprobado de que no podemos captar los requerimientos con el suficiente detalle y precisión.

    2. En el segundo paso del proceso secuencial asumimos que podemos confirmar que nuestro diseño es la correcta solución al problema. Al respecto, los estudiosos de los procesos de desarrollo, también han comprobado que un diseño en “papel” no puede ser completamente comprobados hasta que se implementan en código, a diferencia de los planos de una casa.

- El contexto de desarrollo de software es diferente al de otras disciplinas de ingeniería.

- Se han incorporado factores humanos que nos pueden hacer fracasar.

- Hemos tratado de hacer encajar un enfoque que funciona sólo bajo circunstancias bien definidas. El proceso de *cascada* o secuencial ha funcionado para muchos líderes cuyas duraciones de los proyectos van desde una par de semanas hasta unos cuantos meses, en aquellos proyectos en los cuales se puede anticipar con suficiente claridad qué es lo que va a ocurrir y en aquellos proyectos en los que los detalles finos son completamente entendidos. En los proyectos que son poco novedosos se

puede desarrollar y llevar a cabo un plan con poca o ninguna sorpresa. Sin embargo, el proceso secuencial se viene abajo si se atacan proyectos con un nivel significativo de novedad, desconocidos o riesgosos.

- Estamos aún en una etapa exploratoria de la ingeniería de software. No se tiene la experiencia de cientos de años o las pruebas y errores que se tienen en la construcción de puentes. Ésta es, tal vez, la razón más importante.

Si el proceso secuencial parece funcionar bien y resultar exitoso en proyectos pequeños o en aquellos con un nivel mínimo de riesgo y poco novedosos, ¿por qué no descomponer el proceso de desarrollo de un proyecto largo en una sucesión de pequeños proyectos secuenciales? De esta forma, se pueden manejar algunos requerimientos y algunos riesgos, diseñar un poco, implementar un poco y validar; tomar otros pocos requerimientos, diseñar un poco más, construir un poco más, validar y así sucesivamente hasta que terminemos. A esto se le llama un enfoque *Iterativo*.

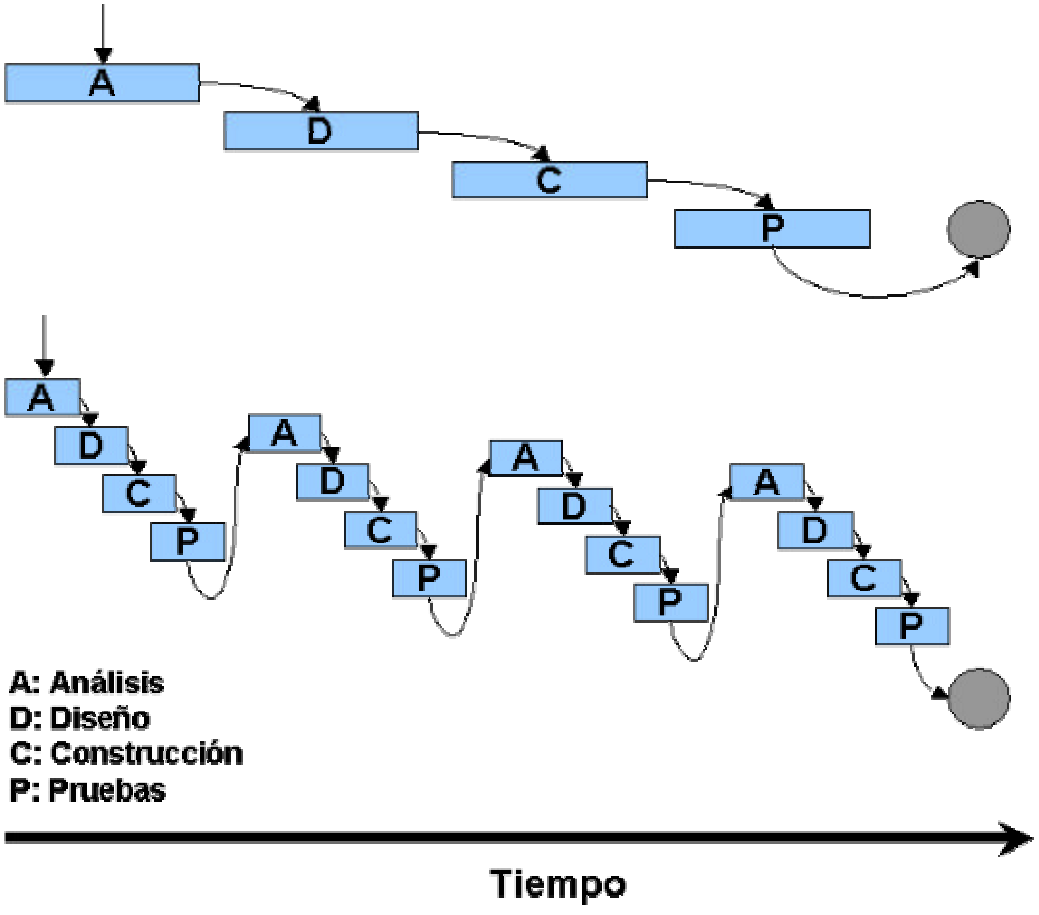


Fig. XII Del proceso secuencial al proceso iterativo

El proceso iterativo esta organizado en fases, como se muestra en la siguiente figura. A diferencia del enfoque secuencial, estas fases no están organizadas de la manera tradicional: análisis, diseño, desarrollo, implementación y pruebas.

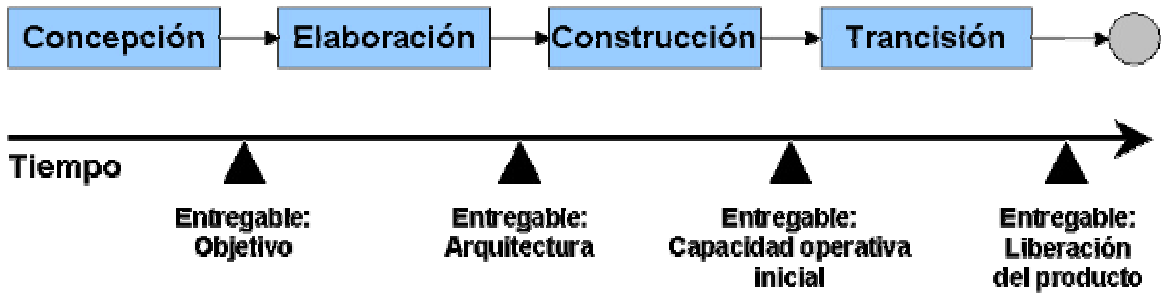


Fig. XIII Las cuatro fases y entregables del proceso iterativo

Ahora, haremos una breve descripción de estas fases:

? **Concepción**[27].

Es aquí donde realizamos la especificación del producto final y definimos el alcance del proyecto.

Foco:

Se concentra en entender totalmente los requerimientos y determinar el alcance de la oferta de desarrollo.

Objetivos:

- Establecer el alcance del proyecto de software y las condiciones implícitas, incluyendo un concepto operacional, criterios de aceptación y qué es lo que está y lo que no está previsto que haga el sistema.
- Discriminar los *casos de uso* críticos del sistema, es decir, los escenarios primarios que definen el comportamiento funcional del sistema.
- Exhibir y, tal vez demostrar, por lo menos una arquitectura.
- Estimar el costo total y la calendarización de todo el proyecto, así como calcular las fechas detalladas para la fase de elaboración.
- Estimar los riesgos.

[27] El término en inglés utilizado para esta fase es *Inception*, que significa: *El principio de algo que no es comprendido*; sin embargo, la mayoría de la bibliografía en castellano la llama *Concepción*.

Actividades esenciales:

- Formular el alcance del proyecto, esto es, capturar el contexto y los requerimientos más importantes, así como las reglas que se manejarán para los criterios de aceptación del producto final.

- Planear y preparar los *casos de negocio* y evaluar alternativas para la administración del riesgo, personal, planeación y la compensación entre costos, calendario y rentabilidad.

- Sintetizar la arquitectura candidata, evaluando la compensación de diseño y decisiones que determinen qué hacer, comprar o reutilizar para que los costos y recursos puedan ser estimados.

Productos:

- El documento de *visión*, que es el documento que describe la visión general de los requerimientos del proyecto, las características primarias y las principales reglas.

- El modelo de casos de uso, que es la lista de todos los casos de uso y los actores que pueden ser identificados en esta fase temprana.

- Un glosario inicial.

- Un modelo de negocio inicial, que incluye:

  - El contexto del negocio

  - Los criterios de éxito

  - Una proyección financiera

- Una determinación inicial de los riesgos.

- Un plan del proyecto, que muestre las fases y las iteraciones.

Artefactos:

- Modelo de casos de uso (del 10% al 20% completo)

- Un modelo del dominio o diagrama conceptual

- Un modelo de negocios, si es necesario

- Uno o varios prototipos

? **Elaboración.**

En esta fase se realiza la planeación de las actividades y los recursos necesarios. Se especifican las características principales y el diseño de la arquitectura.

F*o*co:

El principal interés de la etapa de elaboración son los requerimientos, aunque algunos diseños de software y su implementación son probados en los prototipos de arquitectura, mitigando ciertos riesgos técnicos al probar soluciones y aprendiendo cómo utilizar ciertas herramientas y técnicas.

Objetivos:

- Definir, validar y crear una referencia de arquitectura tan rápida como práctica.
- Referenciar la visión.
- Crear un plan base para la etapa de construcción.
- Demostrar que la arquitectura soportará la visión por un costo razonable en un tiempo razonable.

Actividades:

- La visión es elaborada y se establece un conocimiento sólido de los casos de uso críticos para manejar las decisiones de planeación y arquitectura.
- El proceso, la infraestructura y el ambiente de desarrollo son creados y los procesos, herramientas y soporte automatizado son puestos en su lugar.
- La arquitectura es elaborada y los componentes seleccionados. Los componentes potenciales son evaluados y las decisiones de hacer/comprar/reutilizar son entendidas lo suficiente para determinar el costo y la calendarización de la fase de construcción con certidumbre. Los componentes arquitectónicos seleccionados son integrados y son retados por los escenarios principales. Las lecciones aprendidas de estas actividades pueden resultar en el rediseño de la arquitectura o reconsideración de los requerimientos.

Productos:

- Un modelo de casos de uso (por lo menos un 80% terminado), en el cual todos los casos de uso han sido identificados en un modelo de casos de uso para supervisión, todos los actores han sido identificados y la mayoría de las descripciones de casos de uso han sido desarrolladas.

- Requerimientos suplementarios que capturen los requerimientos no funcionales y cualquier otro requerimiento no asociado a un caso de uso específico.
- Una descripción de la arquitectura de software.
- Un prototipo ejecutable de la arquitectura.
- Una lista de riesgos revisada y un caso de negocios revisado
- Un plan de desarrollo para todo el proyecto, incluyendo las partes finas del proyecto, que muestre las iteraciones y criterios de evaluación para cada una de ellas.
- Una actualización del caso de desarrollo que especifique el proceso a ser utilizado.
- Un manual de usuario preliminar (opcional).

Criterios de evaluación:

- ¿La visión del producto es estable?
- ¿La arquitectura es estable?
- ¿La demostración ejecutable mostró que los principales elementos de riesgo han sido dirigidos y resueltos?
- ¿El plan de la fase de construcción está suficientemente detallado y adecuado?, ¿Se realizó con base en estimados confiables?.
- ¿Los accionistas sociales están de acuerdo que la visión actual puede ser alcanzada si el plan actual es ejecutado para desarrollar el sistema completo, en contexto con la arquitectura actual?
- ¿Los recursos actuales gastados contra los recursos planeados son aceptables?

? **Construcción.**

En esta fase se construye el producto y evoluciona la visión, la arquitectura y los planes. Al final de la fase de construcción el producto está listo para entregarse a la comunidad de usuarios.

Foco:

La fase de construcción de construcción se enfoca principalmente en el diseño y la implementación. Es aquí donde el prototipo inicial evoluciona y va tomando forma el primer producto operacional.

Objetivos:

- Minimizar los costos de desarrollo optimizando recursos y rechazando las piezas de software inservibles y los re-trabajos.
- Llegar a la calidad adecuada de manera práctica y rápida.
- Llegar a versiones utilizables (alfa, beta, y otras versiones de prueba) de manera práctica y rápida.

Actividades:

- Administración de recursos, control de recursos y optimización de procesos.
- Probar de acuerdo a los criterios de evaluación los componentes completamente desarrollados.
- Evaluar que los productos liberados cumplan con los criterios de visión.

Productos:

- El producto de software integrado en la plataforma adecuada.
- Los manuales de usuario.
- Una descripción de la actual liberación.

Criterios de aceptación:

- ¿Esta liberación del producto es estable y lo suficientemente madura para ser llevada a la comunidad de usuarios?
- ¿Todos los dueños del negocio están listos para el cambio en la comunidad de usuarios?
- ¿Los gastos en los recursos actuales contra los planeados se mantienen aceptables?

? **Transición.**

Esta fase lleva a la transición del producto (de la parte de desarrollo) a los usuarios. Incluye manufactura, entrega, entrenamiento, soporte y mantenimiento del producto hasta que los usuarios estén satisfechos. Concluye con la entrega del producto liberado. Esto da fin al ciclo de desarrollo.

Foco:

La fase de transición se enfoca en que el sistema tenga el nivel correcto de calidad para alcanzar sus objetivos; se tienen que arreglar los errores, entrenar a los

usuarios, ajustar algunas características y agregar elementos faltantes. Se produce y se entrega el producto final.

Objetivos:

- Asegurar que el usuario alcance el auto-soporte.
- Ganar el consenso de los dueños de negocio en que la base de la entrega está completa y es consistente con los criterios de evaluación y con la visión.
- Obtener el producto final base tan rápido y con costos efectivos como práctico.

Actividades:

- Ingeniería de entrega, esto es, empaques comerciales y producción, iniciar las ventas, y entrenamiento de personal.
- Actividades de puesta a punto, incluyendo la corrección de errores y las mejoras en rendimiento y usabilidad.
- Medir las bases de la entrega contra la visión y el criterio de aceptación del producto.

Productos:

- Producto final liberado

Criterios de aceptación:

- ¿Está el usuario final satisfecho?
- ¿Los gastos en los recursos contra los gastos planeados continúan siendo aceptables?

Existen varias metodologías de análisis y diseño orientadas a objetos que utilizan el proceso iterativo, algunas de las más populares son la *Programación Extrema,* XP por sus siglas en inglés y el *Proceso unificado de Rational,* RUP por sus siglas en inglés.

Para nuestro caso en particular no utilizaremos ningún método definido, sólo presentaremos la manera de aplicar el proceso de desarrollo iterativo describiendo un orden posible de actividades y un ciclo de vida del desarrollo. Se aplican los modelos generalmente recomendados según el libro de Larman[28], que es la bibliografía principal de este trabajo de tesis.

[28] LARMAN, Craig. *UML y Patrones*..

3.1.6 Beneficios del enfoque iterativo.

Comparado con el enfoque tradicional del proceso de cascada, el proceso iterativo tiene las siguientes ventajas:

- *Los riesgos son mitigados tempranamente.* Debido a que en el proceso secuencial o de cascada la integración final es el único momento en que los riesgos son descubiertos o remitidos. En cambio, en el proceso iterativo disemina las iteraciones tempranas pasando por todos los componentes del proceso, ejercitando muchos aspectos del proyecto, incluyendo herramientas y habilidades de las gentes. Riesgos percibidos pueden probar no ser riesgos, y lo nuevo: riesgos inesperados pueden ser descubiertos.

- *El cambio es más administrable.* Debido a que los diversos tipos de cambios: cambios en requerimientos, tácticos y tecnológicos. Se van presentando al mismo tiempo que las versiones tempranas del producto final. Permitiendo tomar decisiones con mayor prontitud.

- *Existe un nivel alto de Reutilización.* El proceso iterativo facilita la reutilización porque es fácil identificar las partes que son comunes conforme se va diseñando o implementando parcialmente, en vez de hacerlo al principio.

- *El equipo de trabajo puede aprender continuamente a lo largo del proceso.* Debido a que el equipo empieza pronto su trabajo y lo repite en cada iteración. Se pueden percatar de necesidades de entrenamiento, o incluso de la necesidad de ayuda externa. El proceso por sí mismo puede ser mejorado y refinado a lo largo del camino.

- *El producto tiene en su conjunto una mayor calidad.* Debido a que el sistema es probado muchas veces, se mejora la calidad en cada iteración. Los requerimientos son refinados y relacionados más cercanamente con las necesidades reales de los usuarios. En el momento de la entrega, el sistema ha sido ejecutado múltiples veces.

3.2 Plataforma seleccionada

3.2.1 Python, lenguaje de programación seleccionado.

La selección del lenguaje de programación se basó en dos criterios:

1) Las características y requerimientos de la aplicación, y

2) Las características especiales del lenguaje.

Recordemos que hemos mencionado en varias ocasiones durante el desarrollo del presente trabajo, que una de las características importantes que debe cumplir nuestra aplicación es la

capacidad de ser fácilmente adaptable y escalable[29], además, la solución propuesta debe estar basada en el análisis y diseño orientado a objetos. Esto nos determina el primer criterio de evaluación: el lenguaje seleccionado debe ser uno que cumpla con el paradigma orientado a objetos y que facilite la adaptabilidad y escalabilidad.

Aunque bien es cierto que parte de la adaptabilidad y escalabilidad requeridas serán dadas de manera implícita por el simple hecho de utilizar el paradigma orientado a objetos, al asegurar que los criterios de Mayer sobre modularidad sean cumplidos vía la implementación en clases y sus métodos[30] , en algún momento será necesario que el encargado del mantenimiento al sistema necesite agregar nuevos componentes o modificar alguno de ellos. En este caso debemos considerar la facilidad que el lenguaje presenta para realizar estas tareas.

Pero, ¿Qué característica hace a un lenguaje más fácil de usar en términos de utilizar componentes que ya existen?

Según Jonh K. Ousterhout[31], la característica que define este grado de facilidad de interconexión y utilización de componentes es lo que define como *Tipificado*[32] . Este término se refiere al grado en que el significado de la información es especificado a favor de su uso. En un lenguaje con alto grado de tipificado, el programador declara cómo será utilizada cada pieza de software y el lenguaje previene que la información no sea utilizada de ninguna otra forma. En un lenguaje con tipificado bajo no hay restricciones *a priori* de cómo debe ser utilizada la información; el significado de la información es determinado únicamente por la forma en que es utilizada, no por las premisas iniciales.

Los lenguajes de programación compilados de la actualidad presentan un grado de tipificado alto. Por ejemplo:

- Cada variable en estos lenguajes debe ser declarada con un tipo de dato específico (entero, flotante booleano, etc.) y debe ser utilizado en la manera apropiada según el tipo de dato.

- Datos y código están segregados, lo que dificulta o imposibilita la creación de nuevo código al instante.

- Las variables deben ser alojadas dentro de estructuras u objetos con subestructuras y métodos o procedimientos que las manipulan, bien definidos. Un objeto de un cierto tipo no puede ser usado donde un objeto de un tipo distinto sea esperado.

Como vemos, el tipificado es una característica útil para medir el grado en que pueden realizarse interfaces entre los componentes, debido a que al no existir declaración de variables, las mismas pueden manipularse sin afectar otros componentes que esperan recibir cierto tipo de dato específico.

[29] Ver Capítulo 1
[30] Ver Capítulos 2 y 3
[31] OUSTERHOUT, John K. *Scripting: Higher-Level…*
[32] El término en inglés utilizado es *typing*, nosotros utilizaremos el termino de *tipificado* que conceptual mente se acerca a la idea del autor.

En contraste con los lenguajes compilados, los lenguajes interpretados como Perl, Python, Rexx, Tcl, Visual Basic y los *scripts* de UNIX presentan un bajo grado de tipificado. Estos lenguajes presentan un estilo muy diferente de programación, los lenguajes interpretados asumen que una colección de componentes útiles existen en otros lenguajes. Los lenguajes interpretados son utilizados para extender las características de componentes.

La siguiente figura compara una variedad de lenguajes en su nivel de programación y su grado de tipificado.

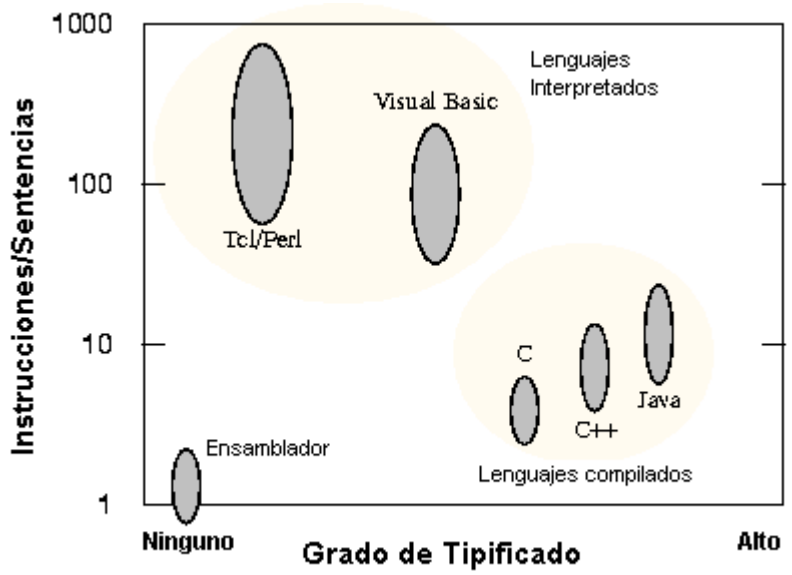


Fig. XIV Comparación de varios lenguajes de programación basados en su nivel (Un nivel alto son los lenguajes que ejecutan más instrucciones máquina por cada sentencia del lenguaje) y su grado de tipificado. Los lenguajes compilados como C tienden a utilizar fuertemente el tipificado y un nivel medio de instrucciones por sentencia. Los lenguajes interpretados como TCL tienden a utilizar un grado menor de tipificado y muy alto nivel de instrucciones por sentencia[33].

Para simplificar la tarea de conectar componentes, los lenguajes interpretados tienden a no utilizar la tipificación. La naturaleza de alta tipificación de los programas compilados demerita la reutilización.

Los lenguajes interpretados proveen de un rápido incremento de productividad en el desarrollo, eliminando los tiempos de compilación. Los interpretados también crean aplicaciones más flexibles permitiendo a los que los utilizan programar en tiempo de ejecución. La desventaja es que los lenguajes interpretados son menos eficientes que los compilados debido a su naturaleza. Afortunadamente, el rendimiento del lenguaje interpretado no es, en nuestro caso, el principal criterio de selección.

A cambio, los lenguajes interpretados permiten un desarrollo más rápido en comparación con los lenguajes compilados (como podemos observar en la siguiente tabla), lo que sí sería una

[33] OUSTERHOUT, John K. *Scripting: Higher-Level Programming...* Pag. 25

característica importante debido a que debemos considerar que posiblemente la aplicación sea mantenida por alumnos de áreas diferentes a la de computación.

Aplicación (contribuidor)	**Comparación**	**Tasa de Código***	**Tasa de esfuerzo****	**Comentarios**
Aplicación de base de datos (Ken Corey)	Versión en C++: 2 meses Versión en TCL: 1 día		60	La versión en C++ fue implementada primero; la versión en TCL tiene mayor funcionalidad
Sistema de pruebas e instalación de sistemas de computadoras (Andy Belsey)	Versión en C: 272,000 líneas, 120 meses. Versión en C FIS: 90,000 líneas, 60 meses. Versión en TCL/Perl: 7,700 líneas, 8 meses.	47	22	La versión en C se implementó primero; La versión en TCL/Perl sustituyó a ambas aplicaciones en C.
Librería de base de datos (Ken Corey)	Versión en C++: 2-3 meses Versión en TCL: 2 semanas		8-12	La versión en C++ se implementó primero.
Escáner para seguridad (Jim Graham)	Versión en C: 3,300 líneas Versión en TCL: 300 líneas	10		La versión en C se implementó primero; la versión en TCL tiene mayor funcionalidad
Despliegue de curvas de producción de petróleo. (Dan Schenck)	Versión en C: 3 meses Versión en TCL: 2 semanas		6	La versión en TCL se implementó primero
Despachador de consultas (Paul Healy)	Versión en C: 1,200 líneas, 4-8 semanas Versión en TCL: 500 líneas, 1 semana	2.5	4-8	La versión en C se implementó primero, no tiene comentarios; la versión en TCL tiene mayor funcionalidad y tiene comentarios
Herramienta de hoja de cálculo	Versión en C: 1,480 líneas Versión en TCL: 380 líneas	4		La versión en TCL se implementó primero
Simulador y GUI (Randy Wang)	Versión en Java: 3,400 líneas; 3-4 semanas. Versión en TCL: 1,600 líneas, menos de una semana	2		

* La tasa de código es la razón de líneas de código de las dos implementaciones
** La tasa de esfuerzo es la razón de los tiempos de desarrollo. En la mayoría de los casos las dos versiones fueron implementadas por programadores diferentes.

TABLA VI. Comparación de aplicaciones implementadas en ambos lenguajes (interpretado y compilado).

Sin embargo, después de esta discusión, debemos dejar en claro que los lenguajes interpretados no son el reemplazo de los lenguajes compilados, ambos atienden necesidades específicas distintas. Ousterhout propone realizar las siguientes preguntas:

- ¿La principal tarea de la aplicación es conectar componentes preexistentes?
- ¿La aplicación manipulará una variedad de cosas distintas?
- ¿La aplicación incluye una GUI?
- ¿La aplicación va a realizar un manejo alto de cadenas?
- ¿Las funciones de la aplicación evolucionan rápidamente en el tiempo?
- ¿La aplicación necesita ser extensible?

Respuestas afirmativas a estas preguntas sugieren que un lenguaje interpretado trabajará de manera óptima.

Los lenguajes interpretados han existido durante mucho tiempo, pero recientemente varios factores combinados han provocado que su importancia se incremente. Por ejemplo:

- *Las Interfaces Gráficas de usuario*. Las GUI's son fundamentalmente aplicaciones que requieren utilizar varios componentes e integrarlos. Su objetivo no es crear nueva funcionalidad sino realizar conexiones entre los controles gráficos y la funcionalidad interna de la aplicación.
- *Internet*. El lenguaje ideal para la mayoría de la programación en Internet es uno que haga posible que todos los componentes conectados trabajen juntos; esto es, un lenguaje interpretado.
- *Mejor tecnología para intérpretes*. Otra razón para la creciente popularidad de lenguajes interpretados son las mejoras en la tecnología de éstos.
- *Programadores casuales*. Los lenguajes interpretados son fáciles de aprender debido a que tienen una sintaxis más simple que la de los compilados. Y esto hace que más utilizados por programadores casuales.

En conclusión, debido a las características de escalabilidad y adaptabilidad, además de su fácil aprendizaje y manejo, los lenguajes interpretados parecen ser la mejor opción para implementar el diseño de la nuestra aplicación.

Comparación de distintos lenguajes interpretados

A continuación presentaremos un cuadro sinóptico, donde comparamos a varios de los lenguajes interpretados más populares, tomando en cuenta los criterios de diversos autores.

Lenguaje	¿Interpretado?	¿Orientado a Objetos?	Comentarios Sobre reutilización	Filosofía	¿Existen Componentes relativos a Astronomía?	¿Es soportado por .NET?
Java	Java no es un lenguaje interpretado *per se*. Es compilado en un código binario intermedio, que después será interpretado por la Máquina Virtual de Java (*JVM*, por sus siglas en inglés). Esto le da gran portabilidad debido a que es totalmente independiente de la plataforma. Sin embargo, no cuenta con las características de bajo tipificado descritas anteriormente.	Sí	Java ofrece un alto grado de reutilización. Sin embargo, no es posible hacer herencias múltiples.	Java es un programa de bajo nivel de propósito general, independiente de la plataforma. Sus características principales son: ? Portabilidad. ? Manejo automático de memoria *(Garbage Collector)* ? Manejo de excepciones.	Sí	Sí
JavaScript	Si	Sí. Aunque algunos programadores piensan que JavaScript no explota todas las características de la Orientación a objetos.	Debido al punto anterior, se piensa que su nivel de reutilización no es óptimo debido a que no explota totalmente los conceptos de clases y herencia.	JavaScript es un lenguaje de alto nivel de propósito general. Sus principales características son: ? Facilita código embebido en aplicaciones WEB	No los que necesitamos.	Sí
Python	Sí. Aunque existe la opción de generar un código intermedio sin afectar las características de bajo tipificado.	Sí. Algunos programadores lo comparan con C++ debido a sus características de Orientación a Objetos.	Ofrece un alto grado de reutilización debido a sus características de orientación a objetos.	Python es un lenguaje de alto nivel de propósito general. Sin embargo ofrece la facilidad de acercarse a una programación a bajo nivel mediante C o	Sí	Sí

				C++. Sus principales características son: ? Portabilidad (existen varias versiones de Python para distintas plataformas) ? Extensibilidad a bajo nivel mediante C y C++ ? Manejo automático de memoria. ? Manejo de excepciones ? Manejo de arreglos simple y poderoso. ? Manejo de *Diccionarios* que son arreglos que pueden ser ordenados y administrados mediante una *llave*.		
Perl	Sí	No	No	Perl es uno de los lenguajes interpretados más antiguos. Es un lenguaje de alto nivel pero más enfocado al procesamiento de texto y la creación de reportes. Algunos piensan que su código es complicado y confuso; sin embargo, tiene un nicho bien definido dentro de los	No	No

				programadores.		
TCL	Sí. Es excelente para agregar funcionalidad a aplicaciones ya existentes. Sin embargo, a madurado más como lenguaje de ejecución local.	Sí	TCL ofrece la reutilización, sobre todo, de aplicaciones ya existentes.	TCL es un lenguaje de alto nivel, que ofrece una programación a bajo nivel mediante C. Como ya hemos dicho, ofrece la ventaja de extender la funcionalidad de aplicaciones ya existentes. Sin embargo, su desventaja es que es muy débil en el manejo de estructuras de datos.	Sí. Aunque no tan especializados.	No

TABLA VII. Comparación de distintos lenguajes interpretados.

Como podemos apreciar, los dos lenguajes que ofrecen mayor número de ventajas son Java y Python.

La elección fue Python básicamente por sus características de bajo tipificado que esperamos que contribuyan a un fácil mantenimiento.

Breve Descripción de Python.

Python fue originalmente desarrollado y mejorado por Guido van Rossum, quien lo llamó así debido al *Circo volador de Monty Python*. Guido liberó Python a través del Internet y la comunidad de usuarios se ha ido incrementando considerablemente desde entonces, lo que a motivado a una mejora continua del lenguaje.

Python retoma las mejores características de los lenguajes OO más populares como SmallTalk, FP y los guiones de UNIX. Desciende de la familia de los lenguajes Modula pero utiliza el tipificado bajo de Lisp. Parece no haber nada nuevo en Python, pero la verdadera novedad es que todas estas características se ofrezcan en un sólo lenguaje, simple, bien diseñado, portable y escrito en C.

Sus derechos permiten un uso arbitrario del lenguaje y su código fuente, incluso para aplicaciones comerciales.

Además Python ofrece soporte a través de su sitio oficial[34], sitios de aficionados al lenguaje, grupos de discusión y diversas publicaciones.

3.2.2 Mysql, Manejador de Base de Datos seleccionado

MySQL es un sistema manejador de bases de datos relacionales que utiliza el lenguaje estructurado de consultas, SQL (Structured Query Language), es el encargado de administrar, crear y actualizar una base de datos relacional. MySQL se caracteriza por ser un manejador de base de datos muy veloz, flexible, multihilos, multiusuario y robusto. La primera versión fue liberada en Enero de 1998, y es marca registrada de MySQL AB, una compañía comercial creada por los desarrolladores de MySQL.

MySQL cuenta con una licencia dual, se puede escoger entre utilizarlo como un producto de código abierto o libre (*open source o free source*), o se puede comprar la licencia comercial de MySQL AB. Cuando hablamos de un software de código abierto, nos referimos a cualquier programa donde su código fuente esta disponible para su uso, modificación, ejecución y distribución. Es decir, un software de código abierto o mejor dicho código libre, es aquél que es desarrollado usualmente como una colaboración pública y es libremente distribuido. Por lo tanto cualquier persona puede obtener una versión de MySQL, ver y modificar su código fuente par adaptarlo a sus necesidades, respetando la licencia pública de GNU (es acrónimo recursivo para *GNU's Not Unix).* Si por el contrario nuestra necesidad es utilizar MySQL como parte de alguna aplicación comercial, no podemos utilizar la licencia de GNU, sino la comercial.

MySQL es un manejador de bases de datos, muy veloz, flexible y fácil de usar, fue desarrollado en un principio con el objetivo de manejar grandes bases de datos con mayor velocidad. Consta de una arquitectura Cliente/Servidor que utiliza un servidor SQL multihilos que es soportado por diferentes plataformas o sistemas operativos, diferentes programas clientes y librerías, herramientas administrativas y una amplia gama de interfaces con programas (APIs), por lo que el usuario de acuerdo a sus necesidades puede decidir con qué aplicación le conviene ligarlo para hacerlo más eficiente.

Las principales interfaces con programas a través de API'S(application program interface), son con C y C++, Eiffel, Java, Perl, Python y Tcl. Actualmente MySQL puede ejecutarse en Linux, Unix, Mac OS y plataformas con Windows. También cuenta con OLE DB y OBDC para la conexión con aplicaciones de ambiente tipo Microsoft y un proveedor nativo para las aplicaciones .NET que no necesita el OLE BD.

Las principales características de MySQL son:

- Escrito en C y C++
- Trabaja en diferentes plataformas.
- Utiliza procedimientos de GNU que le dan portabilidad (Automake, Auto config, Libtool)

[34] http://www.python.org

- APIs para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl disponibles.
- Mutlihilos, utilizando los hilos del sistema (kernel), es decir, puede fácilmente manejar varios procesadores.
- Maneja almacenamiento transaccional y no transaccional.
- Rápido manejo de memoria.
- Puede correr para una aplicación en red o monousuario.
- Encriptación de passwords.
- Gran capacidad de almacenamiento, Ej. , 50 millones de registros.
- Hasta 32 índices por tabla son permitidos.
- Conectividad usando TCP/IP sockets, o en ambiente Windows utilizando pipes.
- Interfaces ODBC y JDBC
- Soporte en varios idiomas.

MySQL es más comúnmente usada para aplicaciones tipo web y aplicaciones contenidas o de manejo de componentes (objetos). Esta ultima característica, la hace buena para la el desarrollo de la aplicación del presente trabajo de tesis.

Debido a las características que se modelarán mas adelante, era necesario contar con un servidor de Base de Datos que soportara un gran manejo de registros, que tuviera velocidad en las consultas, y que tuviera una interfaz con el lenguaje seleccionado, en este caso particular, Python y Tcl. Todo lo anterior lo cumple MySQL, es compatible con la filosofía del código abierto y sus características, cono lo es el sistema operativo seleccionado, Linux, y cuenta con los estándares y licencia de GNU. Así contamos con una plataforma o infraestructura homogénea, para el desarrollo de la aplicación.

Podemos verlo con el enfoque del bajo costo pues las licencias, los ejecutables, el código, actualizaciones, y nuevo s módulos, estarán disponibles por ser código abierto y no representarán inversión para la aplicación.

4 Conceptos de astronomía

El problema que pretende resolver la presente tesis es el de controlar el movimiento de un telescopio motorizado para ubicar la posición de un objeto en la bóveda celeste por medio de un programa de computadora que sea lo suficientemente flexible como para agregar fácilmente nuevas funcionalidades. Si bien es cierto que el problema de controlar un telescopio a través de un programa de computadora ya ha sido resuelto y se ha ido perfeccionando, los autores pretenden dar una nueva solución utilizando el paradigma de programación orientada a objetos para facilitar el mantenimiento, utilizando plataformas robustas y de distribución gratuita.

Para poder iniciar con el desarrollo del presente trabajo, es necesario entender cómo se describe la posición de un objeto sobre la bóveda celeste. La posición de dicho objeto depende del lugar de la tierra donde este ubicado el observador, para lo cual describiremos brevemente las principales líneas de la Tierra; para después explicar los principales sistemas de coordenadas que existen.

4.1 La esfera celeste

Llamamos *esfera celeste* a una gran esfera imaginaria que contiene dentro de su superficie interior a todas las estrellas y que es concéntrica a la tierra, la cual imaginamos como un pequeñísimo globo.

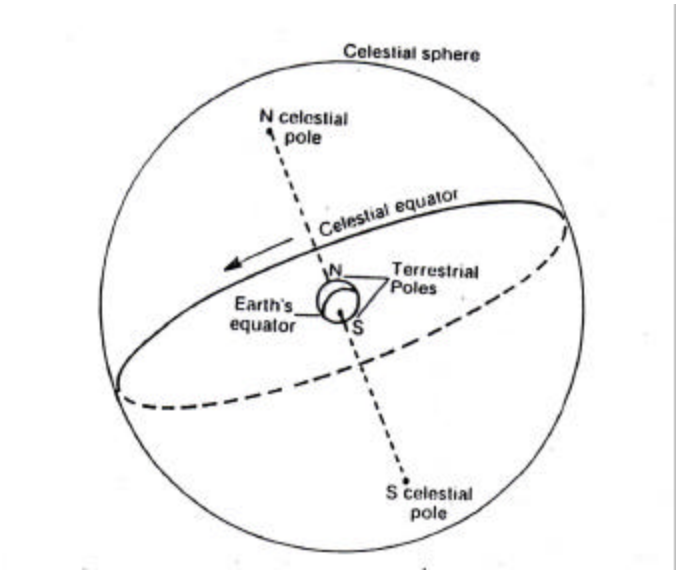


Fig. XV Esfera Celeste

4.1.1 Líneas, planos y círculos de la esfera celeste

La primera línea que se puede apreciar es la *vertical*, señalada por una *plomada*. Prolongando idealmente esta línea hasta que toque la bóveda celeste, marca en ella los puntos llamados *cenit* hacia arriba del observador y *nadir*, hacia abajo.

La segunda línea importante es el *eje polar*, el cual es una extensión del eje polar terrestre hacia la esfera celeste.

Una tercera línea es la *meridiana*, que es la intersección de los planos del meridiano y del horizonte que adelante se definen.

Llamamos plano *meridiano* al plano vertical que contiene a la línea de los polos. Por cada lugar de la tierra sólo pasa un plano meridiano.

El plano del *horizonte* es aquel que divide a la esfera celeste en dos partes iguales y es perpendicular a la vertical del lugar.

El plano del *ecuador* divide a la esfera celeste en dos partes iguales y es perpendicular al eje polar terrestre.

Se llaman *círculos horarios* o meridianos celestes los círculos secundarios que pasan por los polos, y *paralelos celestes* a los círculos menores paralelos al ecuador.

El plano de la *eclíptica* es el que contiene la órbita de la Tierra en su movimiento anual alrededor del Sol.

4.1.2 Movimiento de las estrellas en la esfera celeste

Sea un observador O, en la superficie de la Tierra, entre el ecuador y el polo; OZ su vertical y H'H el plano de su horizonte visual. Para él la esfera celeste gira, aparentemente, alrededor de PP', eje polar terrestre, de Este a Oeste.

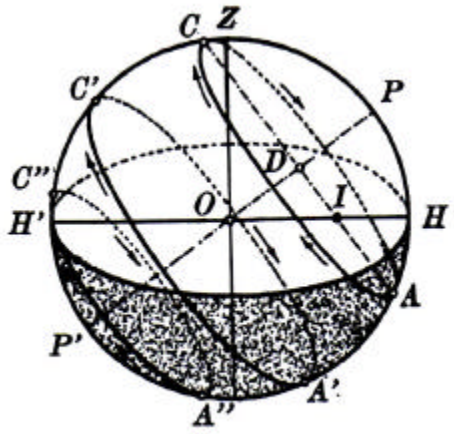


Fig. XVI Movimiento aparente de las estrellas

Cada estrella describe, debido al movimiento de rotación de la Tierra, una circunferencia cuyo plano, inclinado con respecto al horizonte es perpendicular al eje polar terrestre y paralelo, por consiguiente, al plano del ecuador celeste.

El movimiento aparente de las estrellas es uniforme, pues cada una describe arcos iguales en tiempos iguales, e isocrono por que cada estrella tarda invariablemente 24 horas siderales en dar la vuelta completa a la esfera celeste.

El lector puede inferir fácilmente que existe una relación entre la latitud del lugar y la inclinación de los paralelos con respecto al plano del horizonte.

Para un observador situado en el ecuador terrestre, el ecuador celeste y los paralelos descritos por las estrellas son perpendiculares al horizonte; por tanto dicho ecuador es un plano vertical, y los paralelos descritos por las estrellas son normales al horizonte.

En cambio, para un observador que estuviera situado exactamente en el polo norte de la Tierra, tendría el polo norte celeste en su cenit. Allí el horizonte se confunde con el ecuador celeste y los paralelos descritos por las estrellas resultan paralelos al horizonte.

Cuando el observador esta en el polo, es decir, a una latitud de 90°, los paralelos forman con el horizonte un ángulo de 0°; en cambio, cuando el observador está en el ecuador, donde su altitud es igual a 0°, los paralelos forman un ángulo de 90° con el plano horizontal; Si se encuentra a una latitud intermedia, el ángulo del plano de los paralelos con el horizonte se acerca tanto más a 90° cuanto menor sea la latitud del lugar de observación.

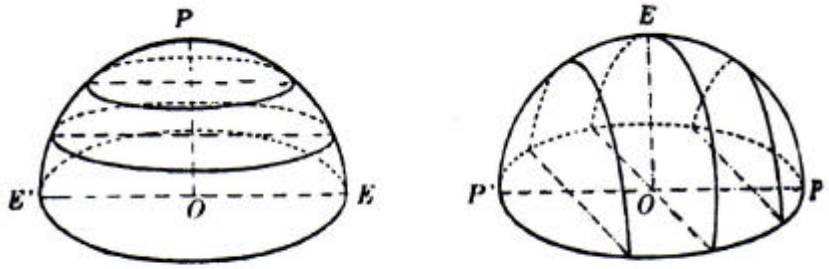


Fig. XVII Observador en el polo (der.) y Observador en el ecuador (izq.)

En cada caso, la inclinación de los paralelos con respecto al horizonte es complementaria de la latitud del lugar que se considera.

4.2 Tiempo

Al considerar los movimientos de los cuerpos celestes con referencia al observador, se adopta como plano vertical de referencia el meridiano del lugar. Se establece que es *mediodía* cuando el Sol pasa por el meridiano superior y *medianoche* cuando lo hace por el meridiano inferior. Así, en el uso diario, el día se considera como el intervalo entre dos pasajes o *tránsitos* sucesivos del Sol por el meridiano inferior del lugar, es decir, por el plano vertical que contiene el nadir y los puntos sur y norte del horizonte.

Para medir el tiempo se emplean el *Sol verdadero*, *Sol medio* y el *equinoccio vernal* que definen, respectivamente, el *tiempo solar verdadero, tiempo solar medio*, y el *tiempo sidéreo*.

4.2.1 Tiempo solar verdadero

El tiempo solar verdadero resulta del intervalo de tiempo transcurrido entre dos pasajes sucesivos del Sol por el meridiano del lugar, no sirve como unidad de tiempo, pues su duración varía diariamente. Por dos razones no es uniforme: por que el Sol se proyecta de manera diferente sobre la Tierra a lo largo del año; y por que además la trayectoria de la

Tierra alrededor del Sol es una elipse, lo que resulta en que el movimiento tenga una velocidad variable.

4.2.2 Tiempo solar medio

Para contar con un tiempo uniforme, se introduce un Sol ficticio, llamado Sol medio, que se desplaza sobre el ecuador a una velocidad constante a lo largo del año. El Sol medio define de esta manera el tiempo solar medio, cuya duración es constante a lo largo del año.

4.2.3 Tiempo sidéreo

El tiempo sidéreo o sideral se define como el *ángulo horario*[35] del equinoccio vernal (o equinoccio de primavera) ♈, es decir:

$$T_s = t_{♈}$$

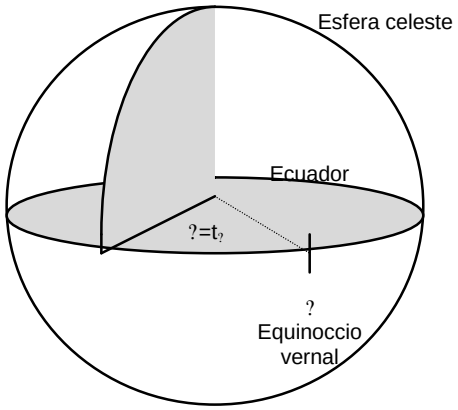


Fig. XVIII Tiempo sidéreo

El día sidéreo se define como el intervalo de tiempo que transcurre entre dos tránsitos sucesivos del punto vernal por el meridiano del lugar. Se mide en horas, minutos y segundos sidéreos y se comienza a contar cuando el punto vernal pasa por el meridiano superior.

Un día solar medio, como se vio anteriormente, resulta del periodo de rotación de la Tierra alrededor del Sol. La duración de uno y otro son distintas.

Un día solar es algo más largo que un día sidéreo, exactamente por 3^m56^s555

4.2.4 Husos horarios

De las definiciones anteriores podemos observar que cada uno de los tiempos sería diferente al desplazarse el observador de longitud, es decir, a lo largo de los meridianos. Esto ocurría

[35] El *ángulo horario* se define como el ángulo entre el plano del meridiano del lugar y el círculo horario que contiene al astro, en este caso, contiene al punto vernal (equinoccio de primavera).

en la práctica hasta el siglo pasado, cuando se estableció el sistema de husos horarios, de acuerdo con el cual se divide a la Tierra en 24 husos de un ancho de 15º. Dentro de cada huso la hora es uniforme y corresponde a la de su meridiano central. Por el centro del primer huso pasa el meridiano de Greenwich, que corresponde al origen 0º de las longitudes geográficas.

4.3 Coordenadas celestes

4.3.1 Coordenadas horizontales

Puede localizarse un astro en la esfera celeste, refiriendo su posición al plano del horizonte y al meridiano del observador, por medio de dos coordenadas, llamadas coordenadas horizontales.

Sea A un astro y ZAB un plano que pasa por él, plano que, por contener la vertical del lugar, es un plano vertical. Este plano corta el del horizonte según la recta OB.

Las coordenadas horizontales de A son: BOA, que es el ángulo formado por la visual del astro y el plano del horizonte (altitud), y el ángulo diedro que forma el plano vertical del astro con el plano del meridiano del lugar, medido por el ángulo plano NOB, que se llama *acimut* del astro.

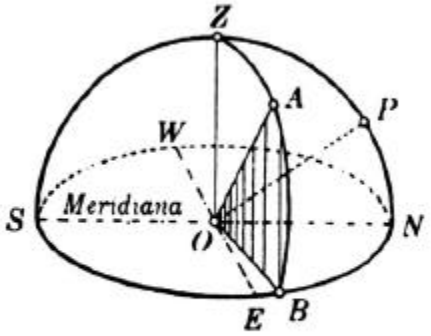


Fig. XIX Coordenadas horizontales

Las alturas se cuentan partiendo del plano del horizonte hacia el cenit, y se cuenta de 0° a 90°; los acimutes se miden normalmente de Norte a Sur teniendo valores entre 0° y 360°.

4.3.2 Coordenadas horarias

Otro sistema para localizar un objeto en la bóveda celeste es utilizar como plano fundamental al del ecuador celeste, que define como eje fundamental el eje polar que pasa por los polos celestes norte y sur, ya que como dijimos anteriormente la esfera celeste gira aparentemente alrededor del eje polar terrestre describiendo arcos de igual magnitud en tiempos iguales.

De esta forma las coordenadas horarias de un lugar son: el ángulo horario (H) y la declinación (?).

El ángulo horario H de un astro E es por definición, el arco ME' del ecuador celeste comprendido entre el meridiano superior del lugar y el círculo horario que pasa por E. Se cuenta sobre el ecuador a partir de del punto de intersección M entre el meridiano superior y el ecuador, de 0h a 24h en sentido retrógrado, es decir, O-N-E-S.

La declinación ? de un astro E es por definición, el arco EE' del círculo horario que pasa por el astro E, comprendido entre E y el ecuador. Se cuenta a partir del ecuador de 0° a 90°, positivamente hacia el polo N y negativamente hacia el polo S.

La declinación es sustituida en ocasiones por la distancia polar *p* que es su complemento algebraico, contada a partir del polo N de 0° a 180°. Su uso en los cálculos tiene la ventaja de evitar posibles errores o equivocaciones en los signos.

Como consecuencia del movimiento diurno respecto al lugar de observación, el astro recorre su paralelo celeste con movimiento uniforme, despreciando las irregularidades debidas a la rotación de la tierra y de las variaciones del polo, las cuales son muy pequeñas en un intervalo de observación de unas cuantas horas; se tiene que: 1. La declinación ? es constante y 2. El ángulo horario H varía proporcionalmente con el tiempo.

Las coordenadas horarias resultan las más adecuadas para estudiar el movimiento diurno de los astros en un intervalo de tiempo corto. Por ésta razón, los grandes telescopios se montan sobre ejes paralelos al eje polar y al plano del ecuador, a lo que se le llama *montura ecuatorial* o *paraláctica*.

Los sistemas de coordenadas que se han estudiado hasta el momento, coordenadas horizontales y coordenadas horarias, dependen del lugar del observador sobre la superficie de la Tierra, llamándose por éste motivo *coordenadas locales*. El movimiento diurno va cambiando la posición del horizonte astronómico local sobre la esfera celeste y por tanto, las coordenadas horizontales acimut, A y altura, h, no sólo son diferentes para dos observadores en distintos lugares de la Tierra, sino que además, van cambiando con el tiempo para un mismo observador. En las coordenadas horarias, el movimiento diurno afecta solamente al ángulo horario H, mientras que la declinación ? permanece constante.

Interesa entonces definir un sistema de coordenadas que sea completamente independiente al lugar de observación.

4.3.3 Coordenadas ecuatoriales

Sea PZE el corte de la esfera celeste según uno de sus meridianos; A un astro y E'E el plano del ecuador. Si se trazan dos planos perpendiculares al ecuador (plano fundamental) y que pasen uno por el astro y el otro por un punto de dicho ecuador llamado *punto gamma* (♈) o *punto vernal*.

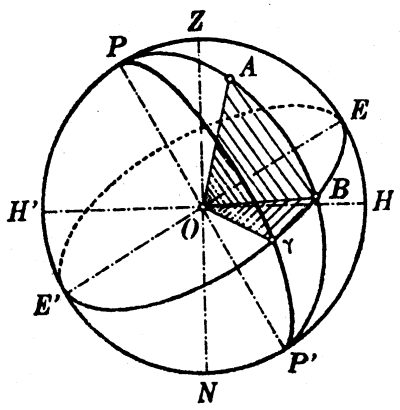


Fig. XX Coordenadas ecuatoriales

La declinación ♈ del astro A, como ya se dijo en la sección anterior, es el ángulo BOA que forma la visual del astro con el plano del ecuador.

La ascensión recta del astro A, que es la otra coordenada ecuatorial, es el ángulo diedro que forma el semicírculo horario PAB del astro con el semicírculo horario del punto vernal. La ascensión recta del astro A está medida por el ángulo ♈OB.

Las ascensiones rectas tienen como origen el punto ♈, y se cuentan de 0° a 360°, o de 0h a 24h, en sentido directo, es decir, hacia el oriente.

Estas coordenadas son invariables en el curso de unos cuantos días, pues al girar la esfera celeste, el punto A describe un paralelo al ecuador; entonces el ángulo BOA permanece constante y, por lo tanto, la declinación no varía. La ascensión recta tampoco varía, puesto que la estrella y el punto ♈ están en dicha esfera y giran con ella. Sin embargo como se verá mas adelante, año con año varían esas coordenadas, aunque ligeramente, porque se desplaza el punto ♈ y con él el plano del ecuador.

4.3.4 Relación entre hora sidérea, ángulo horario y ascensión recta

Como se dijo en la sección 2.2.3, se define como hora sidérea o tiempo sidéreo (T) en un sitio cualquiera, al ángulo horario del punto vernal ♈ en ese lugar. En la figura anterior se observa que éste esta medido por el arco ♈E. Por la misma se ve que:

$$♈E = ♈B + BE \qquad (1)$$

De la igualdad (1) se infiere que cuando se conoce la ascensión recta de una estrella y se mide su ángulo horario, se puede determinar la hora sideral.

Si se conviene en escribir:

♈E = Ts, tiempo sidéreo,

♈B = a, ascensión recta,

BE = H, ángulo horario,

Se tiene por (1):

$$Ts = a + H.$$

Esta relación es fundamental en la astronomía de posición, que nos dice que cada en instante la hora sidérea local es igual a la suma del ángulo horario y la ascensión recta del astro.

Nótese que si Ts > a, da un valor negativo de h, siendo que se ha convenido contar H siempre positivo de 0h a 24h en sentido retrógrado. *Ts* puede ser menor que *a* obteniéndose un valor *–H*, bien cuando el astro se encuentra al este del meridiano del lugar o bien cuando el punto ♈ está entre el meridiano local y el círculo horario del astro, siendo entonces Ts < H.

Para obtener H positivo según las convenciones adoptadas, añadiremos a *–H* una circunferencia completa, o sea 24h. Se tiene, en este caso, para Ts < a, que:

$$H = (-H) + 24h = Ts – a + 24h$$

O bien:

$$Ts = H + a – 2$$

4.4 Telescopios y Monturas

Hasta ahora sólo nos hemos dedicado al estudio de la cosmografía, es decir, al estudio de la bóveda celeste y los cuerpos que en ella se encuentran; sin embargo, es necesario hacer un breve resumen de los tipos de telescopios que existen así como de las distintas monturas a las que pueden acoplarse para poder iniciar el diseño de la solución.

Al entender mejor el comportamiento de los telescopios y sus monturas, podremos implementar un mejor control de acuerdo con las características de cada uno de éstos.

Los telescopios astronómicos pueden ser de varios tipos, según los elementos ópticos que utilicen sean reflectores o refractores.

Los primeros telescopios fueron refractores, pero el gran inconveniente de éstos fue su gran aberración cromática. Esto se trató de solucionar aumentando la distancia entre los lentes, pero el telescopio se volvía inestable e incómodo. Éstas fueron las motivaciones que llevaron a la invención del telescopio reflector. Desafortunadamente este tipo de telescopio tenía sus propios problemas. En esta sección se describirán los telescopios refractores, así como los factores que afectan la calidad de los mismos y las monturas a las que se pueden acoplar.

Cómo funciona un telescopio

Un telescopio es esencialmente un par de lentes, una llamada objetivo porque es la más cercana al objeto, y la otra llamada ocular por que es la más cercana al ojo. El objetivo es una lente convergente que forma una imagen I del objeto. Es fácil comprender que esta imagen es tanto mayor cuanto más larga sea su distancia focal, es decir, cuanto menos convergente sea. Esta imagen I se observa con la ayuda de una pequeña lente.

La imagen I, al ser observada, producirá a su vez una imagen en la retina del ojo, que será tanto más grande cuanto más cerca esté esta imagen del globo ocular. Como el ojo no puede enfocar los objetos que están muy cerca de él, es necesaria la ayuda de otro lente llamada ocular, para realizar este enfoque. Si la imagen está atrás del ojo se usa una lente divergente o negativa, pero si está adelante se usa una convergente o positiva.

4.4.1 Aberraciones de los telescopios

La calidad de la imagen de un telescopio está limitada por muchos factores, unos asociados al telescopio en mismo y que por tal motivo está fuera del alcance de esta tesis; otros al medio en el que se propaga la luz, es decir, a la atmósfera, y otros que dependen de la naturaleza de la luz.

Las aberraciones que dependen del telescopio en sí, es decir, de la construcción del mismo como sistema óptico, son las que siguen:

a) Aberración de esfericidad
b) Aberración de coma
c) Astigmatismo
d) Curvatura de campo
e) Distorsión
f) Aberración cromática axial
g) Aberración cromática lateral

Como ya dijimos, estas aberraciones por ser debidas al telescopio mismo no serán tratadas en este trabajo; sólo diremos que las primeras cinco se pueden manifestar, cualquiera que sea el color del objeto. Las últimas dos, en cambio, sólo pueden aparecer si el objeto es

blanco, es decir, si su luz está formada por la mezcla de muchos colores. Además varias de estas aberraciones pueden evitarse si la construcción de las lentes es cuidadosa.

Las aberraciones se pueden corregir, pero la difracción no es posible eliminarla jamás; sólo se puede reducir su magnitud aumentando el tamaño de la pupila, es decir, el diámetro de los lentes. Afortunadamente, el efecto de la difracción es en general muy pequeño comparado con el que casi siempre introducen aún las pequeñas aberraciones.

Además de las aberraciones y de la difracción, otro factor importante que altera la calidad de la imagen en un telescopio es la turbulencia atmosférica.

Como sabemos, la atmósfera está en continuo movimiento debido a las diferencias locales de temperatura. Como las variaciones en temperatura ocasionan también variaciones en el índice de refracción, la imagen de una estrella se desenfocará y moverá continuamente.

4.4.2 Los telescopios refractores

Los primeros telescopios, los cuales aparecieron en el siglo XVII, fueron los del tipo refractor, y al día de hoy, mucha gente sigue pensando que un buen telescopio refractor es mejor que cualquier otra clase de telescopios debido a que proporciona imágenes nítidas y requiere de muy poco mantenimiento.

El telescopio refractor usa un objetivo, el cual consiste en un par de lentes montados en una celda al inicio del tubo, que concentra la luz y el foco de la imagen.

El punto focal del objetivo, en otras palabras, es el lugar donde se forma la imagen, se encuentra al final del tubo. Aquí es donde el ocular se acopla dentro del tubo de recolección, el cual permite un movimiento lento hacia adentro y hacia afuera mediante una perilla para enfocar la imagen. La distancia focal del objetivo, como se podría esperar, es aproximadamente el largo del tubo del telescopio.

El esquema de un telescopio refractor se muestra a continuación:

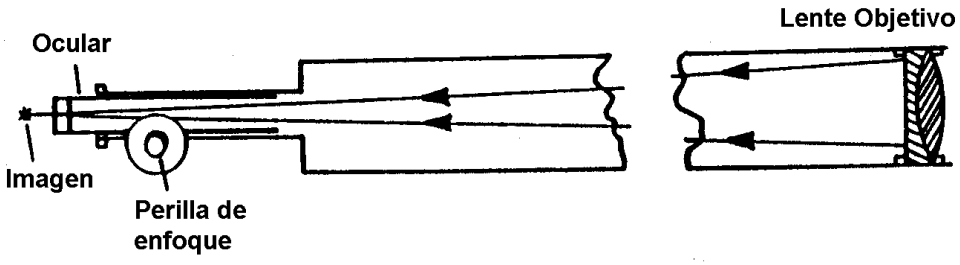


Fig. XXI Esquema de telescopio refractor

4.4.3 Ventajas y desventajas del telescopio refractor

Tal vez la principal desventaja del telescopio refractor no sea, como pudiéramos pensar, sus deficiencias en el sistema óptico, sino su tamaño.

Tres son las razones en cuanto al sistema óptico se refiere que sustentarían esta aseveración:

1. Los errores ópticos en la superficie de un lente tienen menor efecto en la imagen final que los mismos errores en la superficie de un espejo.

2. Las superficies reflejantes de los espejos, tienden a diseminar más luz alrededor de la imagen que las superficies de los lentes.

3. Un lente es menos sensitivo a los errores de ajustamiento que un espejo.

Como sea, la principal desventaja de un telescopio refractor son sus dimensiones. Un telescopio f/15 con un lente de 100 mm puede rebasar el hombro de un adulto. Además debido a que el ocular esta en la parte final del tubo del telescopio, debe estar colocado a un nivel considerable arriba del piso para que el ocular se encuentre a cierta altura para una observación cómoda. Además, para ver objetos en una altura mayor de 60° sobre el horizonte, un accesorio conocido como *prisma cenital*, el cual rota el ocular a un ángulo recto con respecto al tubo recolector, resulta prácticamente esencial. Un prisma cenital puede llegar a ser una gran molestia, debido a que invierte la imagen de arriba a abajo.

Pero un tubo muy largo no es el único problema, sino también lo difícil que es montarlo establemente. Esta es una consideración importante ya que una buena montura es tan importante como la calidad en la óptica.

4.4.4 Telescopios reflectores

Estos telescopios utilizan un espejo cóncavo para recolectar la luz. De hecho, los espejos realizan mucho mejor este trabajo ya que no dispersan la luz en colores, sino que son perfectamente acromáticos, aunque requieren de ajustes ocasionales.

Los espejos de los telescopios reflectores están hechos ordinariamente de vidrios de baja expansión, como el Pyrex, o de silicio puro, los cuales tienen prácticamente cero expansión con los cambios de temperatura. Esto resulta importante ya que la definición puede ser arruinada por una contracción o una expansión en el vidrio.

Además, debido a que un espejo es perfectamente acromático, no es necesario hacer telescopios reflectores con las largas relaciones focales de los refractores. Muchos reflectores trabajan en un rango de f/5 a f/6. En un f/5 se puede tener tres veces la apertura (consecuentemente, nueve veces el poder de recolección) de un telescopio refractor, para el mismo largo de tubo. Esto resulta de enorme beneficio cuando la montura del telescopio es considerada.

4.4.5 Telescopios newtonianos

Todos los telescopios reflectores utilizan un espejo cóncavo situado al fondo de un tubo abierto para reflejar y enfocar la luz al inicio del tubo. En la forma común newtoniana un espejo pequeño, llamado *diagonal*, intercepta la luz reflejada y la refleja de nuevo a un lado del tubo donde se sitúa el ocular. Los telescopios newtonianos son un poco confusos para principiantes, además de que los espejos expuestos acumulan polvo y suciedad y deben ser ajustados ocasionalmente para mantenerlos alineados y colimados. A pesar de todo ofrecen la ventaja de ser baratos.

No menos importante es la característica de que el ocular se encuentra cerca del inicio del tubo, lo que permite una observación cómoda no importando la latitud del objeto observado.

4.4.6 Telescopios del tipo Cassegrain

El telescopio Cassegrain está formado por dos espejos, uno paraboidal y el otro hiperboloidal. Éste último sustituye al diagonal que estudiamos en la configuración newtoniana.

El espejo *secundario* refleja el rayo de luz convergente del espejo primario que se encuentra al final del tubo a un pequeño orificio al centro de la superficie del primario, o es reflejada al estilo newtoniano por un espejo plano a un lado del tubo, generalmente cerca del inicio de éste.

El espejo secundario incrementa considerablemente la distancia focal del primario sin ser necesario el correspondiente aumento de tamaño en la longitud del tubo. La distancia focal efectiva del sistema es el de una lente delgada equivalente, con la misma abertura de entrada, que produzca un haz refractado convergente, con el mismo ángulo que el producido por el sistema de dos elementos. Así pues, la distancia focal efectiva resulta mucho mayor a la separación entre los espejos, por lo que el telescopio es muy compacto. Típicamente el telescopio Cassegrain tiene una distancia focal efectiva tres o cuatro veces mayor que la longitud del telescopio.

En conclusión las ventajas de este tipo de telescopios es su gran poder de trabajo y su tamaño compacto.

4.4.7 Telescopios catadióptricos

Los telescopios catadióptricos son aquellos cuyos sistemas tienen como elementos ópticos tanto lentes como espejos. Hay varios ejemplos de telescopios de este tipo, pero los más importantes son los de Schmidt, Schmidt - Cassegrain, Maksutov y Maksutov – Cassegrain.

El sistema óptico Schmidt, más que un telescopio, es en realidad una cámara fotográfica cuya relación focal es muy corta (f/4 o menor). Esta relación focal corta le da un gran campo donde es necesario que tenga corregidas sus aberraciones.

La cámara Schmidt está formada por un espejo cóncavo de forma esférica, que tiene una placa esférica muy delgada en su centro de curvatura. El principio de funcionamiento se

puede explicar por medio de la figura XXII. Si se coloca un diafragma circular o pupila en el centro de curvatura de un espejo esférico cóncavo, la imagen formada por un haz de rayos paralelos emitidos por un objeto puntual al infinito será idéntica para cualquier dirección. La razón de esto es que el sistema completo tiene simetría alrededor del centro de curvatura.

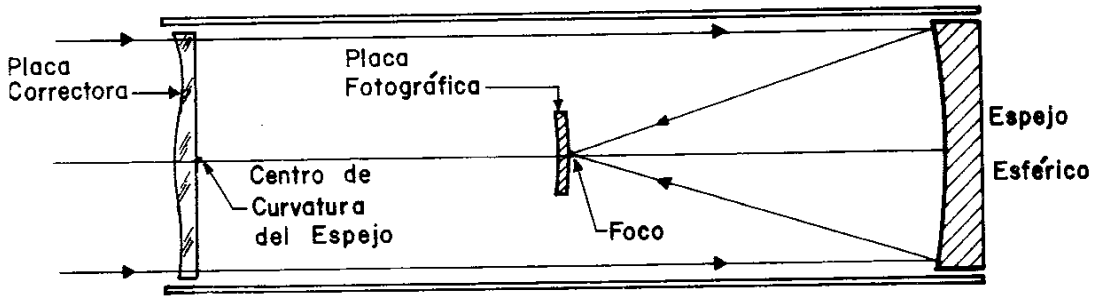


Fig. XXII Esquema de telescopio catadióptrico, Cámara Schmidt.

La cámara Schmidt produce imágenes de sorprendente calidad aunque tiene el problema muy importante de que la placa correctora es muy difícil de construir.

Una alternativa para el sistema Schmidt que usa el mismo principio de simetría alrededor del centro de curvatura fue propuesto por D. D. Maksutov. Como se muestra en la figura XXII se coloca una lente con forma de menisco o cáscara esférica cerca del foco, con sus caras concéntricas con el espejo primario. Este sistema tiene varias ventajas pero también desventajas con respecto al de Schmidt. El sistema es más compacto por tener la placa correctora cerca del foco y no de la curvatura, la simetría alrededor del centro de curvatura es más completa por lo que se pueden lograr campos más amplios, las superficies de la placa correctora son más fáciles de tallar y probar por ser esféricas. En cambio, la corrección de la aberración de esfericidad no es tan buena como en el sistema Schmidt. Además la curvatura tan pronunciada de la superficie hace más caro el bloque de vidrio que se necesita para construirlas.

Los sistemas concéntricos de Schmidt y Maksutov han tenido tanto éxito que se han utilizado para mejorar la calidad óptica del Cassegrain. Los sistemas de Schmidt-Cassegrain o Maksutov-Cassegrain, que se muestran en la figura XXII. no cumplen estrictamente con las características del Schmidt o del Maksutov; sin embargo, la calidad óptica supera de forma significativa a los de tipo Cassegrain.

4.5 Monturas para telescopios

Los telescopios astronómicos por razón natural deben estar montados en una base rígida, estable y adecuada. Por otro lado, el telescopio debe moverse lentamente, por medio de algún mecanismo al que llamamos reloj, para seguir a las estrellas en su movimiento diurno, con un mínimo de problemas.

Cualquier montura debe ofrecer las siguientes características:

1. Rigidez del tubo en cualquier ángulo de elevación. Idealmente la imagen no debe sacudirse cuando la perilla del ocular se gira para enfocar.

2. Facilidad de seguir el movimiento del objeto, ocasionado por la rotación de la Tierra.

Existen una gran variedad de monturas pero las más importantes son la altiazimutal y la ecuatorial que se describirán a continuación.

4.5.1 La montura altiazimutal

Esta montura es la más sencilla y fue utilizada en los primeros telescopios. Tiene la configuración de la figura XXIII., con un eje vertical que permite el movimiento sobre un plano horizontal, y un eje horizontal cuya orientación cambia al mover el eje vertical. La función de este eje horizontal es cambiar la altura de observación. Dicho de otro modo los ejes horizontal y vertical determinan el azimut y la altura respectivamente.

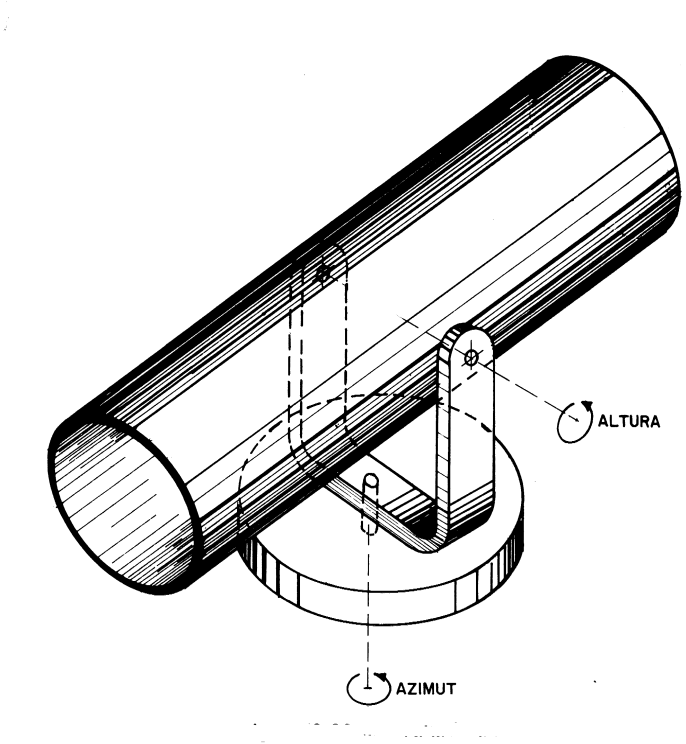


Fig. XXIII Montura Altiazimutal

La montura altiazimutal es la más sencilla y la ideal para observaciones terrestres. En los telescopios modernos también se utiliza la montura altiazimutal debido a su gran rigidez que no permite grandes flexiones. Sin embargo, esta montura no resulta práctica cuando lo que se pretende es dar seguimiento a un astro, ya que, como se puede deducir fácilmente, se necesitaría controlar el movimiento de dos ejes.

4.5.2 La montura ecuatorial

La montura ecuatorial, al igual que la montura altiazimutal, tiene dos ejes mútuamente perpendiculares, pero con la diferencia de que el eje que conserva fija su posición no es vertical sino paralelo al eje de la Tierra y recibe el nombre de *eje polar*. De esta manera, la montura ecuatorial contrarresta el movimiento de rotación de la Tierra directamente, utilizando el giro de alrededor de un sólo eje.

El eje polar recibe también el nombre de eje de ascensión recta y el eje perpendicular a éste el nombre de eje de declinación.

Existen muchos tipos de monturas ecuatoriales, las más comunes se muestran a continuación.

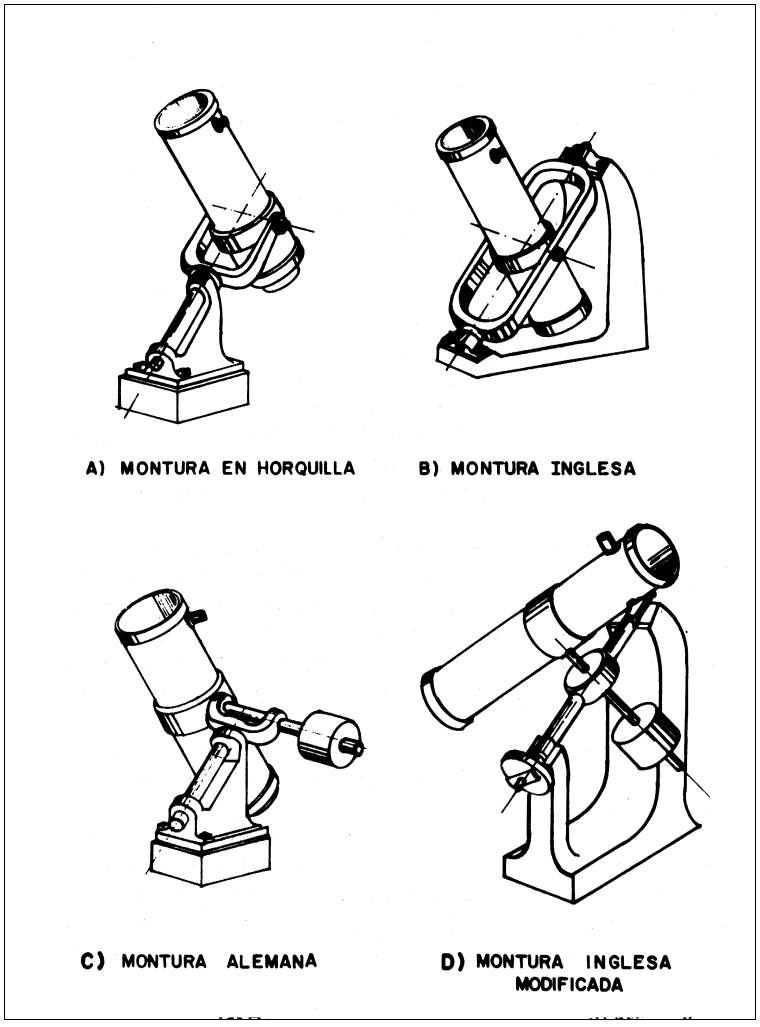


Fig. XXIV Distintas configuraciones de la montura ecuatorial

Básicamente podemos clasificar a las monturas ecuatoriales en simétricas (como las que se encuentran en la parte superior de la figura) o asimétricas (parte inferior).

Generalmente las monturas ecuatoriales son provistas de contrapesos (como puede claramente observarse en la montura alemana en la parte inferior izquierda), los cuales permiten mantener al telescopio en equilibrio en cualquier posición.

Como mencionamos anteriormente, la montura ecuatorial contrarresta mediante el movimiento de un sólo eje (el de ascensión recta) el movimiento de la Tierra. Para que esto suceda es necesario acoplar a este eje un mecanismo impulsor, llamado comúnmente reloj, con una velocidad igual a 15 segundos de arco por un segundo de tiempo. Actualmente, la mayoría de los telescopios con montura ecuatorial tienen acoplados relojes eléctricos directamente al eje de ascensión recta a través de un engrane.

5 Mecanismos de control de telescopios

Existen muchas razones por las que se requiere controlar un telescopio utilizando una microcomputadora, pero tal vez la razón principal es la del costo/beneficio en términos de incrementar la eficiencia del trabajo de los científicos para la adquisición de información, en cuyo caso el costo de los mecanismos de control queda perfectamente justificado.

Otra razón sustentable para la computarización de los pequeños telescopios es el bajo costo del hardware en nuestros días.

Además el uso de telescopios computarizados por los astrónomos amateurs ha hecho posible grandes mejoras en el desarrollo de investigaciones, con sólo una computadora personal y alguna herramienta que cierre el lazo del sistema.

El control de telescopios con computadoras ha evolucionado al paso del tiempo, y podemos describirlo de la siguiente manera. Primero, los sistemas desarrollados por marcas comerciales, que permiten controlar al telescopio con el uso de una EPROM y teniendo además interfase RS-232; segundo, el desarrollo de monturas diseñadas especialmente para el control por computadora; tercero, el control hecho completamente por los aficionados, donde nosotros estaríamos ubicados; y el cuarto y último, los sistemas de control más avanzados y especializados.

En algunas situaciones el control de telescopios puede ser de gran ayuda o incluso esencial. Un buen ejemplo podría ser el telescopio automático del polo sur donde nadie esta ahí para operarlo. Otro ejemplo son los telescopios infrarrojos (IR), los cuales requieren de una gran precisión para apuntarlos y los telescopios de múltiples espejos como el TIM (*Telescopio Infrarrojo Mexicano)* donde se requiere de un control especial para mantener los espejos alineados.

Debemos antes de continuar entender la diferencia entre la automatización y la computarización de un telescopio. La automatización como ya se menciona anteriormente se refiere a que no se necesita la intervención humana en ninguna de las funciones de control

del telescopio; y la computarización en cambio se refiere sólo a agregarle una computadora al sistema de control, la cual es una solución de bajo costo para la mayoría de los problemas de control de los telescopios. Atrás de una gran eficiencia en la captura de datos se encuentra siempre un control de telescopio por microcomputadora y una instrumentación que incrementa la rentabilidad de dicha adquisición de datos. Una captura manual de dichos datos después de una sesión de observación que se ha extendido hasta altas horas de la noche es propicia a errores. El astrónomo que es ayudado por sistemas de posicionamiento de telescopios tiene más tiempo libre para pensar en astronomía y olvidarse del manejo del telescopio lo cual constituye, además, un trabajo monótono y no crítico para sus investigaciones.

5.1 Opciones de Control

Todos los sistemas de posicionamiento y seguimiento de telescopios son principalmente de lazo cerrado (desde retroalimentación humana hasta fotómetros).

A continuación analizaremos las opciones que podemos utilizar para el control de telescopios.

5.1.1 Sistema Clásico o de lazo cerrado

El primer tipo es el *sistema clásico de lazo cerrado*. En este sistema se tiene un sensor angular de posición en cada eje del telescopio y una microcomputadora que compare la posición actual (dada por los sensores) con la deseada (la posición de la estrella), además de hacer lo necesario para lograr que las dos posiciones sean las mismas o se acerquen lo más posible. Como ya se explicó sólo se sensa la posición angular de los ejes y no la posición de la estrella directamente, por lo que no tenemos error al azar o sistemático entre el sensor y el plano de la imagen. Sin embargo, sí tenemos error por la flexión de la montura y la refracción atmosférica. Estos errores pueden ser modelados en la computadora, y utilizando constantes de modelado obtenidas mediante calibración especial para cada tipo de error, se reducen hasta ser despreciables.

Es muy importante resaltar el papel de la computadora en este sistema, pues es ella quien cierra el lazo constantemente, calculando y corrigiendo los errores antes mencionados, por lo que se necesita que la computadora sea multitarea y tenga asignada preferentemente una interrupción para atender las peticiones de los motores.

Desgraciadamente el sistema clásico sensa los errores en los ejes del telescopio y no en la posición de la estrella por sí misma, dejando sin modelar algunos errores sistemáticos que se deberán de resolver de otras maneras.

5.1.2 Sistema de lazo abierto

El segundo tipo de sistema es el *no clásico o de lazo abierto*. Este tipo de sistema trata de detectar los errores en la posición de la estrella misma, detectando y corrigiendo los errores en ese momento, sin complicados cálculos para la computadora. Sin embargo, los sensores

para la posición de la estrella, en su mayoría fotómetros fotoeléctricos, son más difíciles de manejar y acoplar que los decodificadores angulares.

El telescopio se mueve de un área del cielo a otra en un lazo abierto (sin ninguna posición de retroalimentación), “pensando” que cuando llegue a su posición indicada la estrella va a estar ahí. Cuando llega a esa posición el fotómetro empieza a buscar la estrella, la obtiene y la centra, todo bajo el control de la computadora.

Es importante mencionar, que existen excepciones a lo último que mencionamos, un ejemplo de esto es el ojo del astrónomo, que puede funcionar como un sensor directo y barato de la posición de la estrella.

El propósito de un sistema de control de telescopios es apuntar el telescopio al objeto celeste y seguir el objeto lo más exactamente posible. En el sistema clásico el seguimiento es realizado por un decodificador angular en cada eje del telescopio. El sistema determina dónde debe estar el telescopio en cada momento, entonces se envía un comando al controlador del motor para mover el telescopio y de ésta forma este llega a la posición deseada en el tiempo adecuado. Éste es un ejemplo de un sistema general de control con servomotores.

Un sistema de control de lazo abierto encuentra la diferencia entre la posición deseada y la posición actual, y calcula el error, el cual se manda al controlador del motor del telescopio. El principal problema con los sistemas de lazo abierto, es que una vez que el comando de corrección es ejecutado, no hay forma de saber si el comando fue ejecutado con la precisión y exactitud deseada. Si el telescopio es movido manualmente, toda la información calculada sobre la posición en un sistema abierto será errónea. Esto es verdad si utilizamos un motor no incremental en los ejes del telescopio, pues la computadora no tiene forma de saber cuánto ha movido el telescopio el motor. Este tipo de situaciones requieren de algún medio que permita saber la posición del telescopio independiente del comando enviado al motor.

En cambio, en el sistema clásico de lazo cerrado, un dispositivo es colocado en el eje del telescopio para alimentar de la verdadera posición angular del eje al actual registro de posición. En este caso la computadora usa el actual angular del eje para calcular la señal de error. Cualquier control de lazo cerrado es llamado también servomecanismo o sólo servo. Este término es usado por los dispositivos que se usan en este tipo de sistemas de control.

5.2 Motores y controladores de motores

En esta sección describiremos los diferentes tipos de motores que pueden ser conectados a las microcomputadoras. Esto no pretende ser una guía técnica de motores que pueden ser utilizados para el control de telescopios, ya que continuamente salen al mercado nuevos tipos de motores; sólo se quiere hacer una revisión general de los motores comúnmente utilizados en control de telescopios para discusiones futuras.

5.2.1 Motores de esfuerzo de torsión de CD

Esta configuración es de un motor de esfuerzo de torsión de corriente directa montados directamente a los manejadores de los ejes. Esta configuración es utilizada e telescopios de 0.1 a 1.0 metros de apertura diseñados para aplicaciones especiales (seguimiento de satélites de órbita baja) que requieren de alta velocidad y una óptima calidad de ejecución en el servo. La mayoría de los telescopios utilizados para la investigación astronómica ordinaria no utilizan esta configuración.

Una de las ventajas de estos motores es que los errores debidos a los mecanismos de engranaje son eliminados, por que el motor se acopla directamente a los ejes del telescopio. Sin embargo, estos motores suelen ser demasiado caros.

5.2.2 Servo motores de CD

Existen varios tipos de servos: los de campo abierto, de magneto permanente (PM por sus siglas en ingles), de bobina móvil y de circuitos impresos.

Los de campo abierto utilizan bobinas ya sea en la armadura (rotor) o en el campo (estator). Al controlar la corriente en la armadura o en el campo, la velocidad del motor es controlada.

Los motores PM utilizan un magneto permanente en el estator en lugar de los campos de bobinas. Usualmente tienen dos cables, a diferencia de los de campo abierto que utilizan cuatro. Las ventajas son: un torque alto al inicio, una relación lineal de torque vs. velocidad, tamaño pequeño y peso ligero. Este tipo de servos son usados frecuentemente en los modernos telescopios.

Otro tipo son los motores de bobina móvil, los cuales son motores de PM con un distinto tipo de construcción. En vez de utilizar una armadura cilíndrica convencional, utilizan un disco delgado o un caparazón que no es de acero. Una variación es el motor de circuito impreso que utilizan hojas de cobre en un circuito impreso para formar una armadura delgada. Aunque los motores de bobina móvil tienen muy poca inercia, gran aceleración y otras cualidades, son más caros que los motores convencionales de PM y son muy poco utilizados en aplicaciones de control de telescopios.

El rendimiento de los servos los hacen altamente aceptables para aplicaciones de control de telescopios, con la única desventaja de que son más difíciles de conectar a las computadoras que los motores de pasos.

5.2.3 Motores de pasos

Ésta es la tercera configuración de motores que pueden ser utilizados. Un motor de pasos es típicamente un motor de CD con una armadura de magneto permanente que consten de una docena o más caras de imán y cuatro o más bobinas.

La velocidad máxima del motor es determinada por el número máximo de pulsos por segundo que puede traducir a pasos. Los servos y los motores de pasos comparten la propiedad de que el torque cae rápidamente cuando se incrementa la velocidad angular.

Como sea, los dos motores difieren en que en los servos esta relación de torque vs. velocidad es lineal mientras que en los motores de pasos es no-lineal.

Los motores de pasos tienen ventajas significativas sobre otros tipos motores. Primero, pueden ser utilizados sin una retroalimentación de posición o velocidad en un simple sistema de lazo abierto. La segunda ventaja es que el motor de pasos es muy fácil de manejar con una computadora.

Por estas razones, los motores de pasos son la mejor opción para construir un sistema de control para telescopios computarizados.

5.2.4 Controladores de servo motores e interfaces a la computadora.

Existen dos formas básicas de ligar servo motores de CD a las computadoras. La primera es usar una tarjeta de interfase con amplificadores de servos integrados. La segunda es utilizar una tarjeta empleando convertidores digital a analógico (D/A) para convertir el voltaje que se aplicara. La computadora traduce los datos del *bus* y la tarjeta los convierte al voltaje correspondiente.

5.3 Dispositivos de censado de posiciones angulares.

Podemos utilizar varios métodos para retroalimentar la posición angular de nuestro telescopio y cerrar nuestro sistema de servo control, todo esto con un amplio rango de exactitud y precios. La mayoría de los telescopios con un sistema de control con retroalimentación utilizan decodificadores angulares ópticos. Sin embargo, existen diferentes dispositivos que pueden servir para el mismo propósito, a continuación describiremos algunos utilizados en el control de telescopios.

5.3.1 Potenciómetros de precisión

Estos son usados frecuentemente junto con una fuente de poder de corriente directa regulada, es decir, necesitan un voltaje proporcional de corriente directa. En un servo control analógico de telescopios, pueden ser usados tanto para controlar la posición de entrada, como la posición actual de retroalimentación.

La función que es frecuentemente deseada en un sistema de control del telescopios es una relación lineal entre la resistencia y el ángulo del eje del telescopio. Cualquier desviación de la linealidad es una característica del potenciómetro y no puede ser desplazada ajustándola con otros parámetros del sistema.

La exactitud de los potenciómetros es limitada por la misma resolución y también está afectada por otros parámetros como la estabilidad de la temperatura, que debe ser del orden del 0.05% por grado centígrado. Estos datos dan una exactitud total de uno y medio grado de arco para observaciones en condiciones normales.

Transformadores de reducción variable

Los transformadores rotatorios variables diferenciales (RVDT) es otro tipo de sensor de posición angular. Son más exactos que los potenciómetros de posición. Su estabilidad de temperatura es mejor que la de los potenciómetros, pero son más caros y no muy comunes en el mercado electrónico. No son muy utilizados en el control de telescopios debido a su costo, el caro soporte a sus circuitos y la dificultad de construir una interface para conectarlos con la computadora.

5.3.2 Resolvedores (Resolvers)

Un resolvedor es un transformador con dos o más bobinas, con al menos una en el estator y una en el rotor. Éstas están acopladas de tal forma que el voltaje de salida es proporcional al seno o coseno de la posición angular del eje rotatorio, por lo que difiere del RVDT, el cual tiene una salida lineal.

Los resolvedores tienen una gran capacidad de exactitud y alta resolución (18 bits o más). Sin embargo su precio es muy elevado.

5.3.3 Synchros

Los synchros son transformadores que se usan en pares para trasmitir posiciones angulares sobre grandes distancias sin una unión mecánica directa. Los synchros se conectan juntos en un circuito eléctrico de tal forma que cuando un eje es girado, el otro eje también gira con el mismo ángulo.

Pueden ser usados en pares como una forma de sistema de control analógico de lazo abierto.

5.3.4 Inductosyns

Una variación de los resolvedores que se espera sea utilizada en el control de telescopios son los inductosyns, los cuales son una especie de resolver que usa tecnología de codificación óptica.

El inductonsyn está hecho usando técnicas de fotorresistencias, el cual deposita un modelo metálico en un substrato dieléctrico estable. Debido que el modelo es más simple de generar y reproducir que un decodificador óptico, el inductosyn puede tener un costo menor que el decodificador óptico, con una resolución y exactitud equivalente.

5.3.5 Decodificadores angulares ópticos

Los decodificadores angulares ópticos constan de un disco, usualmente de vidrio, en donde es guardado por métodos fotográficos uno o más patrones alternados de líneas o ranuras opacas o transparentes, que cuando el disco gira a través de un haz de luz y un fotodetector, se generan una serie de pulsos. El número de pares de líneas transparente y opacas alrededor del disco determina el número de pulsos por turno en el disco, lo cual es la resolución del decodificador.

Los decodificadores pueden dar una posición absoluta usando un código único de salida por cada ángulo(decodificador absoluto), o puede enviar un pulso de salida por cada incremento del ángulo (decodificador incremental). La resolución típica de un decodificador absoluto es de 2^N, donde N es el número de anillos concéntricos.

6 Desarrollo

A continuación, y una vez descrito el problema, el marco de referencia y las metodologías, tecnologías, técnicas y herramientas a aplicar; pasaremos a describir el capitulo central del presente trabajo de tesis, en el cual se tomará un solo caso de uso para ilustrar el proceso de desarrollo y ejemplificar los criterios tomados para la base del diseño.

Solicitamos al lector recordar que seguiremos un proceso de desarrollo iterativo pero sin aplicar ninguna metodología específica sino las mejores prácticas de éste. Antes de comenzar, si existe alguna duda al respecto, por favor consulte el capítulo 3.

6.1 Fase de Concepción.

Como vimos, el proceso de desarrollo inicia con una fase de definición de la problemática y de la solución posible, que conocemos como fase de concepción.

En esta fase lo más importante es conocer la visión del producto, lograr el consenso de ésta visión entre los involucrados para poder definir entonces el alcance del proyecto y la estrategia a seguir.

Como podemos intuir, gran parte de éste proceso lo hemos venido describiendo a lo largo del presente trabajo, al definir el problema, al seleccionar las herramientas a aplicar en la solución y al empezar el conocimiento del área de estudio. Por tanto, en esta sección sintetizaremos éstos conceptos y agregaremos algunos nuevos para completar nuestra fase de concepción.

6.1.1 Visión del proyecto global

Como vimos en el capítulo 1, la visión del proyecto general es:

Poner en funcionamiento un observatorio astronómico en las instalaciones de la Facultad de Ingeniería en Ciudad Universitaria

¿Por que es la visión?, por que es nuestra meta a más alto nivel. La idea que todos los miembros del equipo deben tener en mente para poder alcanzarla. ¿Por qué es importante definirla? Una visión clara y diseminada entre todos los miembros del equipo asegura un trabajo coordinado y una capacidad de toma de decisiones. Existe una metáfora que ilustra bien, la intención de definir la visión:

Para nuestra metáfora asumiremos que nos encontramos en medio de una construcción que no sabemos qué es en realidad, por lo que escogemos a tres trabajadores y preguntamos a cada uno qué es lo que está construyendo.

El primero nos contesta: *Estoy construyendo una pared.*

El segundo: *Estoy construyendo un templo nuevo.*

Y el tercero nos dice: *Estoy construyendo la casa de Dios.*

¿Cuál de los tres trabajadores realizará mejor su trabajo? Pues seguramente aquel que tiene una visión mucho más global de la aportación de su trabajo. Aquel trabajador que conoce hacia dónde irá su trabajo y lo importante que resulta dentro de todo el proyecto, así mismo este trabajador podrá tomar sus propias decisiones e impactar positivamente en el trabajo de los demás.

6.1.2 Objetivo del equipo de trabajo

El objetivo específico de nuestro proyecto es:

Diseñar un programa de computadora para obtener las coordenadas del objeto celeste a partir de la posición absoluta alojada en una base de datos, la interfaz de la computadora con la tarjeta controladora y la calibración de la posición inicial del telescopio siguiendo dos directrices de diseño: 1) el sistema diseñado será modular, de tal forma que puedan agregarse nuevas funciones y dispositivos más complejos. 2) La aplicación de diversas ramas de la ingeniería en el desarrollo de la astronomía.

6.1.3 Usuarios y dueños del negocio

Nuestros usuarios finales serán todos los miembros de la comunidad de la facultad de ingeniería interesados en la observación astronómica. Pero quienes definirán la funcionalidad y apariencia del sistema, debido a que conocen las reglas internas de la sociedad, sus procesos y forma de conducirse[36], serán un grupo selecto de miembros fundadores y miembros activos de la mesa directiva de SAFIR. A este grupo le llamamos *dueños del negocio*[37], debido a que conocen a detalle la problemática a resolver.

6.1.4 Funciones del sistema

Las funciones del sistema son lo que éste deberá hacer. La siguiente tabla muestra una síntesis de las funciones del sistema consensuadas entre los dueños del negocio y el equipo de trabajo. La Clasificación de las funciones es arbitraria.

Referencia	Función	Categoría
R1.1	El usuario deberá proporcionar un identificador y una contraseña para poder utilizar el sistema, de acuerdo al nivel de usuario.	Evidente
R1.2	El sistema deberá permitir asignar las autorizaciones de uso a los usuarios.	Evidente

TABLA VIII. Funciones de Validación de usuarios (R1)

[36] En general estos conceptos son referidos comúnmente como *reglas del negocio*.
[37] Esta será la traducción que utilizaremos para el término en inglés *Stakeholders*.

Referencia	Función	Categoría
R2.1	El sistema permitirá observar objetos de la bóveda celeste.	Evidente
R2.2	El sistema permitirá observar planetas y satélites.	Evidente
R2.3	El sistema deberá calcular a partir de las coordenadas absolutas las coordenadas relativas de un objeto celeste *np* (que no sea planeta).	Oculta
R2.4	El sistema debe ser capaz de obtener las coordenadas absolutas del usuario o de un catalogo.	Evidente
R2.5	El sistema deberá presentar las coordenadas actuales de un objeto en la bóveda celeste.	Evidente
R2.6	El sistema deberá mover el telescopio a las coordenadas relativas.	Evidente
R2.7	El sistema deberá mostrar las coordenadas relativas reales donde se encuentra el telescopio.	Evidente
R2.8	El sistema deberá llevar una bitácora de observación.	Oculta
R2.9	El sistema deberá registrar la observación actual.	Oculta
R2.10	El sistema deberá manejar un acola de observación.	Evidente
R2.11	El sistema deberá identificar aquellos objetos que no es posible observar por los limites del campo visual.	Oculta

TABLA IX. Funciones de Observación (R2)

Referencia	Función	Categoría
R3.1	El sistema deberá conocer en todo momento la posición del telescopio.	Oculta
R3.2	El sistema deberá permitir mover el telescopio solo a personas autorizadas.	Oculta

R3.3	El sistema deberá permitir realizar la configuración del Telescopio solo a personas autorizadas.	Evidente
R3.4	El sistema deberá configurar los movimientos del telescopio de acuerdo a los limites mecánicos y visuales.	Oculta
R3.5	El sistema deberá ser capaz de llevar a una posición de descanso al telescopio.	Evidente
R3.6	El sistema deberá evitar posiciones no deseadas del telescopio, con base en los limites mecánicos.	Oculta
R3.7	El telescopio deberá mantener su electrónica de fabrica y podrá moverse con esta o con la electrónica del sistema.	Oculta
R3.8	El sistema deberá permitir calibrar, probar la comunicación y reinicializar los motores y los sensores.	Evidente

TABLA X. Funciones de Control de Telescopio (R3)

Referencia	Función	Categoría
R4.1	El sistema deberá permitir la conexión de clientes remotos.	Opcional

TABLA XI. Funcionalidades extras deseadas (R4)

Explicación de la tabla

La tabla donde se sintetizó la funcionalidad requerida del sistema fue tomada del libro de Larman[38]. Consta de tres columnas:

La primera, denominada *Referencia*, que hace las veces de un identificador que nos ayuda a realizar el seguimiento de la funcionalidad a través de los casos de uso, como lo veremos más adelante, además de ser una manera de tipificar funcionalidades.

[38] LARMAN, Craig. *UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos*. Prentice Hall 1999, México, Pág. 42- 44

La segunda columna titulada *Función*, muestra una descripción de la funcionalidad requerida. Demos observar que en la tabla se listan las funcionalidades deseadas por los dueños del negocio, en esta fase no representa una lista definitiva.

Por último, la tercer columna llamada *Categoría* ayuda a clasificar las funcionalidades a fin de establecer prioridades entre ellas e identificar las que pudieran pasar inadvertidas. Se utilizan tres tipos de categorías:

- *Evidente*. Significa que la funcionalidad debe realizarse y el usuario debería saber que se ha realizado.
- *Oculta*. Debe realizarse, aunque no es visible para los usuarios. Esto aplica a servicios técnicos subyacentes.
- *Opcional*. Su inclusión no repercute significativamente en el costo ni en otras funciones.

6.1.5 Atributos del sistema

Los atributos del sistema son sus características o dimensiones, de ninguna manera funciones. La siguiente tabla muestra una síntesis de los atributos del sistema consensuados entre los dueños del negocio y el equipo de trabajo.

Atributo	Detalles y restricciones de frontera
Metáfora de interfaz	(detalle) Metáfora orientada a ventanas y cuadros de texto (detalle) Maximiza la navegación utilizando el puntero. (detalle) deben agregarse de manera fácil nuevos módulos y nuevos dispositivos.
Plataformas de sistema operativo	(detalle) Linux distribución Red Hat.

TABLA XII. Síntesis de atributos del sistema

6.1.6 Modelo de Casos de Uso

En este momento tenemos conocimiento, por lo menos de manera parcial, de qué requerimientos necesita nuestro sistema. Éstos requerimientos aparecen de manera aislada, es necesario comenzar a estructurar el problema y una práctica, actualmente muy utilizada, son los casos de uso.

Lo primero, antes de concentrarnos en lo que serán los casos de uso del sistema, es encontrar los actores del sistema[39], esto se realiza mediante algunas técnicas descritas anteriormente en el capitulo 3 y una lluvia de ideas entre los dueños del negocio y el equipo de desarrollo.

Los actores encontrados y su descripción, se muestran a continuación:

Nombre	Observador
Rol	
Descripción	La persona que hace al sistema una petición de observación para mover el telescopio al punto en la bóveda celeste.

TABLA XIII. Actor Observador

Nombre	Administrador
Rol	
Descripción	La persona que configura y da mantenimiento al catálogo de estrellas dentro del sistema.

TABLA XIV. Actor Administrador

Nombre	Motores
Rol	
Descripción	Son los encargados de llevar al telescopio a la posición deseada.

TABLA XV. Actor Motores

[39] En la jerga del desarrollo de sistemas se utiliza indistintamente el término *caso de uso* para referirse a una funcionalidad y el actor que la dispara o solo para referirse a la funcionalidad.

Nombre	Censores
Rol	
Descripción	Son los que permiten conocer la posición real del telescopio en todo momento.

TABLA XVI. Actor Censores

Ahora, pasemos a la descripción de los casos de uso.

Como vimos en el capitulo 3, los casos de uso son casos de utilización del sistema. Existen técnicas para identificarlos con facilidad. Nosotros utilizamos algunas preguntas sugeridas[40] y lluvia de ideas entre los dueños del negocio y el equipo de desarrollo.

En esta primera fase, no se pretende tener un conocimiento total del sistema (recordemos que estamos siguiendo un proceso iterativo) sino un panorama global. Algunos autores coinciden en que al finalizar la etapa de concepción debe tenerse alrededor de un 10% o 20% de avance en los casos de uso[41].

Los casos de uso pueden escribirse en diferentes formatos y expresarse con diverso grado de detalle y de aceptación de las decisiones concernientes al diseño. La decisión está en manos del equipo de desarrollo.

Para nuestro caso se inicio con un caso de uso de alto nivel, se pasó por 4 versiones del caso de uso, en cada una de las cuales se agregaba mayor detalle a la descripción. Para el desarrollo de éste capítulo se toman únicamente las versiones iniciales y finales.

Llamamos *caso de uso de alto nivel* a aquellos que describen el proceso muy brevemente, casi siempre en dos o tres enunciados. Conviene servirse de este tipo de casos de uso durante la fase de concepción, a fin de entender rápidamente el grado de complejidad y de funcionalidad del sistema[42]. Estos caso de uso son muy sucintos y vagos en las decisiones de diseño.

Además del grado de detalle, un caso de uso puede clasificarse en primario, secundario u opcional.

Los casos de primarios de uso representan los procesos comunes más importantes.

Los casos de uso secundarios representan procesos menores o raros.

Por último, los casos de uso opcionales representan procesos que pueden no abordarse.

[40] Capitulo 3, Pág. 28 y 29.
[41] Capitulo 3, Pág. 46.
[42] Ídem

A continuación se presentan los casos de uso de alto nivel encontrados. Al tratarse de una primera aproximación los llamamos, arbitrariamente, casos de uso a nivel 0 (cero):

Caso de uso 1: Administración de Usuarios

Actores: Administrador

Tipo: Primario

Descripción nivel 0:

El administrador necesita llevar un control de los usuarios que utilizan el sistema.

Descripción nivel 1:

El administrador da de alta, modifica o borra un usuario y asigna su contraseña, datos y perfil.

TABLA XVII. Caso de uso Administración de Usuarios nivel 0

Caso de uso 2: Configuración del sistema

Actores: Administrador

Tipo : Primario

Descripción nivel 0:

El administrador establece los parámetros iniciales de longitud, latitud, posición de descanso y rango de visibilidad del telescopio.

Descripción nivel 1:

1. El administrador registra la localidad o punto geográfico de observación (Longitud y latitud).
2. El administrador indica las coordenadas relativas de la posición de descanso del telescopio.
3. El administrador indica el ángulo mínimo sobre el horizonte donde hay visibilidad.
4. El administrador indica los ángulos fuera del rango de visibilidad de la cúpula.
5. El administrador indica cada una de las posibles posiciones de inicio.

TABLA XVIII. Caso de uso configuración del sistema nivel 0

Caso de uso 3: Mantenimiento al catálogo de Objetos Celestes NP

Actores: Administrador

Tipo: Primario

Descripción nivel 0:

El administrador necesita mantener actualizado el catalogo de los objetos celestes np a observar.

Descripción nivel 1:

El administrador agrega, borra o modifica los datos de un objeto celeste no dentro del catálogo.

TABLA XIX. Caso de uso Mantenimiento al catálogo de Objetos Celestes NP nivel 0

Caso de uso 4: Consultar Bitácora de Observación

Actores: Administrador

Tipo: Primario

Descripción nivel 0:

El administrador necesita conocer que personas realizaron observaciones para llevar un control de ellas.

Descripción nivel 1:

1. El administrador realiza una petición de consulta a la bitácora de observación.
2. El administrador indica el criterio por el que desea buscar: día, usuario, objeto celeste np, planeta o satélite.
3. El administrador obtiene los datos de acuerdo a su criterio de búsqueda.

TABLA XX. Caso de uso Consultar Bitácora de Observación nivel 0

Caso de uso 5: Consultar Tipos de Objetos Celestes NP

Actores: Usuario

Tipo: Primario

Descripción nivel 0:

El usuario consulta los atributos de un objeto celeste np en el catalogo de acuerdo al tipo de objeto al que pertenece.

Descripción nivel 1:

1. El usuario busca el objeto celeste np por tipo de objeto, es decir, galaxia, estrella, constelación, nebulosa, catálogo especial (messier, NGCM, NBC).

TABLA XXI. Caso de uso Consultar Tipos de Objetos Celestes NP

Caso de uso 6: Obtener Coordenadas Relativas de Objetos Celestes NP

Actores: Usuario

Tipo : Primario

Descripción nivel 0:

El usuario consulta las coordenadas absolutas del objeto celeste np, del catalogo o tecleándolas o de una lista de peticiones de observación, para obtener las coordenadas relativas.

Descripción nivel 1:

1. El usuario indica las coordenadas absolutas del punto a observar (ascensión recta, AR y declinación, DEC).

2. Obtiene siempre las coordenadas relativas (Angulo horario y declinación).

TABLA XXII. Caso de uso Obtener Coordenadas Relativas de Objetos Celestes NP

Caso de uso 7: Mover telescopio

Actores: Usuario, Motores

Tipo : primario

Descripción nivel 0:

Mover el telescopio a las coordenadas relativas de un objeto para hacer la observación.

Descripción nivel 1:

1. El usuario
    a. Ingresa las coordenadas relativas del objeto celeste np o
    b. Ingresa las coordenadas relativas de planeta o satélite o
    c. Un offset a partir de las coordenadas actuales del telescopio o.
    d. Recupera de una lista de coordenadas relativas observadas.
    e. Proporciona las coordenadas absolutas del objeto celeste np(caso de uso 6)
    f. Se calculan el número de pasos que se enviarán a los motores.
2. Los motores hacen girar los ejes del telescopio hacia la posición deseada (c.u. 8).

TABLA XXIII. Caso de uso Mover telescopio nivel 0

Caso de uso 8: Girar Ejes del Telescopio

Actores: Motores

Tipo: Primario

Descripción nivel 0:

Los motores giran los ejes del telescopio para llevarlo a la posición deseada.

Descripción nivel 1:

De acuerdo a la información recibida de sentido y número de pasos se giran los ejes del telescopio.

TABLA XXIV. Caso de uso Girar Ejes del Telescopio

Caso de uso 9: Informar la posición Actual del Telescopio.

Actores: Sensores

Tipo: Primario

Descripción nivel 0:

Los sensores deben saber la posición del telescopio.

Descripción nivel 1:

Ante cualquier movimiento del telescopio, los sensores obtienen su posición.

TABLA XXV. Caso de uso Informar la posición Actual del Telescopio. Nivel 0

Caso de uso 10: Configuración de motores

Actores: Administrador

Tipo : Primario

Descripción nivel 0:

El administrador establece los parámetros de velocidad, la longitud de los pasos (pasos completos o medios) y la dirección.

Descripción nivel 1:

El administrador registra la velocidad deseada de los pasos, la longitud (pasos medios o completos) de éstos y la dirección (definición del sentido de giro).

TABLA XXVI. Caso de uso Configuración de motores. Nivel 0

Caso de uso 11: Calculo de errores

Actores: Administrador

Tipo : Primario

Descripción nivel 0:

Pendiente ...

Descripción nivel 1:

Pendiente...

TABLA XXVII. Caso de uso Calculo de errores. Nivel 0

Caso de uso 12: Reinicialización

Actores: Administrador.

Tipo : Secundario

Descripción nivel 0:

Colocar en valores iniciales los parámetros de todo el sistema.

Descripción nivel 1:

1. El usuario elige el parámetro de sistema (Cuenta de los sensores, Posición actual del telescopio) que se reinicializará.

2. Asigna el valor inicial guardado al parámetro o lo teclea directamente.

TABLA XXVIII. Caso de uso Reinicialización. Nivel 0

Caso de uso 13: Comunicaciones

Actores: Administrador.

Tipo : Primario

Descripción nivel 0:

Probar comunicación con los dispositivos de I/O que conectan a los motores y los sensores al sistema.

Descripción nivel 1:

1. El administrador elige los valores de configuración de los dispositivos de comunicaciones.
2. El administrador envía datos de prueba por los dispositivos de comunicaciones a los motores y sensores.
3. El administrador valida que los datos se hayan enviado.

Fig. XXV Caso de uso Comunicaciones. Nivel 0

Caso de uso 14: Calibrar

Actores: Administrador.

Tipo : Primario

Descripción nivel 0:

Calibrar los motores y los sensores, asignando una equivalencia grados-pasos de motor y grados-pasos del sensor.

Descripción nivel 1:

1. El administrador escoge el dispositivo a calibrar.
2. El administrador asigna la equivalencia en grados-pasos para el dispositivo.
3. El administrador verifica visualmente que la calibración este hecha.

TABLA XXIX. Caso de uso Calibrar. Nivel 0

Caso de uso 15: Encolar petición de observación.

Actores: Usuario

Tipo : Secundario

Descripción nivel 0:

Forma la petición de observación de coordenadas absolutas para que se mueva el telescopio a este punto.

Descripción nivel 1:

1. El usuario indica las coordenadas absolutas del punto a observar (ascensión recta, AR y declinación, DEC).

2. Se guarda la petición en una lista de observaciones.

TABLA XXX. Caso de uso Encolar petición de observación. Nivel 0

Caso de uso 16: Mantenimiento al catálogo de planetas y satélites

Actores: Administrador

Tipo: Primario

Descripción nivel 0:

El administrador necesita mantener actualizado el catalogo de planetas y satélites a observar.

Descripción nivel 1:

1. El administrador agrega, borra o modifica los datos de un planeta o satélite dentro del catálogo.

TABLA XXXI. Caso de uso Mantenimiento al catálogo de planetas y satélites. Nivel 0

Caso de uso 17: Consultar Planetas y Satélites

Actores: Usuario

Tipo: Primario

Descripción nivel 0:

El usuario consulta los atributos de un planeta o satélite en el catálogo.

Descripción nivel 1:

El usuario busca el planeta o satélite por nombre, masa, distancia media al sol, excentricidad, etc.

TABLA XXXII. Caso de uso Consultar Planetas y Satélites. Nivel 0

Caso de uso 18: Obtener Coordenadas Relativas de Planetas y Satélites

Actores: Usuario

Tipo : Primario

Descripción nivel 0:

El usuario consulta las coordenadas absolutas del planeta y/o satélite, del catalogo o tecleándolas o de una lista de peticiones de observación, para obtener las coordenadas relativas.

Descripción nivel 1:

2. El usuario indica las coordenadas absolutas del planeta o satélite (ascensión recta, AR y declinación, DEC) de acuerdo al día de observación o las toma del catalogo de planetas o satélites.

3. Obtiene las coordenadas relativas (Angulo horario y declinación) de pendiendo de la fecha y parámetros especiales del planeta o satélite.

TABLA XXXIII. Caso de uso Obtener Coordenadas Relativas de Planetas y Satélites. Nivel 0

Hemos descrito los casos de uso y los actores, a continuación presentamos su representación gráfica en UML:

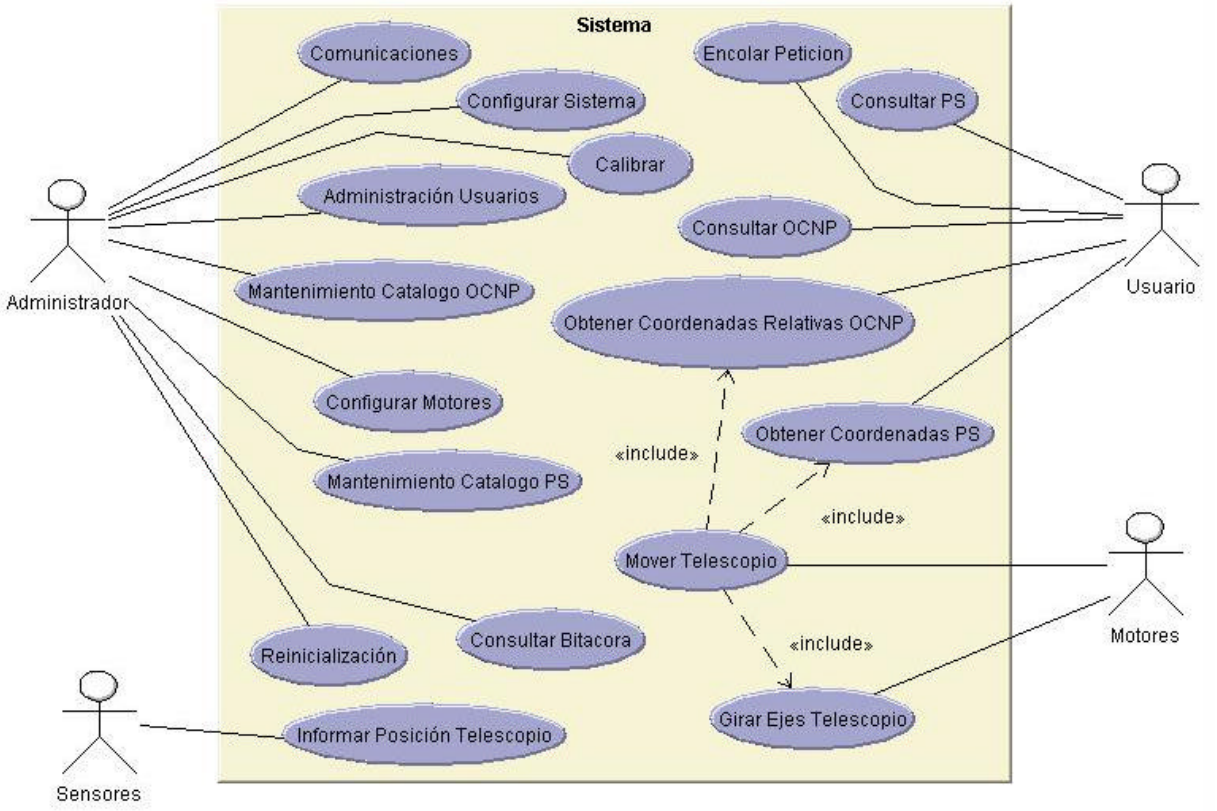


Fig. XXVI Diagrama de casos de uso del sistema en estudio

6.1.7 Prototipo del sistema

Una vez que tenemos el modelo de casos de uso que representa la funcionalidad deseada de nuestro sistema, es momento de hacer un prototipo que nos sirva de base para expandir los casos de uso encontrados, referenciándolos a una posible forma en como serán percibidos por los actores cuando interactúen con el producto de software. El prototipo realizado se encuentra en el anexo A del presente trabajo.

6.2 Fase de Elaboración

Una vez que hemos definido el problema que queremos resolver durante la fase de concepción, es momento de iniciar un diseño más detallado del sistema.

Iniciaremos expandiendo los casos de uso, basados en el prototipo realizado, que nos permitan organizar en un futuro los ciclos de desarrollo.

Como mencionamos al inicio del capitulo, elegiremos un caso de uso que nos ayude a explicar el proceso de desarrollo seguido, haciéndolo evolucionar en cada una de las fases. Sin embargo, pueden consultarse todos los casos de uso y sus diagramas UML derivados en el anexo B del presente trabajo.

El caso de uso seleccionado para este fin es el caso de uso *Mover Telescopio*, que, como podemos ver en el diagrama, incluye otros tres casos de uso más: *Obtener coordenadas relativas de Objetos Celestes No Planetas, Obtener coordenadas de Planetas y Satélites,* y *Girar ejes Telescopio*.

Para evitar alargar demasiado el desarrollo, eliminaremos de la lista anterior el caso de uso *obtener coordenadas Planetas y satélites*, pues consideramos que si la exposición es lo suficientemente clara, éste caso de uso puede ser fácilmente desarrollado de manera independiente.

Seleccionar un caso de uso de inicio es cosa común (y buena práctica) durante el proceso de desarrollo. La estrategia general consiste en escoger primero los casos que influyen profundamente en la arquitectura básica[43]. Para nuestro caso, resulta obvio que el caso de uso seleccionado es el que tiene más influencia en la arquitectura, debido a que es la principal funcionalidad de nuestro sistema.

6.2.1 Expansión de los casos de uso seleccionados

Un caso de uso expandido describe un proceso más afondo que el de alto nivel. La diferencia básica con el caso de uso de alto nivel consiste en que tiene una sección destinada *al curso normal de los eventos*, que los describe paso por paso. Durante esta fase, conviene escribir en formato expandido los casos más importantes y de mayor influencia; en cambio los menos importantes pueden posponerse hasta el ciclo de desarrollo en el cual van a ser abordados.

Existen varios formatos para expandir los casos de uso, la decisión nuevamente pertenece al equipo de diseño. Nosotros utilizamos un formato propuesto por Martín Fowler[44], cuyo autor es Alistair Cockburn,[45] por que consideramos que éste formato muestra de una manera resumida y ordenada, toda la información necesaria para definir el caso de uso a un nivel detallado antes de iniciar el diseño.

[43] LARMAN, Craig. *UML y Patrones...* Pág. 75
[44] FOWLER, Martin y SCOTT Kendall, UNL Gota a Gota…pag. 59
[45] http://members.aol.com/acockburn

A continuación se muestra el formato utilizado, al cual se le agregó alguna información adicional con respecto al formato original, la explicación de cada dato y algún comentario de considerarse necesario:

Caso de Uso #	*Aquí se escribe el nombre del caso de uso. Se recomienda escribir el nombre del caso de uso como una frase corta que empiece con un verbo, la cual describa de la mejor manera el objetivo del caso de uso.*		
Tipo	*El tipo del caso de uso, tal y como lo hicimos cuando desarrollamos los casos de uso de nivel cero en la fase anterior (esta información fue agregada por los autores)*		
Objetivo	*Una descripción detallada del objetivo del caso de uso (de considerarse necesaria)*		
Referencias	*Los identificadores de las referencias cruzadas de las funciones relacionadas al sistema.*		
Alcance y Nivel	*Alcance se refiere a qué elemento es considerado la "caja negra" durante el diseño.* *Nivel se refiere a alguno de los siguientes calificadores: Resumen, Acción primaria ó Subfunción*		
Pre-condiciones	*Lo que se espera que ya este listo o completado en el estado del ambiente.*		
Condición para fin exitoso	*El estado del ambiente después de un fin exitoso.*		
Condición para fin erróneo	*El estado del ambiente después de un objetivo abandonado.*		
Actores primarios y secundarios	*El nombre del actor principal y de los otros actores involucrados.*		
Disparador	*La acción sobre el sistema que dispara el caso de uso.*		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	*Aquí van los pasos del escenario que el actor realiza. Desde el disparador hasta alcanzar al objetivo*	*Aquí van los pasos del escenario que el sistema realiza. Desde el disparador hasta alcanzar al objetivo*
Extensiones	**Paso**	**Acción de la ramificación**	
	1.a	*Aquí va la condición que causa la desviación. Es necesario escribir las acciones a tomar en la bifurcación o el nombre del caso de uso.*	
Variaciones		**Acción de la ramificación**	
	1	*Lista de variaciones*	

TABLA XXXIV. Formato para la descripción de caso de uso

A continuación se muestran los casos de uso seleccionados, expandidos mediante el formato anteriormente explicado:

Caso de Uso 7	MOVER TELESCOPIO		
Tipo	Primario.		
Objetivo	Mover el telescopio a las coordenadas relativas de un objeto para hacer la observación.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que el telescopio este calibrado. Que exista comunicación con los motores. Tiene que saber la posición en la que se encuentra (Implica que exista comunicación con los sensores).		
Condición para fin exitoso	Que el telescopio, llegue al punto deseado.		
Condición para fin erróneo	El telescopio no llegue al punto deseado.		
Actores primarios y secundarios	Usuario, Motores, Sensores		
Disparador	Después de que proporciona las coordenadas relativas o absolutas desde un catalogo, ejecuta mover el telescopio.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el usuario elige la opción de mover telescopio.	
	2		El sistema recibe las coordenadas relativas
	3		Obtiene la posición actual.
	4		Se calculan el número de pasos que se enviarán a los motores, la dirección y el tipo de pasos completos o micropasos.
	5		Se avisa al usuario que se comenzarán a enviar los datos al motor para que se muevan.
	6		Se envían los datos a los motores. (Caso de uso 8)
	8		Si avisa al usuario que se ha terminado de enviar los datos para que se muevan los motores.

	9		Después de hacer los cálculos necesarios, presenta información adicional de la posición.
	10	El usuario realiza la observación en el telescopio	
	11		Actualiza la ultima posición reportada por los sensores.
Extensiones	**Paso**	**Acción de la ramificación**	
	4	En caso de que no exista comunicación con los motores, se notifica al usuario y termina el caso de uso.	
Variaciones	**Paso**	**Acción de la ramificación**	
	2	Las coordenadas relativas se teclean directamente	
	2	Las coordenadas las calcula el sistema después de tomar las coordenadas absolutas del catálogo de objetos celestes (Caso de uso 6).	
	2	Las coordenadas las calcula el sistema después de tomar las coordenadas absolutas del catálogo de planetas o satélites (Caso de uso 18).	
	2	Un offset a partir de las coordenadas actuales del telescopio.	
	2	O recupera las coordenadas relativas de una lista de coordenadas relativas observadas.	
	3	Si se mueve por primera vez, la posición actual es la posición de inicio en la configuración	
	3	Si ya se movió anteriormente obtiene la posición registrada como posición actual (que se actualiza por los sensores)	

Caso de Uso 6	OBTENER COORDENADAS RELATIVAS DE OBJETOS CELESTE NP		
Tipo	Primario, esencial.		
Objetivo	El usuario consulta las coordenadas absolutas del objeto celeste np, del catalogo o tecleándolas, para obtener las coordenadas relativas.		
Referencias			
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que estén creados los usuarios. Si se busca un objeto en específico que este exista en el catálogo de objetos celestes. El sistema debe conocer a fecha y hora actual. Debe estar configurado (Caso de uso 2)		
Condición para fin exitoso	El usuario encuentra las coordenadas absolutas del objeto celeste np y obtiene las coordenadas relativas.		
Condición para fin erróneo	El usuario NO encuentra las coordenadas absolutas del objeto celeste np y NO obtiene las coordenadas relativas. (Por error, o por que el objeto no esta en catálogo).		
Actores primarios y secundarios	Usuario		
Disparador	El usuario solicita la transformación de las coordenadas absolutas.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el usuario solicita la transformación de coordenadas absolutas.	
	2		El sistema ejecuta los algoritmos de transformación de coordenadas absolutas a relativas. En base a la posición actual del telescopio (Caso de uso 9) y en base a la configuración de la localidad y hora actual. (Caso de uso 2) Determina además si el objeto es visible en ese momento (por limites de visibilidad y configuración).

	3		El sistema presenta las coordenadas relativas del objeto celeste np. Presenta información adicional de sí se puede mover el telescopio a ese punto.
	4		Se despliega la pantalla el resultado de la consulta elegida
Extensiones	**Paso**	**Acción de la ramificación**	
	2	Si el objeto no puede observarse con el telescopio, el sistema se lo notifica al usuario.	
Variaciones	**Paso**	**Acción de la ramificación**	
	1	El usuario proporciona las coordenadas absolutas directamente.	
	1	El usuario obtiene las coordenadas absolutas desde el catálogo de objetos celestes np.	
	1	O de una lista de peticiones de observación.	

Caso de Uso 18	OBTENER COORDENADAS RELATIVAS DE PLANETAS O SATÉLITES.		
Tipo	Primario, esencial.		
Objetivo	El usuario consulta las coordenadas absolutas del planeta y/o satélite, del catalogo o tecleándolas o de una lista de peticiones de observación, para obtener las coordenadas relativas.		
Referencias			
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que estén creados los usuarios. Si se busca un planeta o satélite en específico que este exista en el catálogo de planetas y satélites. El sistema debe conocer a fecha y hora actual. Debe estar configurado (Caso de uso 2)		
Condición para fin exitoso	El usuario encuentra las coordenadas absolutas del objeto celeste np y obtiene las coordenadas relativas.		
Condición para fin erróneo	El usuario NO encuentra las coordenadas absolutas del objeto celeste np y NO obtiene las coordenadas relativas. (Por error, o por que el objeto no esta en catálogo).		
Actores primarios y secundarios	Usuario		
Disparador	El usuario solicita la transformación de las coordenadas absolutas.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el usuario solicita la transformación de coordenadas absolutas del planeta o satélite.	

	2		El sistema ejecuta los algoritmos de transformación de coordenadas absolutas a relativas. En base a la posición actual del telescopio (Caso de uso 9), en base a la configuración de la localidad y hora actual. (Caso de uso 2), en base a la fecha de observación y en base a los parámetros especiales de los planetas o satélites(excentricidad, etc.) Determina además si el objeto es visible en ese momento (por limites de visibilidad y configuración).
	3		El sistema presenta las coordenadas relativas del planeta o satélite. Presenta información adicional de sí se puede mover el telescopio a ese punto.
	4		Se despliega la pantalla el resultado de la consulta elegida
Extensiones	**Paso**	**Acción de la ramificación**	
	2	Si el planeta o satélite no puede observarse con el telescopio, el sistema se lo notifica al usuario.	
Variaciones	**Paso**	**Acción de la ramificación**	
	1	El usuario proporciona las coordenadas absolutas directamente.	
	1	El usuario obtiene las coordenadas absolutas desde el catálogo planetas o satélites.	
	1	O de una lista de peticiones de observación.	

Caso de Uso 8	GIRAR EJES DEL TELESCOPIO		
Tipo	Primario, esencial		
Objetivo	Que los motores giren los ejes del telescopio para llevarlo a la posición deseada.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que exista comunicación entre los motores y el sistema. Necesita que el caso de uso 7 se haya iniciado.		
Condición para fin exitoso	Que los motores giren los ejes del telescopio de acuerdo a los datos enviados.		
Condición para fin erróneo	Que los motores NO giren los ejes del telescopio de acuerdo a los datos enviados.		
Actores primarios y secundarios	Motores		
Disparador	Que el caso de uso 7 se haya iniciado.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1		El sistema informa de que comenzará a enviar los datos a los motores.
	2		El sistema envía una serie de datos que contiene la dirección, pasos y si son completos
	3	El motor gira hacia el sentido indicado y número de pasos indicado.	
	4		El sistema informa cuando ha terminado de enviar los datos a los motores.
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	

Los demás casos de uso expandidos pueden consultarse en el anexo B del presente trabajo.

6.3 Fase de Construcción

Ahora bien, una vez concluida la fase de concepción y elaboración, y que los casos de uso han sido identificados, clasificados y programados. Comienza una fase de transición importante: la fase de construcción. Es en esta fase donde nuestro esquema de desarrollo iterativo comienza. Debemos escoger algunos casos de uso (o todos, depende del tamaño de los mismos y las decisiones del equipo de diseño) y comenzarlos a construir, empezando por el análisis, levándolos al diseño, construyéndolos y probándolos; para después llevarlos a un segundo grado de especialización, hacerlos recorrer de nueva cuenta todo el ciclo y así sucesivamente, hasta que alcancemos la funcionalidad total deseada.

6.3.1 Desarrollo del modelo conceptual.

El modelo conceptual explica (a sus creadores) los conceptos significativos en un dominio del problema; Según Larman, es el artefacto más importante a crear durante el análisis orientado a objetos, ya que los casos de uso son un importante artefacto de análisis de requerimientos, pero realmente no están orientados a objetos[46].

Identificar muchos objetos o conceptos constituye la escencia del análisis orientado a objetos y el esfuerzo se compensa con los resultados conseguidos durante la fase de diseño e implementación.

Una cualidad esencial que debe ofrecer un modelo conceptual es que representa cosas del mundo real, no componentes del software. Es, en términos prácticos, un mapa mental.

Algunos autores coinciden en que es frecuente omitir conceptos durante la fase inicial de identificación y descubrirlos más tarde cuando se examinen atributos y/o asociaciones. Cuando se detecten, habrá que incorporarlos al modelo conceptual.

Otro consejo útil es el no excluir un concepto simplemente por que los requerimientos no indiquen una necesidad evidente que permita recordar la información acerca de ella o por que el concepto carezca de atributos.

Existen varias técnicas sugeridas para ayudar a identificar conceptos, citaremos sólo una de ellas, que fue la utilizada por el equipo de desarrollo: la identificación de a partir de frases nominales.

La técnica consiste en identificar las frases nominales en las descripciones textuales del dominio de un problema y considerarlas como conceptos. Los casos de uso son una excelente descripción que puede utilizarse para este análisis. A manera de ejemplo utilizaremos una parte del caso de uso *Mover telescopio*, para identificar, dentro de la narración, los sustantivos que podrían utilizarse como candidatos de conceptos:

[46] LARMAN, Craig. *UML y Patrones*. . Pág. 85

…

2. El **sistema** recibe las **coordenadas relativas**
*3. Obtiene la **posición actual**.*
*4. Se calculan el número de **pasos** que se enviarán a los **motores**, la **dirección** y el tipo de **pasos completos o micropasos**.*
*5. Se avisa al **usuario** que se comenzarán a enviar los **datos** al **motor** para que se muevan.*

…

Como podemos ver, de la pequeña extracción de la descripción del caso de uso se obtuvieron (señalados con negritas) varios sustantivos que podrían ser candidatos a conceptos, este ejercicio se sigue con todos los casos de uso, obteniendo una lista extensa de sustantivos que serán discriminados por el equipo de diseño mediante una lluvia de ideas, quitando los sinónimos y agrupando sustantivos de manera intuitiva los conceptos para obtener generalizaciones y atributos.

Por ejemplo, del ejercicio anterior, observamos que podemos agrupar los conceptos de: ***motores, dirección, pasos, pasos completos, micro pasos***; ya que todos tienen relación con el concepto de ***motor***.

Después de ésta agrupación quitamos los posibles sinónimos y los conceptos que pudieran ser atributos. Como podemos intuir, los conceptos de ***dirección, pasos, pasos completos,*** *y* ***micro pasos***, son atributos del concepto ***motor****.*

Tal vez un error frecuente cuando se crea un modelo conceptual es el de representar algo como atributo, cuando debió ser un concepto. Una regla práctica para no caer en él es: *si en el mundo real no consideramos algún concepto X como número o texto, probablemente X sea un concepto y no un atributo*[47].

Una debilidad de la técnica de identificación por frases nominales es que la imprecisión del lenguaje natural puede provocar que varias frases nominales designen el mismo concepto o atributo, entre otras ambigüedades que pueden presentarse.

A la larga, la experiencia podrá ayudarnos a identificar de manera fácil los conceptos del campo en estudio; Mientras tanto, puede combinarse la técnica descrita con otras técnicas.

Para nuestro caso, se identificaron todos los conceptos mediante el análisis de frases nominales de todos los casos de uso y se discriminaron los sinónimos, conceptos irrelevantes y atributos mediante una lluvia de ideas.

Una vez identificados los conceptos debemos pasar ahora a relacionarlos mediante sus asociaciones.

La asociación es una relación entre dos conceptos que indica alguna conexión significativa e interesante entre ellos[48].

[47] LARMAN, Craig. *UML y Patrones.*.. Pág. 97
[48] Ídem, Pág. 105

Para agregar las asociaciones puede utilizarse una lista de comprobación como la que se muestra a continuación:

Categoría[49]:

- A es una parte física de B
- A es una parte lógica de B
- A está físicamente contenido en B
- A esta contenido lógicamente en B
- A es una descripción de B
- A es un elemento de línea de una transacción o reporte B
- A sé conoce/introduce/registra/presenta/captura/ en B
- A es miembro de B
- A es una subunidad organizacional de B
- A usa o dirige a B
- A se comunica con B
- A se relaciona con una transacción B
- A es una transacción relacionada con otra transacción B
- A está contiguo a B
- A es propiedad de B

A continuación se enumeran algunas categorías consideradas *de alta prioridad* que siempre conviene incluir en un modelo conceptual:

- A es una parte física o lógica de B
- A está físicamente o lógicamente contenido en B
- A está registrado en B

Aunque las asociaciones son importantes, no es necesario invertir un tiempo excesivo en su investigación. Es más importante identificar los conceptos que las asociaciones.

[49] LARMAN, Craig. *UML y Patrones*. . Pág. 108 y 109.

Como vemos hemos avanzado en la construcción del modelo conceptual relacionando los conceptos encontrados mediante asociaciones. Ahora, es momento de incorporar los atributos.

En este momento vale la pena recordar que un modelo conceptual no es modelo de componentes de software (aunque veremos más adelante que a partir del primero evolucionaremos al segundo, también llamado diagrama de clases).

Un atributo es un valor lógico dentro de un objeto[50].

Los atributos no deberían servir para relacionar conceptos en el modelo. La violación más frecuente de ésta regla consiste en agregar un tipo de atributo de llave foránea, lo cual suele hacerse con los diseños de base de datos relaciónales, a fin de asociar dos entidades.

Al integrar los conceptos, asociaciones y atributos que se descubrieron mediante el proceso descrito anteriormente obtenemos el conceptual que se muestra a continuación:

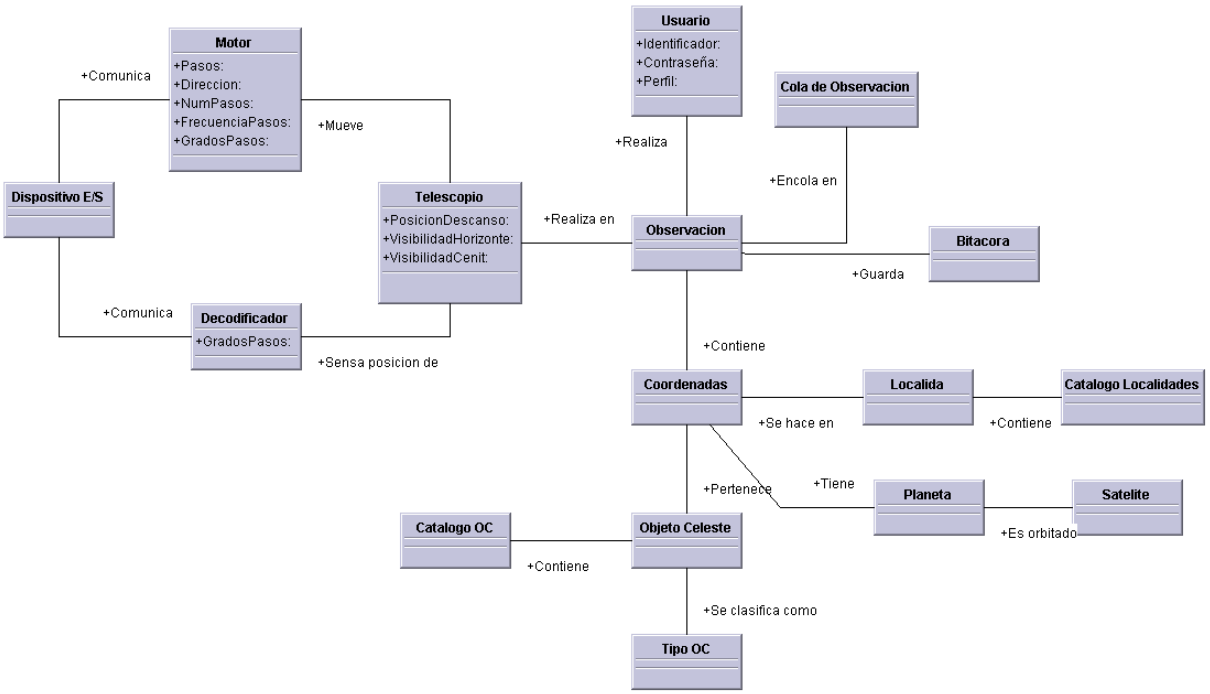


Fig. XXVII Diagrama conceptual del sistema en estudio.

[50] LARMAN, Craig. *UML y Patrones*... pag. 120

6.3.2 Diagrama de secuencia

El diagrama de secuencia de un sistema muestra gráficamente los eventos que fluyen de los actores al sistema[51].

Para nuestro caso, el diagrama de secuencia de un sistema describe, en el curso particular de los eventos de un caso de uso, los actores externos que interactúan directamente con el sistema (como caja negra) y con los eventos del sistema generados por los actores.

Para elaborar los diagramas de secuencia se identificaron a los actores que operaban directamente sobre el sistema y a partir del curso normal de los eventos del caso de uso, se identificaron los eventos externos del sistema que son generados por los actores.

Es de ésta manera que, basados en los casos de uso expandidos expuestos en éste mismo capítulo, desarrollamos sus respectivos diagramas de secuencia.

Para el caso de uso 6, *Obtener coordenadas relativas de objetos celeste NP*, tenemos los siguientes diagramas:

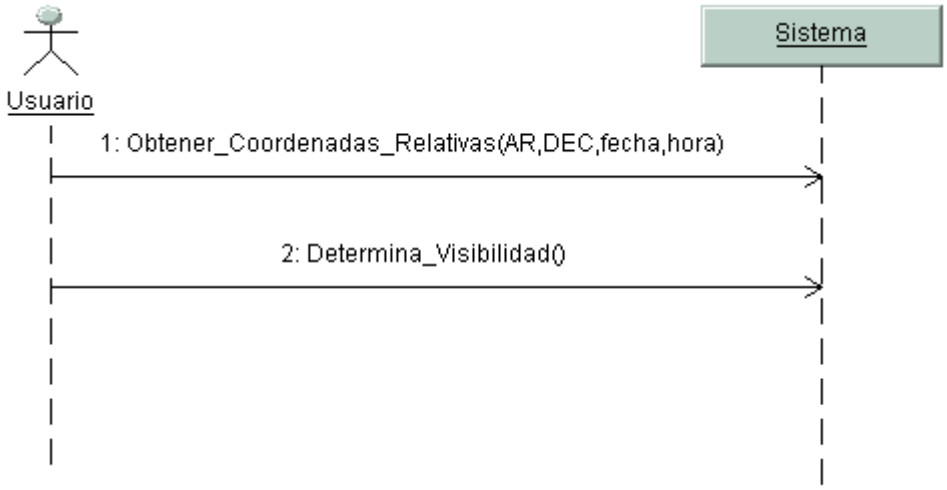


Fig. XXVIII Diagrama de secuencia 1 del caso de uso *Obtener coordenadas relativas de objetos celeste NP*

[51] LARMAN, Craig. *UML y Patrones*. Pág. 135

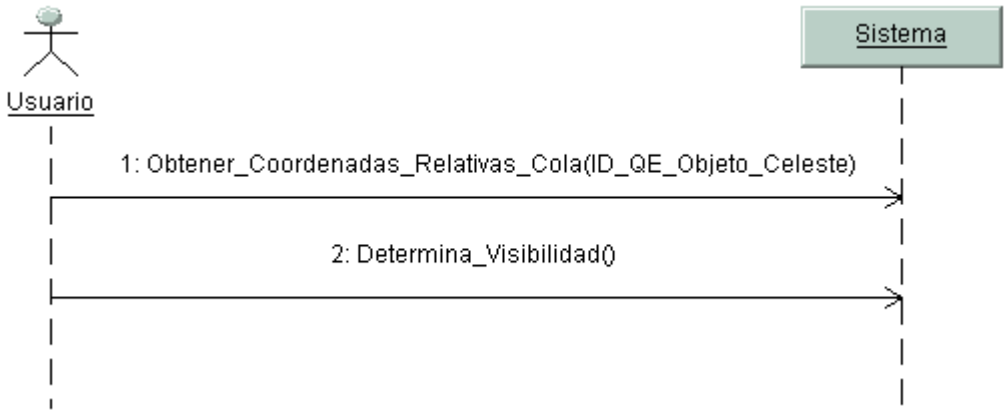


Fig. XXIX Diagrama de secuencia 2 del caso de uso *Obtener coordenadas relativas de objetos celeste NP*

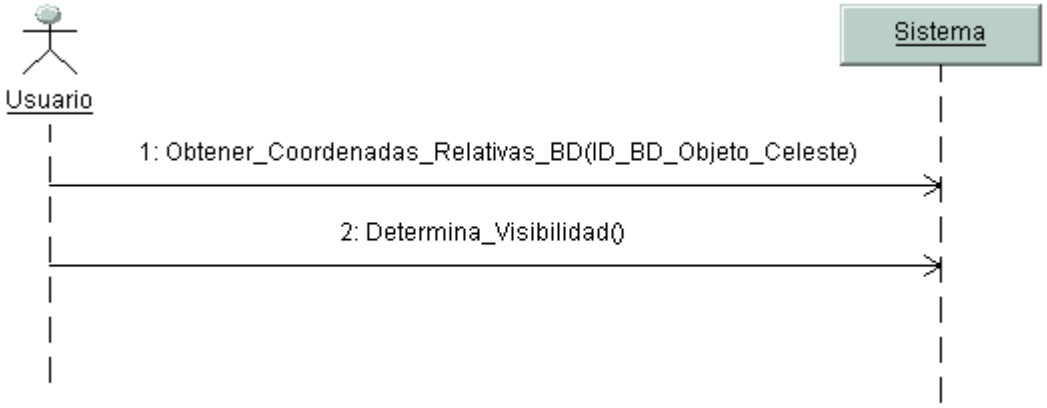


Fig. XXX Diagrama de secuencia 3 del caso de uso Obtener coordenadas relativas de objetos celeste NP

Para el caso de uso 7, *Mover Telescopio*:

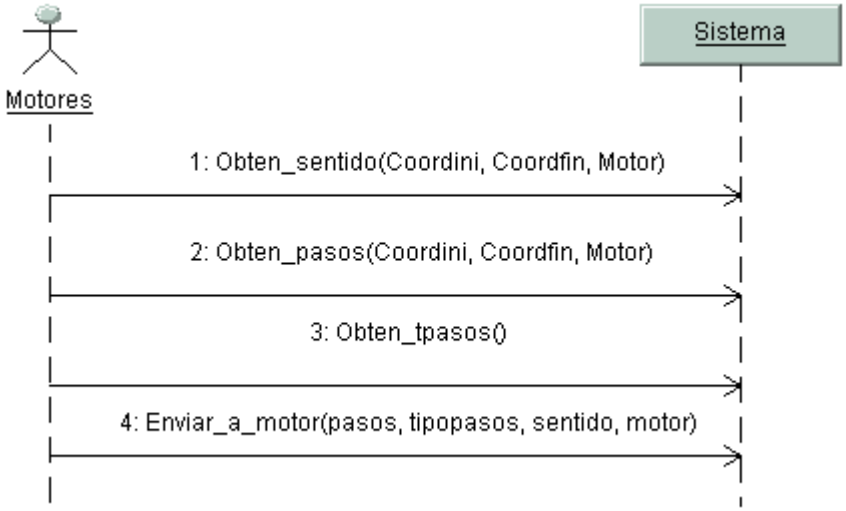


Fig. XXXI Diagrama de secuencia 1 del caso de uso *Mover Telescopio*.

Por último, para el caso de uso 8, *Girar ejes telescopio:*

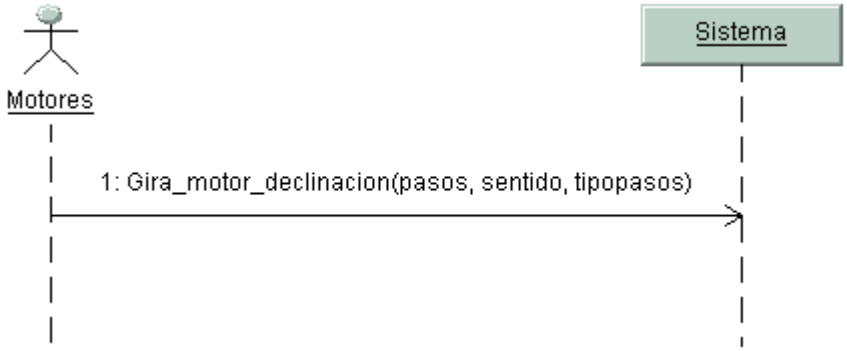


Fig. XXXII Diagrama de secuencia 1 del caso de uso *Girar ejes telescopio*.

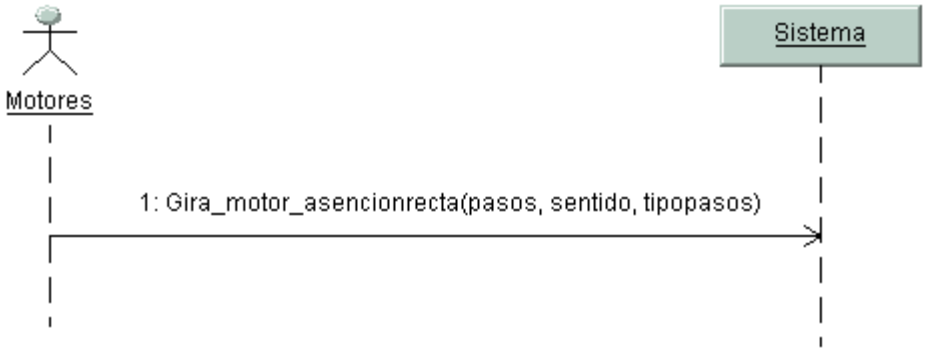


Fig. XXXIII Diagrama de secuencia 2 del caso de uso *Girar ejes telescopio.*

Los demás diagramas de secuencia pueden consultarse en el anexo B del presente trabajo.

6.3.3 Contratos de operación del sistema

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

En términos generales un contrato es un documento que describe lo que una operación se propone lograr. Suele redactarse en estilo declarativo, enfatizando lo que sucederá y no cómo se conseguirá. Los contratos suelen expresarse a partir de los cambios de estado de las precondiciones y de las poscondiciones. Se puede elaborar un contrato por un método de una clase de software o para una operación global del sistema[52].

[52] LARMAN, Craig. *UML y Patrones*. Pág. 147

La preparación de los contratos depende del desarrollo previo de los casos de uso, el modelo conceptual, los diagramas de secuencia de sistema y la identificación de sus operaciones.

Los casos de uso sugieren los diagramas de los eventos y de la secuencia del sistema.

Mediante los diagramas de secuencia se identifican las operaciones del sistema.

Por último el efecto de las operaciones del sistema se describe en los contratos.

A continuación se describen las secciones de un contrato, mediante el siguiente esquema:

Contrato	***IDENTIFICADOR***
Nombre	*Nombre de la operación y parámetros*
Responsabilidades	*Descripción informal de las responsabilidades que debe cumplir la operación.*
Tipo / Referencias cruzadas / Salidas	*Nombre del tipo/* *Números de referencia de las funciones del sistema, casos de uso, etc.* *No de salidas de la Interfaz de usuario; Por ejemplo, mensajes o registros que se envían fuera del sistema.*
Notas	*Notas de diseño, algoritmos e información afín.*
Excepciones	*Casos excepcionales.*
Precondiciones	*Suposiciones acerca del estado del sistema antes de ejecutar la operación.*
Poscondiciones	*El estado del sistema después de la operación.*

TABLA XXXV. Formato empleado para la descripción de contratos

El formato anterior fue tomado del libro de Larman, los autores agregaron al formato anterior el dato identificado como *Contrato*, el cual hace referencia a un identificador.

La forma en que se crea un identificador fue definida por el equipo de la siguiente manera:

CU###-DS###-OP###

Donde:

- CU: Caso de uso
- DS: Diagrama de secuencia
- OP: Operación
- #### identificador de 3 dígitos

De esta manera se puede rastrear de que operación, qué diagrama y qué caso de uso se deriva el contrato.

La misma codificación aplica para los casos de uso, y los diagramas de secuencia, omitiendo los diagramas de secuencia y las operaciones respectivamente.

A continuación se muestran, para fines ilustrativos, los contratos que resultaron del caso de uso 7: *Mover Telescopio*. Los demás contratos pueden ser consultados en el anexo B del presente trabajo.

Contrato	**CU007-DS001-OP001**
Nombre	Obten_sentido(Coordini: coord, Coordfin: coord. , Motor: class motor) De Instancia motor, se necesita la información de cual motor es y el sentido positivo configurado para ese motor. Coord: es AH y Dec
Responsabilidades	Obtiene el sentido de giro del movimiento del motor de acuerdo a la distancia mas corta entre la coordenada inicial y final.
Tipo / Referencias cruzadas / Salidas	
Notas	Se pasa la posición actual como inicial. Se supone que durante el calculo no sé esta moviendo el motor.
Excepciones	Coordenadas invalidas. Se detecta un movimiento al inicio del calculo y al final
Precondiciones	Las coordenadas finales deseadas están dentro del rango de movimiento Las coordenadas finales deseadas están dentro del rango de visibilidad Se ha configurado el sentido positivo del motor Se tiene la instancia del motor pedida Las coordenadas iniciales se conocen y es la posición actual al inicio del calculo. Las coordenadas finales se conocen y son las coordenadas del punto a donde se quiere mover el telescopio.
Poscondiciones	Se obtiene un valor positivo o negativo que indica el sentido del movimiento del motor.

Contrato	**CU007-DS001-OP002**
Nombre	Obten_tipopasos(Coordini: coords, Coordfin: coords, Motor: class motor) De *Instancia motor*, se necesita la información de cual motor es y el sentido positivo configurado para ese motor. Coord: es AH y Dec
Responsabilidades	Obtener el tipo de pasos, cortos o largos de acuerdo a las coordenadas iniciales y finales, tipo de motor y configuración de los tipos de pasos configurados sistema
Tipo / Referencias cruzadas / Salidas	
Notas	Se supone que durante el calculo no sé esta moviendo el motor.
Excepciones	Coordenadas invalidas.
Precondiciones	Se tiene la instancia del motor pedida Las coordenadas iniciales se conocen y es la posición actual al inicio del calculo. Las coordenadas finales se conocen y son las coordenadas del punto a donde se quiere mover el telescopio.
Poscondiciones	Se obtiene un valor de relación que indica el tipo de pasos.

Contrato	**CU007-DS001-OP003**
Nombre	Obten_pasos(Coordini: coords, Coordfin: coords, Motor: class motor, sentido, tipopasos) De *Instancia motor*, se necesita la información de cual motor es y el sentido positivo configurado para ese motor. Coord: es AH y Dec
Responsabilidades	Obtener el número de pasos de acuerdo al motor requerido, al tipo de pasos, sentido, las coordenadas iniciales y finales.
Tipo / Referencias cruzadas / Salidas	
Notas	Se supone que durante el calculo no sé esta moviendo el motor.
Excepciones	No se obtienen los pasos por valores inválidos
Precondiciones	Se tiene la instancia del motor pedida Las coordenadas iniciales se conocen y es la posición actual al inicio del calculo. Las coordenadas finales se conocen y son las coordenadas del punto a donde se quiere mover el telescopio.
Poscondiciones	Se obtiene un valor de número de pasos a desplazarse.

Contrato	**CU007-DS001-OP004**
Nombre	Enviar_a_motor(pasos, tipopasos, sentido, motor)
Responsabilidades	Enviar los datos para el movimiento del motor al dispositivo de comunicación con los motores
Tipo / Referencias cruzadas / Salidas	
Notas	Se usa el caso de uso 8 Se obtiene una notificación antes de enviar y después de enviar la información
Excepciones	Cualquier error reportado por el caso de uso.
Precondiciones	Las coordenadas finales deseadas están dentro del rango de movimiento Las coordenadas finales deseadas están dentro del rango de visibilidad
Poscondiciones	El resultado del caso de uso 8. Se reporta la posición actual del telescopio.

Los demás contratos se presentan en el Anexo B del presente trabajo.

6.3.4 Diagramas de colaboración

En los contratos se incluye una primera conjetura óptima sobre las poscondiciones referentes al inicio de las operaciones del sistema. Sin embargo, los contratos no muestran una forma de cómo los objetos de software van a cumplir con ellas.

Los diagramas de colaboración no se pueden realizar si no existen antes el diagrama conceptual y los contratos. Las demás dependencias se pueden inferir fácilmente.

Como vimos en el capítulo 3, los diagramas de colaboración junto con los diagramas de secuencia forman parte de lo que en UML se conocen como los diagramas de interacción, los cuales explican gráficamente las interacciones entre las clases.

Los diagramas de colaboración del presente trabajo se diseñaron a partir de los contratos. Por cada operación del sistema o contrato se dibujó un diagrama de colaboración, diseñando un sistema de objetos interactivos, usando como punto de partida las responsabilidades descritas en los contratos, las poscondiciones y la descripción de casos de uso. Aplicando los patrones explicados en el capítulo 3 como guías de diseño.

A continuación se muestran los diagramas de colaboración resultantes de los contratos anteriormente diseñados. Para el caso del contrato CU007-DS001-OP001**,** *Obten_sentido(Coordini: coord, Coordfin: coord., Motor: class motor)*. Se obtiene el siguiente diagrama de colaboración:

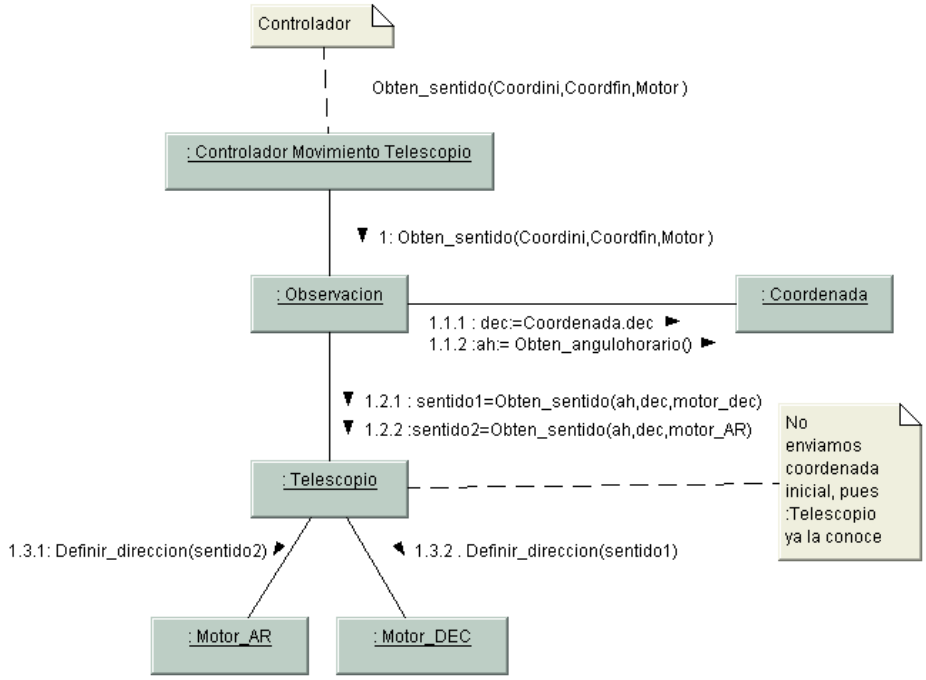


Fig. XXXIV Diagrama de colaboración según el contrato CU007-DS001-OP001

Para el caso del contrato CU007-DS001-OP002**,** *Obten_tipopasos(Coordini: coords, Coordfin: coords, Motor: class motor),* el diagrama resultante es el siguiente:

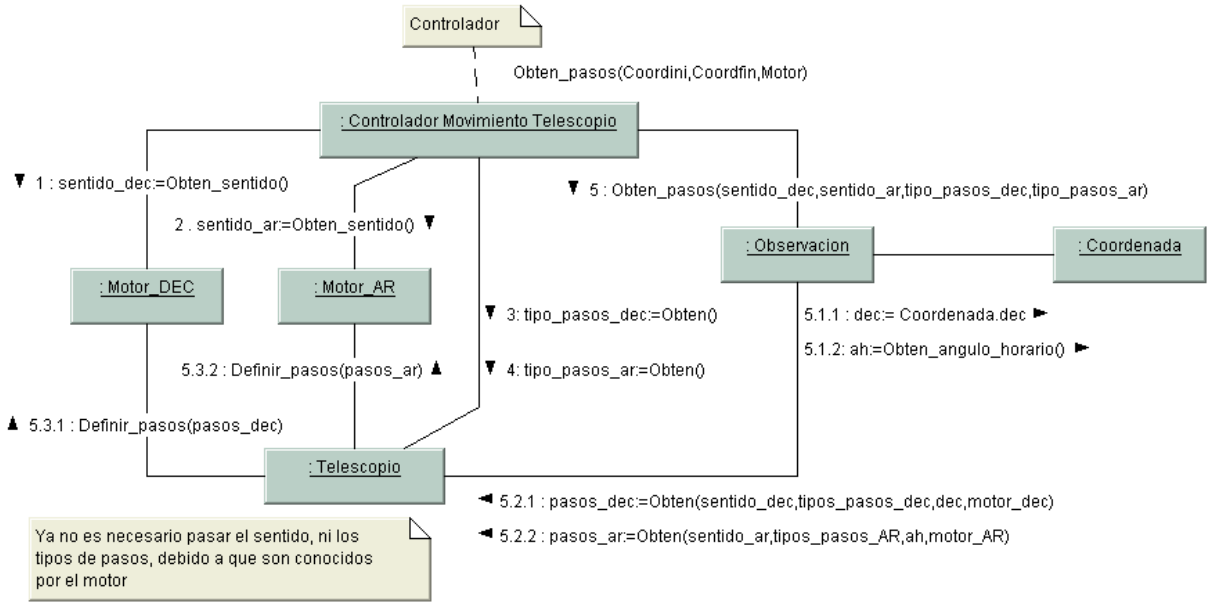


Fig. XXXV Diagrama de colaboración según el contrato CU007-DS001-OP002

A continuación se muestra el diagrama para el desarrollado a partir del contrato **CU007-DS001-OP003,** *Obten_pasos(Coordini: coords, Coordfin: coords, Motor: class motor, sentido, tipopasos):*

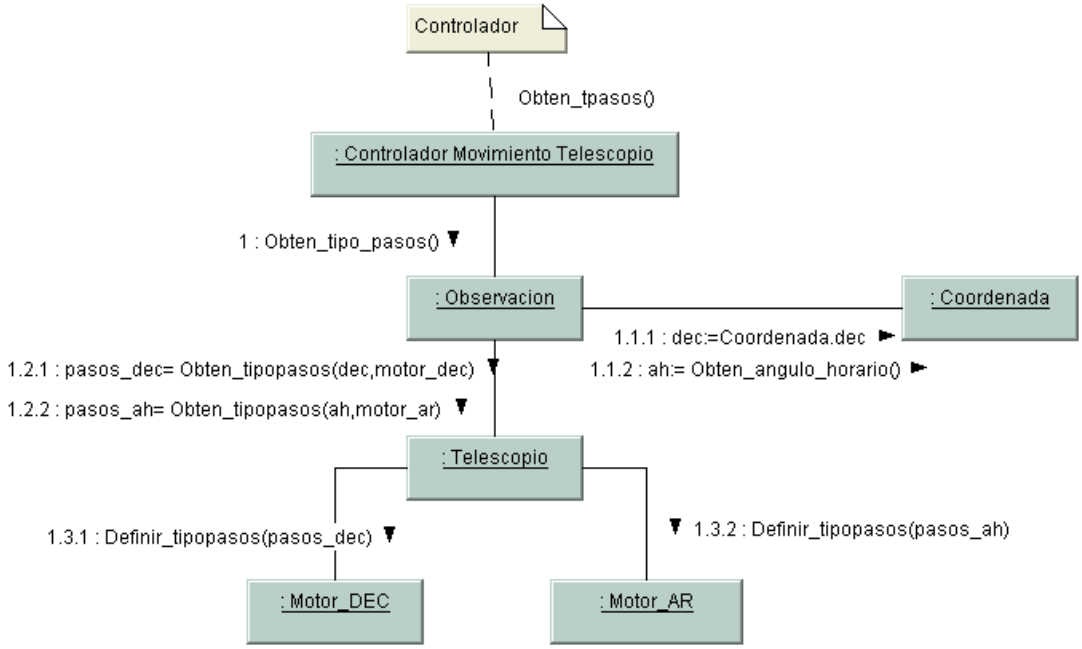


Fig. XXXVI Diagrama de colaboración según el contrato CU007-DS001-OP003

Por último, el diagrama de colaboración diseñado a partir del contrato CU007-DS001-OP004, *Enviar_a_motor(pasos, tipopasos, sentido, motor)*.

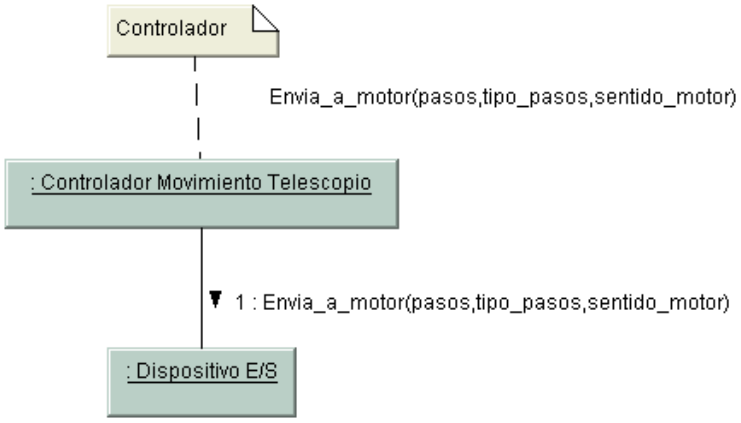


Fig. XXXVII Diagrama de colaboración según el contrato CU007-DS001-OP004

Los diagramas de colaboración faltantes pueden consultarse en el anexo B del presente trabajo.

6.3.5 Diagrama de clases

Al mismo tiempo que se van desarrollando los diagramas de colaboración, se comienzan a identificar, partiendo del diagrama conceptual y analizando los diagramas de interacción, la especificación de las clases de software (y las interfaces) que participarán en la solución, completándolas con detalles de diseño como lo son los métodos.

Los atributos para nuestros bosquejos de clases provienen del diagrama conceptual, y los métodos, de los diagramas de colaboración que desarrollamos. Debemos recordar que la navegabilidad de los diagramas nos indican a qué clase pertenecen.

Las asociaciones se construyen partiendo de la visibilidad entre los objetos, haciendo este análisis a partir de los diagramas de colaboración.

La visibilidad es la capacidad de un objeto para ver otro o hacer referencia a el. Existen 4 formas de que el objeto A consiga la visibilidad al objeto B:

1. Visibilidad por atributos: B es un atributo de A

2. Visibilidad por parámetros: B es un parámetro de A

3. Visibilidad declarada localmente: Se declara que B es un objeto local en un método de A

4. Visibilidad global: en alguna forma B es visible globalmente.

Para determinar el tipo de visibilidad, basta con analizar los diagramas de colaboración. Por ejemplo, de los diagramas anteriores podemos observar con relativa facilidad que la clase *Observación* es visible a ControladorMovimientoTelescopio debido a que es un atributo de ella.

El diagrama UML resultante de este trabajo es el diagrama de clases que se muestra a continuación:

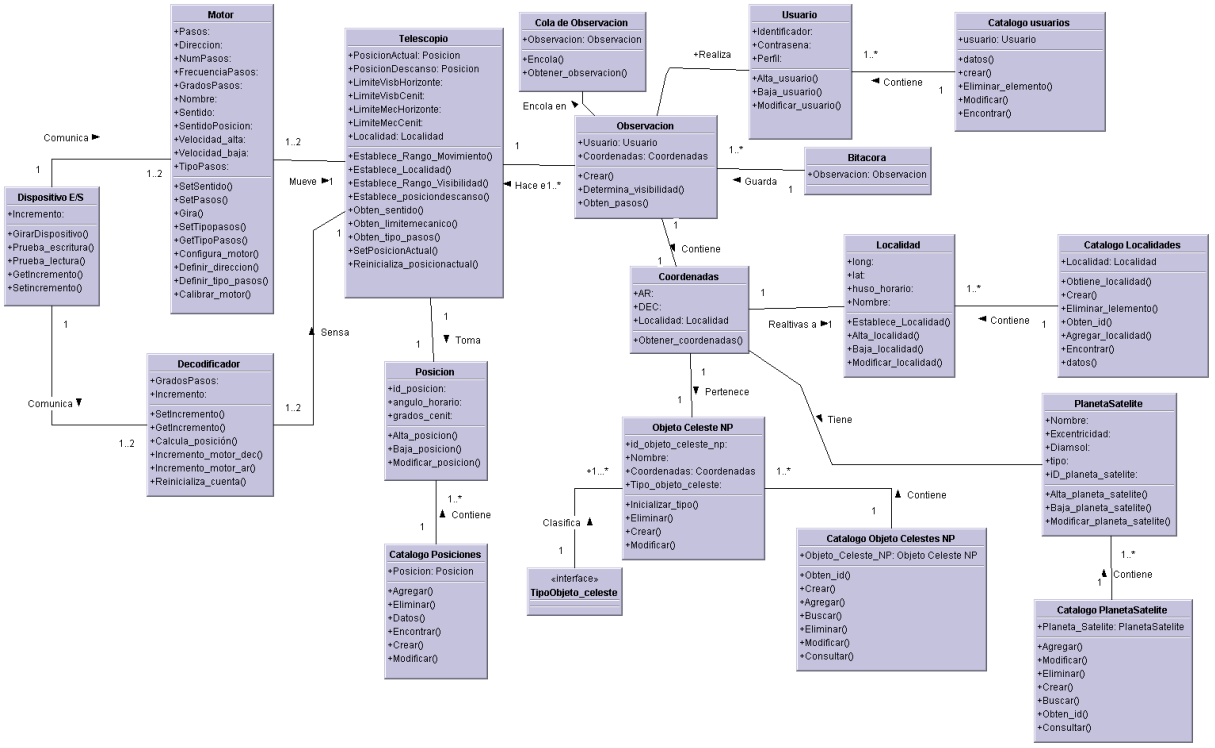


Fig. XXXVIII Diagrama de clases del sistema en estudio

A partir de éste diagrama podemos iniciar la codificación de los componentes de software, que serán parte de la solución de nuestro problema.

6.3.6 Codificación

Como hemos visto, durante la fase de construcción, hemos conseguido, hasta el momento, un diseño robusto que es independiente del lenguaje en el que se implemente, ya que hemos aplicado los conceptos del paradigma Orientado a objetos explicados en el capítulo 3 y que en cualquier lenguaje orientado a objetos, conociendo la sintaxis específica, pueden ser fácilmente codificados.

Para realizar la codificación es necesario comenzar la definición de la clase declarando los atributos y métodos que se muestran en el diagrama de clases. Y definiendo los métodos basados en los diagramas de colaboración y los contratos en los que se basaron.

El código resultante se puede consultar en el anexo C del presente trabajo, junto con el diagrama de clases definitivo.

6.3.7 Comentarios adicionales

El trabajo realizado hasta el momento, cumple con el objetivo de dar solución al problema de diseño de un programa de localización de objetos celestes, que se espera sea fácilmente adaptable y escalable debido a su diseño mediante técnicas orientadas a objetos. Sin embargo, el proceso expuesto durante este capitulo define los conceptos y comportamientos

básicos del sistema, es decir, nos centramos en los *objetos del dominio del problema*. Pero un sistema ordinario se conecta con una interfaz de usuario y un mecanismo de almacenamiento que sea persistente.

Como la finalidad de éste capítulo central era mostrar el proceso de desarrollo para lograr un diseño que cumpliese con el paradigma de orientación a objetos, para que las características de las unidades de implementación (clases) explicadas en el capítulo 3 nos permitieran cumplir con los criterios de Meyer para asegurar la modularidad del sistema y su fácil mantenimiento. No se explica el desarrollo de la Interfaz gráfica de usuario (GUI) ni del mecanismo de persistencia; Sin embargo, puede ser consultado en los anexos D y E del presente trabajo algunos ejemplos de éstos desarrollos.

Sin duda creemos que el hecho de plantear esta arquitectura de 3 capas (presentación, dominio y almacenamiento o persistencia) ayuda de nueva cuenta a asegurar la modularidad y la escalabilidad del sistema, ya que si llegara a cambiar la lógica de la aplicación (capa de dominio) no tendrían que modificarse la interfaz, ni los métodos de acceso, ya que son independientes. Además, podría, inclusive, cambiarse la interfaz con cierta facilidad para llevarse a otra plataforma (p.e. WEB) o cambiarse con relativa facilidad los métodos de persistencia para acceder a una base de datos distinta.

6.4 Diseño de Base de Datos

Se diseño una base de datos con un modelo Entidad / Relación y utilizando las reglas de normalización. Existían ya diversos archivos planos con la información de las coordenadas de los objetos celestes, sin embargo debido a nuestro diseño orientado a objetos se decidió no utilizar los archivos y crear una Base de Datos que se acoplara el diseño ya realizado.

Además de que en este diseño, se pretende englobar los principales catálogos astronómicos, como lo son el *Messier*, el NGC, el NBC y el IC. Y permitir realizar búsquedas más complejas, que faciliten la interacción con el usuario.

6.4.1 Diagrama Entidad Relación

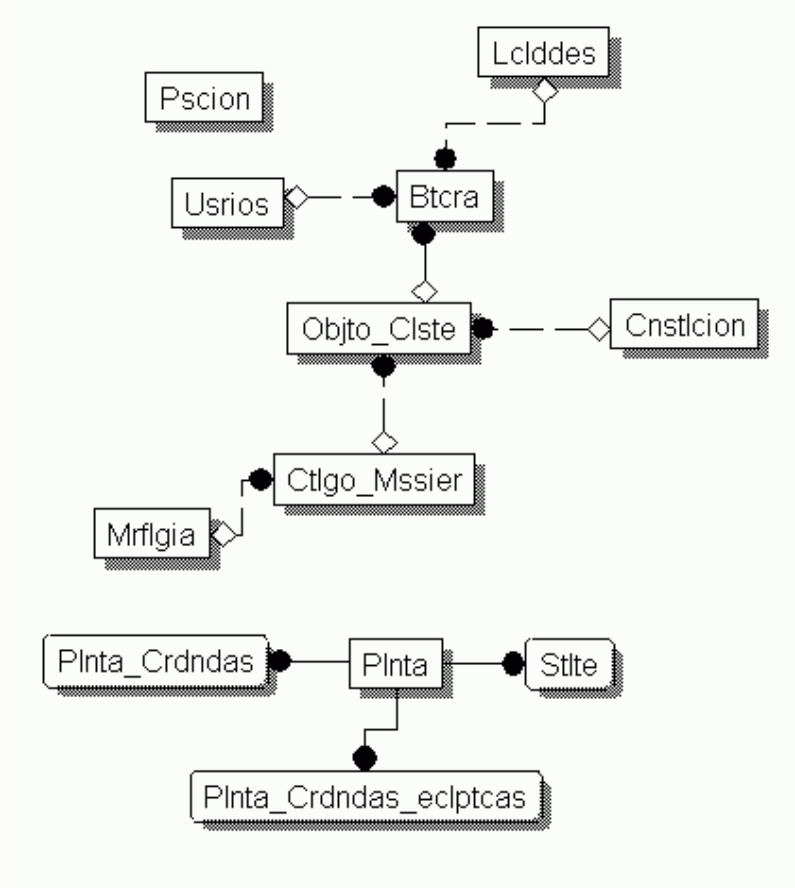


Fig. XXXIX Diagrama entidad – relación del sistema en estudio

A continuación describimos las tablas:

Nombre	Definición
Btcra	Tabla que guardara los objetos observados por el sistema
Cnstlcion	Conjunto de estrellas, cuya asociación esquemática y mítica, sirve para identificar una región de la esfera celeste.
Ctlgo_Mssier	El listado de objetos que al ser vistos con telescopios pequeño, son de aspecto difuso. Contiene cúmulos estelares, nebulosas y galaxias.
Lclddes	Datos de las diferentes localidades para observar en el mundo.
Mrflgia	Clasificación de la morfología de los objetos del catalogo Messier.
Objto_Clste	Es cualquier objeto que se encuentra en la bóveda celeste.
Plnta	Cuerpo celeste esférico de tamaño considerable de al menos 1000 Km. de diámetro. No emite luz propia.
Plnta_Crdndas	Coordenadas ecuatoriales d los planetas (incluimos al Sol y a la Luna).
Plnta_Crdndas_eclptcas	Coordenadas eclípticas de los planetas(incluimos al Sol y a la Luna)
Pscion	Tabla que Guarda las posiciones de configuración y de descanso del telescopio.
Stlte	Cuerpo que orbita alrededor de otro. Luna de un planeta.
Usrios	Es la persona que utilizara el sistema

TABLA XXXVI. Descripción de tablas de la Base de Datos

6.4.2 Diagrama Entidad/Relación con atributos

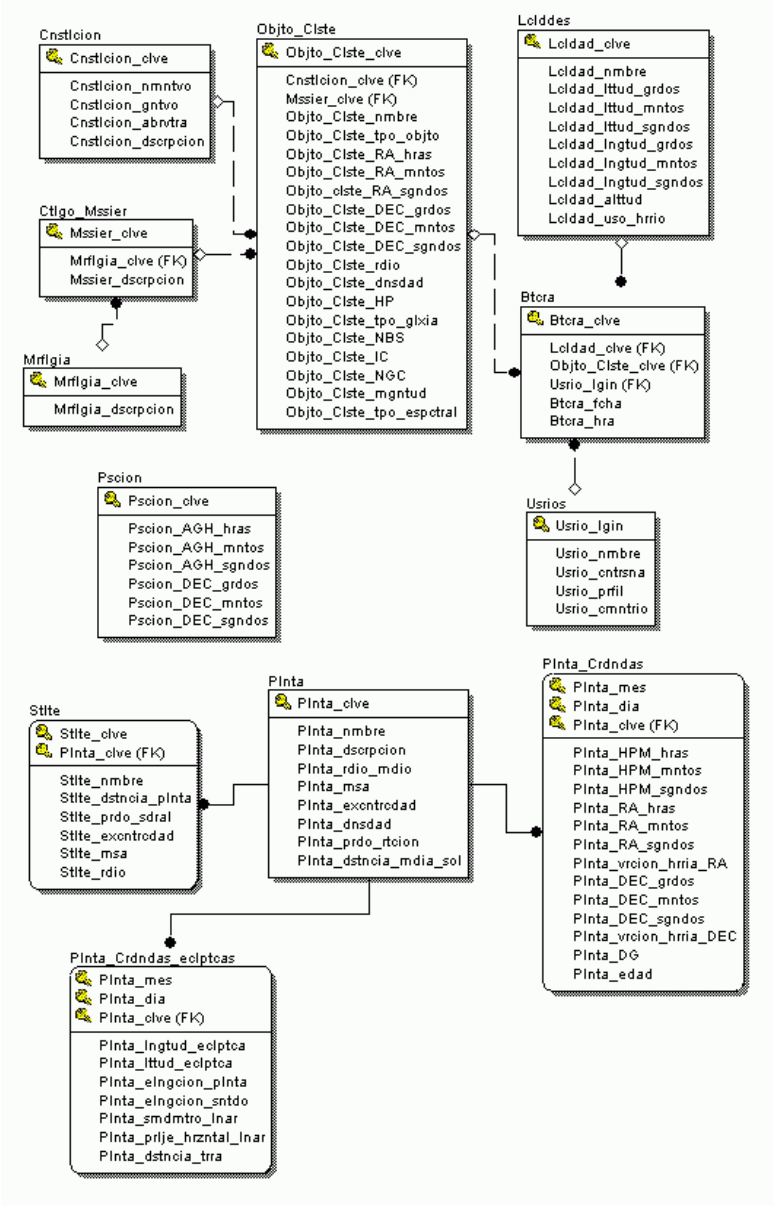


Fig. XL Diagrama entidad – relación (detalle)

Podemos ver mas adelante en el Anexo C, el diccionario de datos, así como los *scripst* de creación de la Base de Datos en Linux, utilizando el Manejador MySQL.

7 Conclusiones

Este es el último capítulo del presente trabajo de tesis, en donde revisaremos nuestro objetivo inicial, nuestra hipótesis de trabajo y los resultados obtenidos para poder emitir una conclusión al trabajo desarrollado.

7.1 Los riesgos de la migración hacia las tecnologías de orientación a objetos

Definitivamente en los últimos años hemos experimentado en la ingeniería de sistemas un *boom* hacia la orientación a objetos. Los métodos de análisis y diseño orientado a objetos que han evolucionado constantemente parecen, repentinamente, ser la solución a los problemas de tiempo, costo y calidad de los proyectos de construcción de programas para computadora; sin embargo, antes de tomar la decisión para migrar hacia estas tecnologías es necesario tomar en cuenta otros factores como:

- *La curva de aprendizaje de las nuevas metodologías*. Es un hecho comprobado que el adoptar una metodología orientada a objetos para el análisis y diseño es una oportunidad para lograr la construcción de sistemas más eficientes; sin embargo, al mismo tiempo representa un riesgo, un factor para el fracaso, ya que si no se tiene cuidado en definir una estrategia para la adopción de la metodología y se intenta de golpe una transición total, podemos alargar los tiempos de análisis y diseño, olvidar el enfoque iterativo, crear confusión y frustración entre los miembros del equipo y, finalmente, el abandono total del proyecto.

  Durante el desarrollo del presente trabajo de tesis, el interés de los miembros del equipo por entender y manejar estas metodologías fue el principal motivador para lidiar con los problemas antes descritos. Pero si proyectáramos las experiencias negativas de la migración hacia las nuevas tecnologías a un ambiente laboral el riesgo seguramente se vería amplificado debido a la tendencia natural de resistencia al cambio.

  ¿Qué acciones podríamos seguir para mitigar el riesgo implícito de la migración hacia las metodologías orientadas a objetos? A continuación presentamos algunas ideas del equipo de trabajo producto de la experiencia adquirida en este trabajo de tesis:

  - Práctica de la metodología que se desea incorporar en proyectos piloto de menor impacto.

  - Tener cuidado de utilizar siempre la regla de la identificación de 7 ± 2 conceptos, que permite una aproximación rápida al objeto de estudio y promueve el desarrollo iterativo.

  - Guiarse en la parte de diseño por patrones, pero no coleccionar una cantidad grande de los mismos en el primer proyecto, sino seleccionar 5 ó 7 básicos.

  - No ser rigoristas. Cada persona conceptualiza y entiende de manera distinta, si nuestra preocupación es la aplicación *al pie de la letra* de la metodología seleccionada, seguramente desviaremos nuestra atención a complacer al autor en turno y no en la aplicación de las ideas básicas de la orientación a objetos. Para nuestro caso nos funcionó muy bien no aplicar una metodología específica sino utilizar las mejores prácticas.

- *La selección del lenguaje de programación*. Si bien el 70% del esfuerzo se concentra en el análisis y en el diseño de la solución, no debemos olvidar que el verdadero entregable de cualquier proyecto de ingeniería de programación es el código ejecutable, y que el lenguaje seleccionado para la implementación impacta directamente en éste. Un lenguaje que nos facilite la implementación de los conceptos básicos de orientación a objetos de una manera transparente y rápida puede ser factor de éxito en la migración hacia las tecnologías orientadas a objetos. A continuación presentamos algunas recomendaciones:

    - Seleccionar el lenguaje que utilizaremos pensando no solamente en su penetración en el mercado sino también en las características que lo identifican y en cómo estas características aportarían valor a nuestra aplicación o a nuestro entrenamiento.

    - Si el equipo tiene experiencia comprobada sobre algún lenguaje de programación que no es orientado a objetos, puede realizarse la combinación de análisis y diseño orientado a objetos con la construcción en un lenguaje que no lo es. Tal vez los resultados en cuanto a reutilización y mantenimiento no sean los esperados, pero la experiencia en el manejo del lenguaje ayudará a aterrizar de manera práctica y rápida los conceptos de objetos.

Consideramos que la Facultad de Ingeniería, a través de la División de Ingeniería Eléctrica y el Departamento de Ingeniería en Computación podría ayudar a la migración hacia estas tecnologías de la siguiente manera:

- Enseñando desde los primeros semestres la programación en algún lenguaje orientado a objetos y la implementación de los patrones más utilizados en estos lenguajes.

- Utilizando en los semestres avanzados el enfoque orientado a objetos para los proyectos de sistemas operativos, bases de datos, etc.

- Utilizando el UML como herramienta de modelado no sólo para programación sino para diseño y construcción de sistemas digitales.

- Promoviendo la reutilización para la construcción de proyectos de ingeniería de software.

- Creado un repositorio de objetos para poder ser utilizado por los alumnos de Ingeniería en Computación.

7.2 Las tecnologías seleccionadas y su contribución a la adaptabilidad y la escalabilidad del sistema diseñado.

En este punto no podíamos dejar pasar de largo la revisión y la crítica a nuestra hipótesis de trabajo. ¿Realmente la utilización de las metodologías de AOO y DOO, así como el UML y la elección de Pyhton ayudaron a aumentar la modularidad y adaptabilidad del sistema?

Definitivamente la contribución de las metodologías de AOO y DOO no está en discusión, ya que la aplicación del concepto fundamental (el objeto) asegura la modularización y al mismo tiempo el relativamente fácil mantenimiento del sistema, ya que el objeto forma la unidad mínima funcional y sintáctica en la cual recae la responsabilidad de una parte muy específica del trabajo global. De esta manera, un objeto o un conjunto de éstos nos dan la pauta para la construcción de nuevas funcionalidades o mejoras de las mismas sin preocuparnos de cómo está implementado el sistema en su totalidad.

Sin embargo, hay algunas consideraciones que debemos tomar en cuenta:

- Las guías de diseño de alta cohesión y bajo acoplamiento deben encontrar su justo medio de acuerdo a la especialización de nuestro sistema, ya que uno impacta directamente sobre el otro. Es decir, en la medida que una clase sea responsable de una cierta funcionalidad (cohesión) su interrelación con otras clases se verá afectada (acoplamiento). ¿Por qué? Porque si queremos que la clase mantenga un nivel menor de complejidad, entonces deberemos diseminar su funcionalidad en otras clases y por lo tanto su acoplamiento se verá afectado. Entonces, ¿Con base en qué debemos tomar la decisión de tendencia hacia cohesión o hacia acoplamiento? Esto dependerá de qué tan especializado queremos que sea nuestro sistema, si sabemos que nuestro sistema es muy especializado y que difícilmente podremos reutilizar los componentes construidos, tal vez debamos pensaren una cohesión menor. En cambio si nuestro sistema no es tan especializado y deseamos poder utilizar nuestros componentes en otros sistemas similares deberíamos pensar en una alta cohesión.

- El UML al ser realmente una herramienta no ayuda directamente a elaborar un mejor diseño, sin embargo su aportación es significativa para ayudar al entendimiento del problema (análisis) y por consiguiente en plantear su solución (diseño). Además lo intuitivos que resultan algunos de sus diagramas (sobre todo los de casos de uso y los de secuencia) aseguran que el “plano” del sistema sea fácilmente entendido y por lo tanto su mantenimiento sea más fácil.

- La elección de un lenguaje interpretado para fines didácticos resulta, al parecer, adecuada, ya que pueden probarse de manera inmediata conceptos de OO, relaciones entre componentes y funcionalidades nuevas, sin la necesidad de pasar por la compilación y por los errores que muy seguramente traerá consigo, sobre todo para personas que no están familiarizadas con la programación y la declaración de variables.

7.3 Comentario final

Por último, queremos mencionar que el presente trabajo de tesis puede ser utilizado para la implementación del sistema completo de posicionamiento o su extensión a nuevos telescopios o nuevas funcionalidades, así como guía para la aplicación del proceso iterativo y los artefactos UML más utilizados, así como la aplicación de los patrones más importantes. Las conclusiones derivadas del trabajo son lecciones aprendidas durante el desarrollo que, en algunas ocasiones, concuerdan con las recomendaciones de algunos autores como KRUTCHEN y ROYCE.

Trabajos futuros podrían enfocarse a la aplicación de métricas para estimar matemáticamente la modularidad y la adaptabilidad; en la aplicación de más patrones de diseño o incluso la creación de los mismos para sistemas de posicionamiento, o en la extensión del trabajo presentado para sistemas de posicionamiento más especializados.

8 Referencias

8.1 Bibliografía

LARMAN, Craig. *UML y Patrones Introducción al análisis y diseño orientado a objetos*. Traducido al castellano por Luz María Hernández Rodríguez, con revisión técnica de Humberto Cárdenas Anaya. 1ra edición. México, Prentice Hall, 1999. 507 p. Ils.

SIGFRIED, Stefan. *Understanding object-oriented software engineering*. NY, EUA; IEEE Press & IEEE Computer Society Press, 1996.

OUSTERHOUT, John K. *Scripting: Higher-Level Programming for the 21st Century.* Computer Magazine. Publicada por la IEEE. Marzo. 1998. p. 23-29. Ils.

KRUCHTEN, Philippe. *The Rational Unified Process an Introduction*. 2nd edition. EUA, Addison-Wesley, 2001. 298 p. Ils. The Addison-Wesley Object Technology Series.

FOWLER, Martín y SCOTT, Kendal. *UML Gota a Gota*. Traducido al castellano por Jaime González V y con revisión técnica de Dr. Gabriel Guerrero. 1ª edición. México, Addison Wesley, 1999. 224 p. Ils.

SUN MICROSYSTEMS INC. *Migrating to OO Programming with Java Technology. Student Guide.* California, EUA, Sun Microsystems Inc., 2000.

GRAHAM, Ian. *Object Oriented Methods*. 2ª edición. EUA, Addison Wesley, 1994. 473 p.

MARTÍN, James & ODELL, J. James. *Análisis y diseño orientado a objetos.* Traducción por Oscar Alfredo Palmas Velasco y revisado por Dr. Gabriel Guerrero. 1ª edición. México, Prentice Hall Iberoamericana, 1994. 545 p. Ils.

TRUEBLOOD, Mark & GENET, Russell. Microcomputer control of telescopes. EUA, Willmann-Bell, Inc., 1985, 377 p. Ils.

DUFFER-SMITH, Peter. *Astronomy with your personal computer*. 1a edición. Inglaterra, Cambridge University Press, 1985. 256 p.

GALLO, Joaquín y ANFOSSI, Agustín. *Curso de cosmografía*. 8ª. Edición. México, Editorial Progreso. P. Ills.

VIVES, Teodoro J. *Astronomía de posición, espacio y tiempo*. México, Editorial Alambra, 1971.

MALACARA, Daniel y MALACARA, Juan Manuel. *Telescopios y estrellas*. 2ª edición. México, Fondo de Cultura Económica, 1998. 175 p. Ills. Colección la ciencia para todos.

VAN ROSSUM, Guido. *Python tutorial release 1.5.1.* EUA, Corporation for National Research Initiatives (CNRI), 1998. 56 p.

VAN ROSSUM, Guido. *Python Library Reference 1.5.1.* EUA, Corporation for National Research Initiatives (CNRI), 1998. 257 p.

BROWN, Martin, C. *Python*. EUA, Osborne/McGraw-Hill, 2000, 722 p.

LUNDTH, Fredrik. *An introduction to Tkinter*. Editado por el autor. 1999. 164 p.

8.2 Sitios en la Internet

http://www.iis.ee.ic.ac.uk/~frank/surp00/article1/sb398/. Marzo, 2004

http://www.csee.umbc.edu/courses/undergraduate/CMSC202/Spring99/abaumg1/lectures/OOAD/. Marzo, 2004

http://www.uml.org. Marzo, 2004

http://www.omg.org. Marzo, 2004

http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci516819,00.html. Marzo, 2004

http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci212709,00.html. Marzo, 2004

http://searchexchange.techtarget.com/sDefinition/0,,sid43_gci213778,00.html. Marzo, 2004

http://www.webopedia.com/TERM/M/MySQL.html. Marzo, 2004

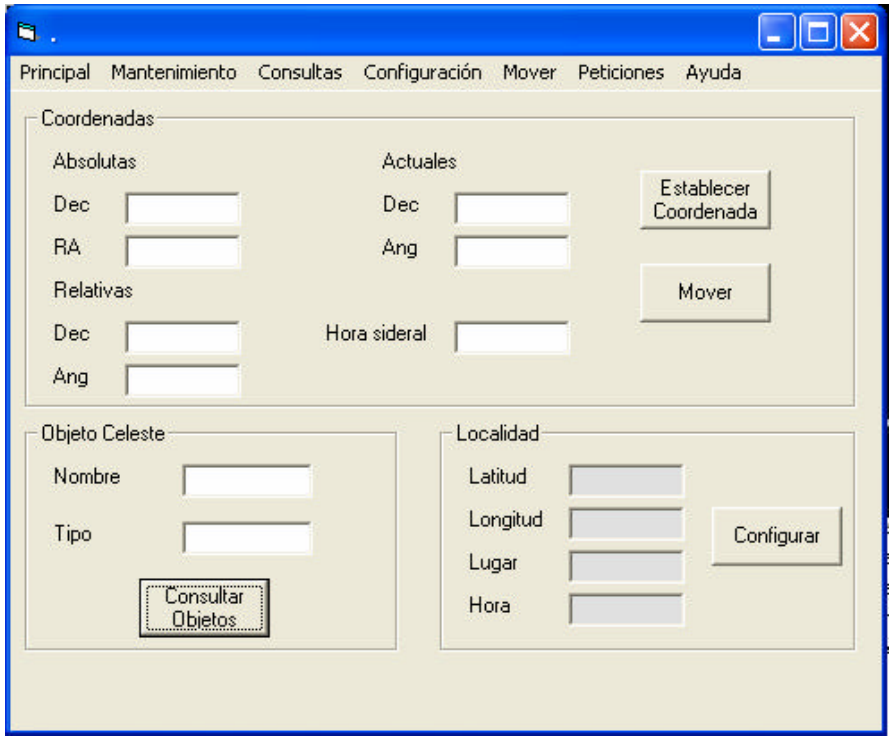
http://www.gnu.org/gnu/thegnuproject.es.html. Marzo, 2004

http://www.mysql.com/documentation/mysql/bychapter/manual_Introduction.html#History. Marzo, 2004

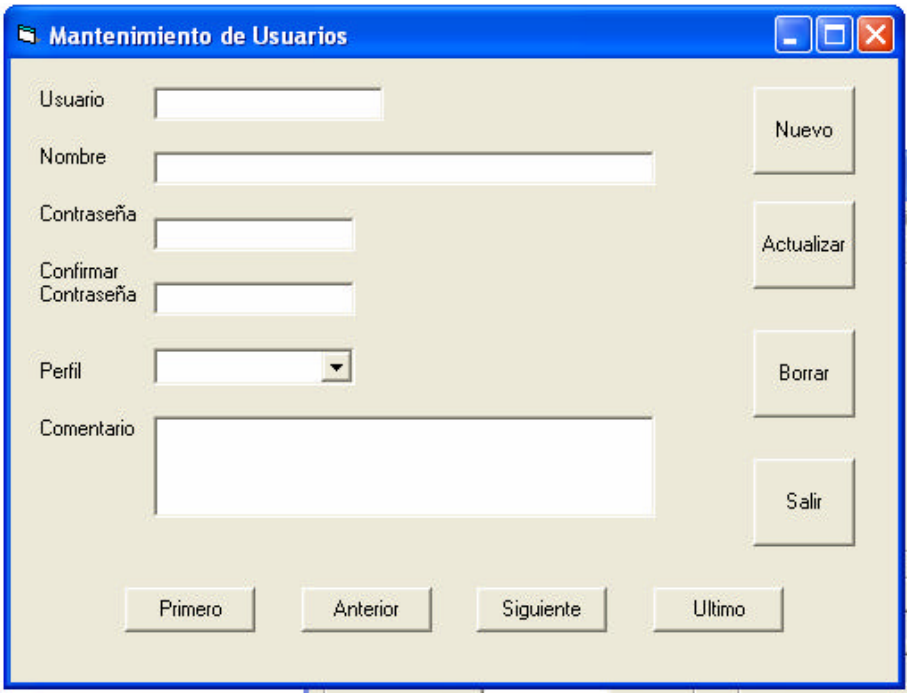
http://members.aol.com/acockburn.

http://www.python.org

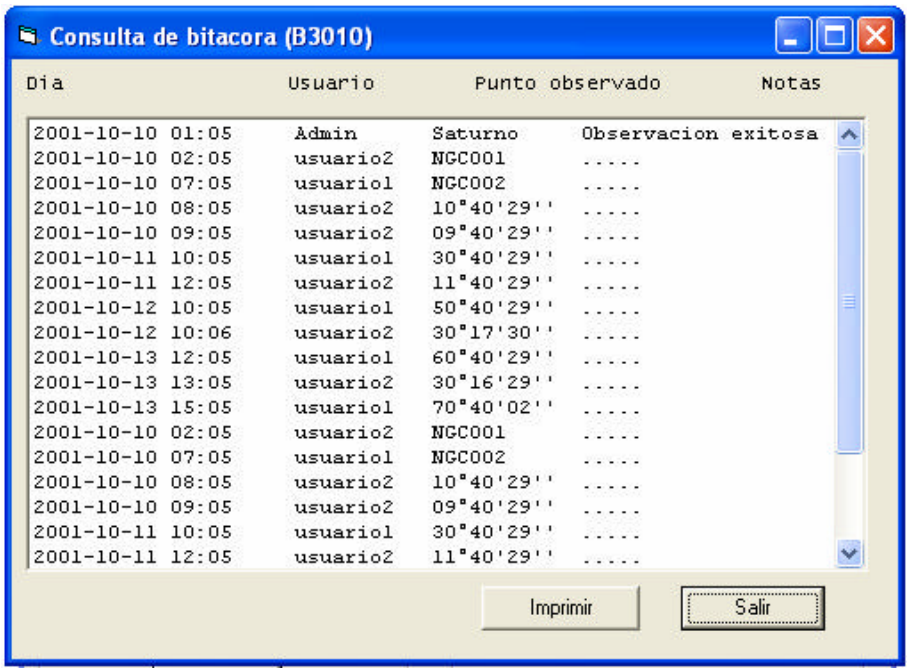
Anexo A: Prototipo



Pantalla Principal

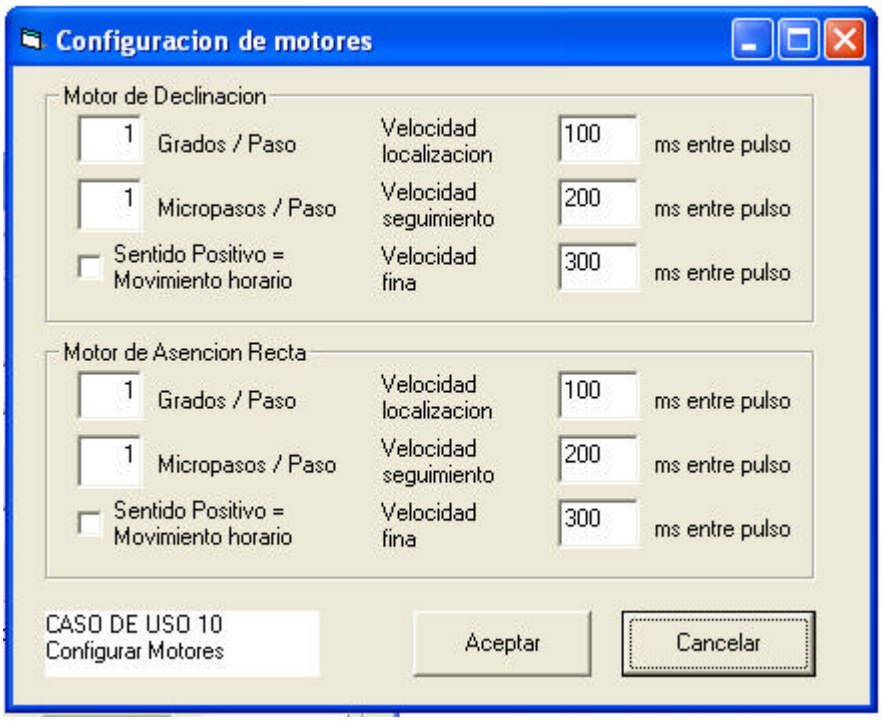


Pantalla de mantenimiento a usuarios

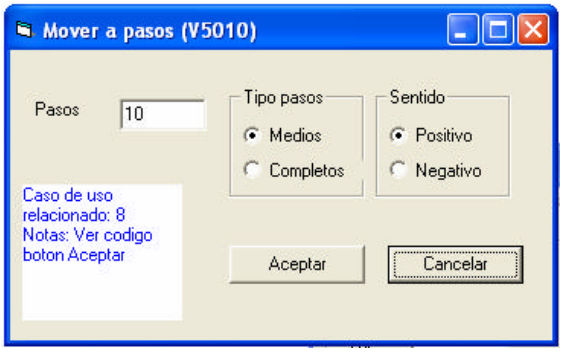


Pantalla de bitácora de observación

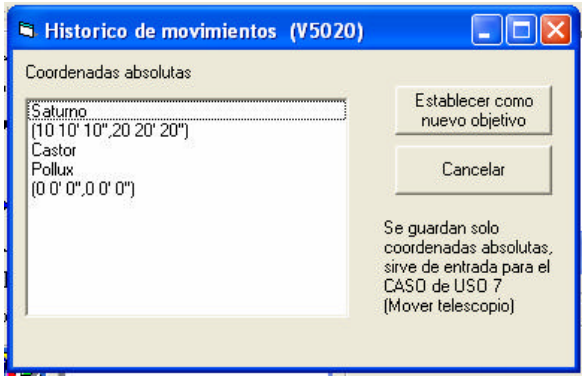
Pantalla de configuración de sistema



Pantalla de configuración de motores



Pantalla para mover telescopio a pasos



Pantalla para mover telescopio

Anexo B: Diagramas UML

Casos de Uso Expandidos

Caso de Uso 1	ADMINISTRACIÓN DE USUARIOS		
Tipo	Secundario, Esencial		
Objetivo	El administrador necesita llevar un control de los usuarios que utilizan el sistema.		
Referencias	R7		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Ninguna		
Condición para fin exitoso	El administrador logra mantener actualizado el catálogo de usuarios.		
Condición para fin erróneo	El administrador no logra mantener actualizado el catálogo de usuarios.		
Actores primarios y secundarios	Administrador.		
Disparador	El Administrador elige alguna de las opciones de administración.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador elige algunas de las opciones de administración.	
	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3		Se despliega en pantalla el menú de opciones: Alta usuario, Baja de usuario o modificación.
	4	El administrador selecciona alguna de las opciones	
	5		Se despliega la pantalla de la opción elegida
	6	El administrador realiza la operación que requiere.	
Extensiones	**Paso**	**Acción de la ramificación**	
	2	En caso de que no tenga atributos de administrador, se le notifica que no puede realizar la operación. Y termina el caso de uso.	
Variaciones	**Paso**	**Acción de la ramificación**	
	4	El administrador elige dar de alta a un usuario y le asigna un identificador, contraseña y permisos de	

		uso.
	4	El administrador elige la opción de dar de baja a un usuario e indica el identificador del usuario que será eliminado.
	4	El Administrador selecciona modificar un usuario, indica el identificador del usuario y modifica los atributos.

Caso de Uso 2	CONFIGURACIÓN DEL SISTEMA		
Tipo	Secundario, Esencial		
Objetivo	El administrador establece los parámetros iniciales de longitud, latitud, posición de descanso y rango de visibilidad del telescopio.		
Referencias	R7		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Ninguna		
Condición para fin exitoso	Se registran todos los parámetros correctamente.		
Condición para fin erróneo	No registran todos los parámetros correctamente.		
Actores primarios y secundarios	Administrador.		
Disparador	Solicitud de configurar sistema.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador elige algunas de las opciones de administración.	
	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3		Se despliega en pantalla el menú de Configuración del Sistema
	4	El administrador selecciona alguna de las opciones	
	5		Se despliega la pantalla de la opción elegida
	6	El usuario realiza la operación que requiere.	
Extensiones	**Paso**	**Acción de la ramificación**	
	2	En caso de que no tenga atributos de administrador,	

		se le notifica que no puede realizar la operación. Y termina el caso de uso.
Variaciones	**Paso**	**Acción de la ramificación**
	4	El administrador elige registrar la localidad o punto geográfico de observación (Longitud y latitud).
	4	El administrador elige establecer las coordenadas relativas de las diferentes posiciones de descanso del telescopio.
	4	El administrador elige registrar el rango de visibilidad de la cúpula(ángulos).

Caso de Uso 3	MANTENIMIENTO AL CATÁLOGO DE OBJETOS CELESTES NP		
Tipo	Secundario, Esencial		
Objetivo	El administrador necesita mantener actualizado el catálogo de los objetos celestes np a observar.		
Referencias	R7		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Ninguna		
Condición para fin exitoso	El administrador logra mantener actualizado el catálogo de objetos celestes.		
Condición para fin erróneo	El administrador NO logra mantener actualizado el catálogo de objetos celestes.		
Actores primarios y secundarios	Administrador.		
Disparador	El Administrador elige alguna de las opciones de administración.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador elige algunas de las opciones de administración.	
	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3		Se despliega en pantalla el menú de opciones: Alta Objeto Celeste NP, Baja de Objeto Celeste NP o modificación.
	4	El administrador selecciona alguna de las opciones	
	5		Se despliega la pantalla de la opción elegida

	6	El administrador realiza la operación que requiere.	
Extensiones	**Paso**	**Acción de la ramificación**	
	2	En caso de que no tenga atributos de administrador, se le notifica que no puede realizar la operación. Y termina el caso de uso.	
Variaciones	**Paso**	**Acción de la ramificación**	
	4	El administrador elige dar de alta un objeto celeste np dando todos sus atributos y de acuerdo al tipo de objeto celeste np, ejemplo: nombre, constelación, etc. El identificador se asigna automáticamente y esta compuesto también por el tipo de objeto celeste np.	
	4	El administrador elige la opción de dar de baja un objeto celeste np indicando el identificador del objeto celeste np y su tipo de objeto celeste np que será eliminado.	
	4	El Administrador selecciona modificar un objeto celeste np, indica el identificador del objeto celeste np y el tipo del objeto celeste np y modifica los atributos.	

Caso de Uso 4	CONSULTAR BITÁCORA DE OBSERVACIÓN		
Tipo	Secundario,		
Objetivo	El administrador necesita conocer que personas realizaron observaciones para llevar un control de ellas.		
Referencias			
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que estén creados los usuarios, Que estén creados los dispositivos de almacenamiento.		
Condición para fin exitoso	El administrador obtiene la información deseada de la bitácora.		
Condición para fin erróneo	El administrador NO obtiene la información deseada de la bitácora.		
Actores primarios y secundarios	Administrador.		
Disparador	El Administrador elige las opciones de consulta de bitácora		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador la opción de consulta de bitácora.	

	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3		Se despliega en pantalla el menú de criterios de consulta en la bitácora
	4	El administrador selecciona alguna de las opciones	
	5		Se despliega la pantalla el resultado de la consulta elegida
Extensiones	**Paso**	**Acción de la ramificación**	
	2	En caso de que no tenga atributos de administrador, se le notifica que no puede realizar la operación. Y termina el caso de uso.	
Variaciones	**Paso**	**Acción de la ramificación**	
	4	El administrador elige de entre las opciones de consulta por usuario	
	4	El administrador elige de entre las opciones de consulta por día.	
	4	El administrador elige de entre las opciones de consulta por objeto celeste np.	
	4	El administrador elige de entre las opciones de consulta por tipo de objeto celeste np.	
	4	El administrador elige de entre las opciones de consulta por planeta o satélite.	

Caso de Uso 5	CONSULTAR TIPOS DE OBJETOS CELESTES NP
Tipo	Secundario, esencial
Objetivo	El usuario consulta los atributos de un objeto celeste np en el catálogo de acuerdo al tipo de objeto celeste np al que pertenece.
Referencias	
Alcance y Nivel	Sistema, 2
Pre-condiciones	Que estén creados los usuarios. Que los catálogos de objetos celestes np tengan información. Que el tipo de objeto celeste np exista.
Condición para fin exitoso	El usuario logra consultar información de los catálogos.
Condición para fin erróneo	El usuario NO obtiene información de los catálogos. (Por error, o por que el objeto celeste np no esta en catalogo o porque el objeto celeste np no pertenezca a ningún tipo de objeto celeste np).
Actores	Usuario

primarios y secundarios			
Disparador	El usuario elige la opción de consulta de tipos de objetos celestes np.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el usuario elige la opción de consulta de tipo de objetos celestes np.	
	2		Se despliega en pantalla los tipos de objetos celestes np, por los que se puede hacer la búsqueda.
	3	El usuario selecciona algún tipo de objeto celeste np de las opciones.	
	4		Se despliega la pantalla el resultado de la consulta elegida
	5	El usuario de acuerdo a la selección de tipo de objeto celeste np, hace la búsqueda del objeto celeste np dando su identificador.	
	6		Se despliegan los resultados de la búsqueda de objeto celeste np
Extensiones	**Paso**	**Acción de la ramificación**	
	4	En caso de no existir el tipo de objeto celeste np, el sistema lo informa al usuario.	
	5	En caso de no existir el de objeto celeste np, el sistema lo informa al usuario.	
Variaciones	**Paso**	**Acción de la ramificación**	
	3	El usuario navega por el catálogo de tipo de objetos celestes np, ejemplo: galaxia, estrella, constelación, nebulosa, catálogo especial (messier, NGC, NBC).	

Caso de Uso 9	INFORMAR LA POSICIÓN ACTUAL DEL TELESCOPIO.		
Tipo	Primario, esencial		
Objetivo	Los sensores deben reportar la posición del telescopio cuando haya cualquier cambio de posición.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Que exista comunicación entre los sensores y el sistema.		
Condición para fin exitoso	Que los sensores reporten cualquier cambio en la posición del telescopio.		
Condición para fin erróneo	Que los sensores NO reporten cualquier cambio en la posición del telescopio.		
Actores primarios y secundarios	Sensores		
Disparador	Que el telescopio tenga un cambio de posición.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Los sensores están en espera de un cambio de posición y cuando ocurre envían el incremento en el cambio de la posición y la dirección.	
	2		El sistema recibe el incremento reportado por los sensores y actualizan la posición que tenían registrada anteriormente para que sea la misma que tiene el telescopio actualmente
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	
	1	Si no hay cambio en la posición no reportan nada.	

Caso de Uso 10	CONFIGURACIÓN DE MOTORES
Tipo	Primario, esencial
Objetivo	El administrador establece los parámetros de velocidad, la longitud de los pasos (pasos completos o medios) y la dirección.
Referencias	???
Alcance y Nivel	Sistema, 2
Pre-condiciones	Que exista un lugar de almacenamiento para los parámetros
Condición para fin exitoso	Los motores se configuraron satisfactoriamente.
Condición para	Los motores NO se configuraron satisfactoriamente

fin erróneo			
Actores primarios y secundarios	Administrador		
Disparador	Solicitud de configuración de motores		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador elige las opciones de configuración de motores.	
	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3	El administrador registra la velocidad deseada de los pasos, la longitud (pasos medios o completos) de éstos y la dirección (definición del sentido de giro).	
	4		El sistema hace una validación y actualiza los parámetros.
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	
	1	Si no hay cambio en la posición no reportan nada.	

Caso de Uso 12	REINICIALIZACIÓN		
Tipo	Primario, esencial		
Objetivo	Colocar en valores iniciales los parámetros de todo el sistema.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones			
Condición para fin exitoso	Los parámetros del sistema son reinicializados correctamente		
Condición para fin erróneo	Los parámetros del sistema no son reinicializados correctamente		
Actores primarios y secundarios	Administrador		
Disparador	Solicitud de reinicialización		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**

	1	El usuario realiza la petición de reinicialización de parámetros	
	2		El sistema muestra las opciones de parámetros a reinicializar.
	3	El usuario elige el parámetro de sistema (Cuenta de los sensores, Posición actual del telescopio) que sé reinicializará.	
	4		El sistema recupera el valor inicial y lo asigna al parámetro seleccionado.
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	

Caso de Uso 13	COMUNICACIONES		
Tipo	Primario, esencial		
Objetivo	Probar comunicación con los dispositivos de I/O que conectan los motores y los sensores al sistema.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones			
Condición para fin exitoso	La prueba es capaz de determinar si existe o no comunicación con los motores y los sensores		
Condición para fin erróneo	La prueba no es capaz de determinar si existe o no comunicación con los motores y/o sensores.		
Actores primarios y secundarios	Administrador		
Disparador	Solicitud de prueba de comunicaciones		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	El administrador realiza la petición de pruebas de comunicaciones	
	2		El sistema despliega las opciones de prueba
	3	El administrador elige el dispositivo al que realizará la prueba y los valores de configuración	
	4		El sistema envía datos de prueba por los dispositivos de comunicaciones a los motores y sensores.

	5	El administrador valida que los datos se hayan enviado.	
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	

Caso de Uso 14	CALIBRAR		
Tipo	Primario, esencial		
Objetivo	Calibrar los motores y los sensores, asignando una equivalencia grados-pasos de motor y grados-pasos del sensor.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones			
Condición para fin exitoso	La nueva relación de grados pasos es admitida por el sistema.		
Condición para fin erróneo	La nueva relación de grados pasos NO es admitida por el sistema.		
Actores primarios y secundarios	Administrador		
Disparador	Solicitud de prueba de comunicaciones		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	El administrador realiza la petición de calibración	
	2		El sistema despliega las opciones de calibración
	3	El administrador escoge el dispositivo a calibrar.	
	4	El administrador asigna la equivalencia en grados-pasos para el dispositivo.	
	5		El sistema salva la nueva relación de grados –pasos del dispositivo seleccionado.
	6	El administrador verifica visualmente que la calibración este hecha	
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	

Caso de Uso 15	ENCOLAR PETICIÓN DE OBSERVACIÓN
Tipo	Secundario, esencial
Objetivo	Formar la petición de observación de coordenadas absolutas para que se mueva el telescopio a este punto.
Referencias	???
Alcance y Nivel	Sistema, 2
Pre-condiciones	Que se configure un criterio para atender las peticiones.
Condición para fin exitoso	La petición logro ser encolada.

Condición para fin erróneo	La petición NO logro ser encolada.		
Actores primarios y secundarios	Usuario		
Disparador	Petición de encolamiento de observaciones.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	El usuario realiza la petición de encolamiento.	
	2		El sistema recibe la petición.
	3	El usuario indica las coordenadas absolutas del punto a observar (ascensión recta, AR y declinación, DEC).	
	4		Se guarda la petición en una lista de observaciones
Extensiones	**Paso**	**Acción de la ramificación**	
Variaciones	**Paso**	**Acción de la ramificación**	

Caso de Uso 16	MANTENIMIENTO AL CATÁLOGO DE PLANETAS Y SATÉLITES		
Tipo	Secundario, esencial		
Objetivo	El administrador necesita mantener actualizado el catálogo de planetas y satélites a observar.		
Referencias	???		
Alcance y Nivel	Sistema, 2		
Pre-condiciones	Ninguna		
Condición para fin exitoso	El catálogo de planetas y satélites se mantiene actualizado.		
Condición para fin erróneo	El catálogo de planetas y satélites no se mantiene actualizado.		
Actores primarios y secundarios	Administrador.		
Disparador	El Administrador elige alguna de las opciones de administración.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el Administrador elige algunas de las opciones de administración.	

	2		Realiza la consulta a la BD Para hacer la validación de usuarios y verificar que tenga atributos de administrador.
	3		Se despliega en pantalla el menú de opciones: Alta Planeta o Satélite, Baja planeta o Satélite o modificación de Planeta o Satélite.
	4	El administrador selecciona alguna de las opciones	
Extensiones	**Paso**	**Acción de la ramificación**	
	2	En caso de que no tenga atributos de administrador, se le notifica que no puede realizar la operación. Y termina el caso de uso.	
Variaciones	**Paso**	**Acción de la ramificación**	
	4	El administrador elige dar de alta un planeta o satélite dando todos sus atributos, ejemplo: nombre, excentricidad, distancia media al sol, etc. El identificador se asigna automáticamente.	
	4	El administrador elige la opción de dar de baja un planeta o satélite indicando el identificador del planeta o satélite que será eliminado.	
	4	El administrador selecciona modificar un planeta o satélite, indica el identificador del un planeta o satélite y modifica los atributos.	

Caso de Uso 17	CONSULTAR PLANETAS O SATÉLITES.
Tipo	Secundario, esencial
Objetivo	El usuario consulta los atributos de un planeta o satélite en el catálogo.
Referencias	
Alcance y Nivel	Sistema, 2
Pre-condiciones	Que estén creados los usuarios. Que los catálogos de planetas y satélites tengan información.
Condición para fin exitoso	El usuario logra consultar información de los catálogos.
Condición para fin erróneo	El usuario NO obtiene información de los catálogos. (Por error, o por que planeta o satélite no esta en catálogo).
Actores primarios y secundarios	Usuario

Disparador	El usuario elige la opción de consulta de planetas o satélites.		
Descripción	**Paso**	**Acción del actor**	**Acción del sistema**
	1	Este caso comienza cuando el usuario elige la opción de consulta de planetas o satélites.	
	2		Se despliega en pantalla los criterios, por los que se puede hacer la búsqueda.
	3	El usuario selecciona criterio de las opciones.	
	4		Se despliega la pantalla el resultado de la consulta elegida.
Extensiones	**Paso**	**Acción de la ramificación**	
	4	En caso de no existir el planeta o satélite, el sistema lo informa al usuario.	
Variaciones	**Paso**	**Acción de la ramificación**	
	3	El usuario escoge el criterio de búsqueda, ejemplo: nombre, masa, distancia media al sol, excentricidad, etc.	

Diagramas de Secuencia

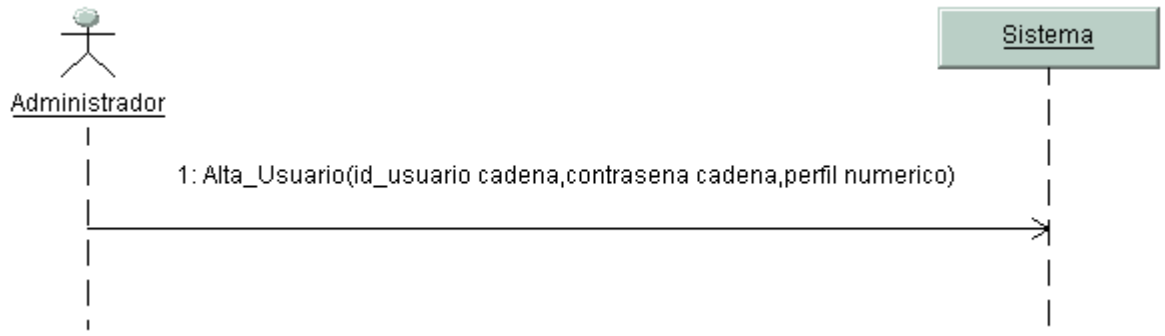


Diagrama de secuencia 1 del caso de uso Administración de Usuarios.

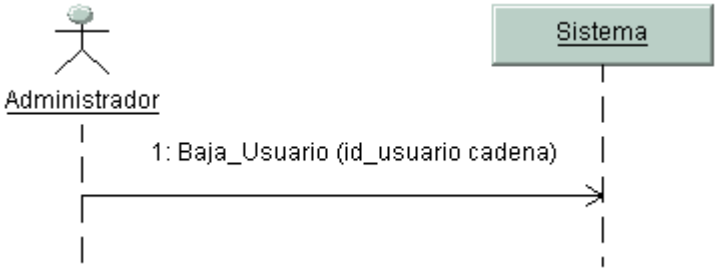


Diagrama de secuencia 2 del caso de uso Administración de Usuarios

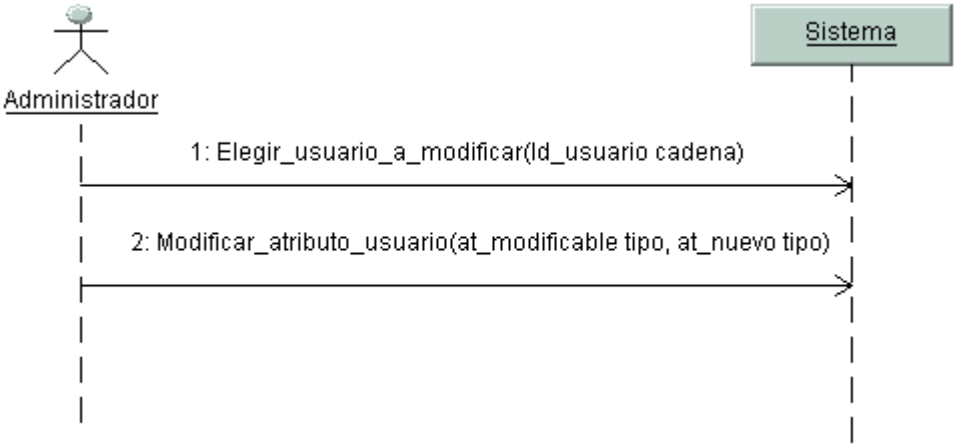


Diagrama de secuencia 3 del caso de uso Administración de Usuarios

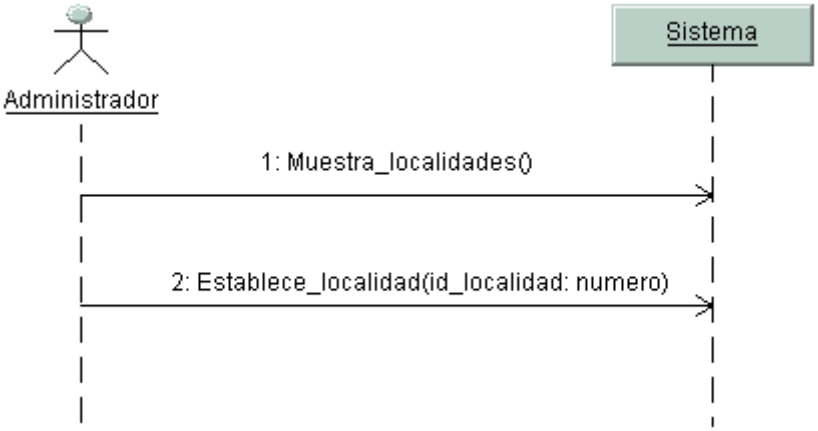


Diagrama de secuencia 1 del caso de uso Configuración del Sistema

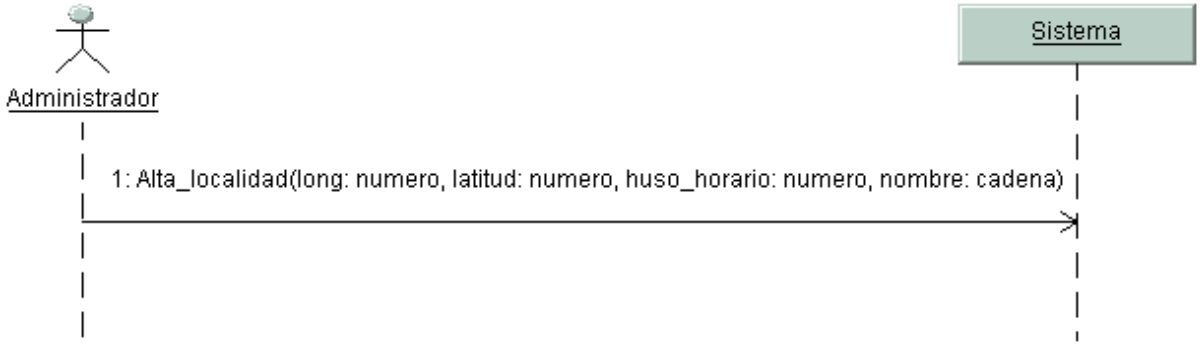


Diagrama de secuencia 2 del caso de uso Configuración del Sistema

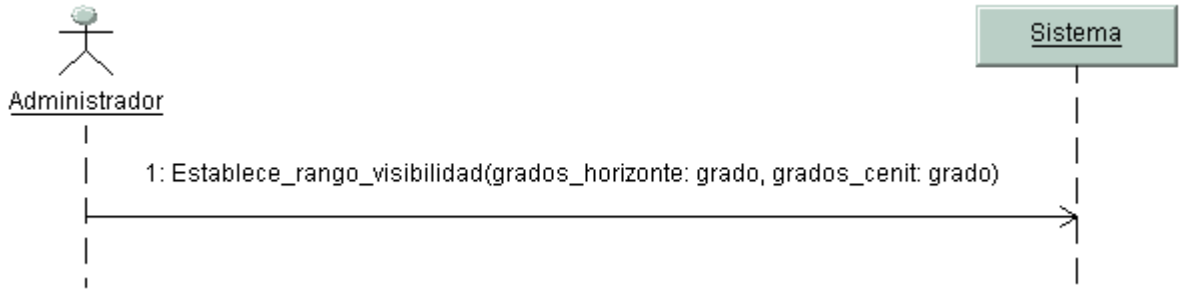


Diagrama de secuencia 3 del caso de uso Configuración del Sistema

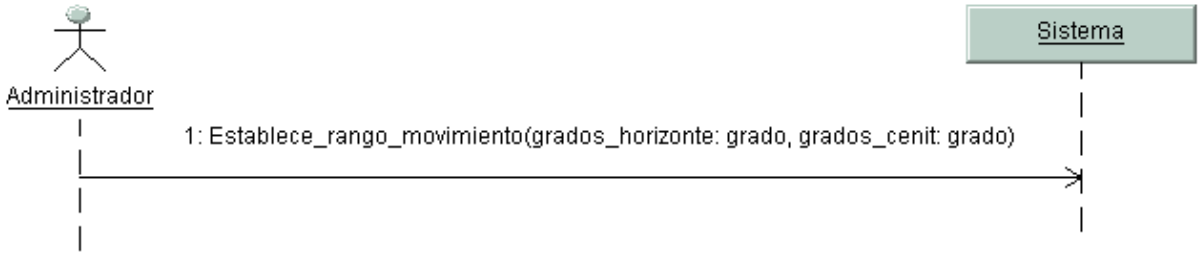


Diagrama de secuencia 4 del caso de uso Configuración del Sistema

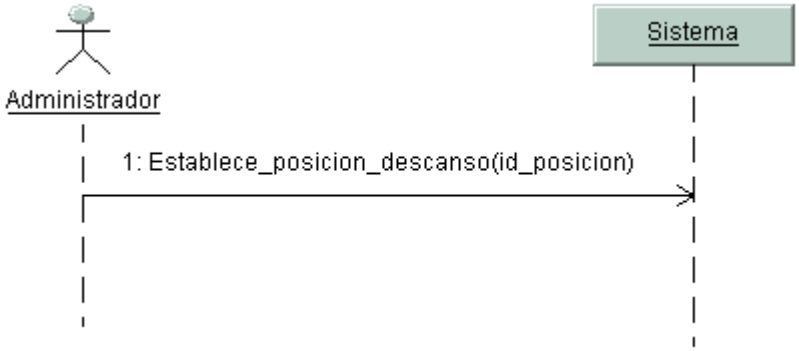


Diagrama de secuencia 5 del caso de uso Configuración del Sistema

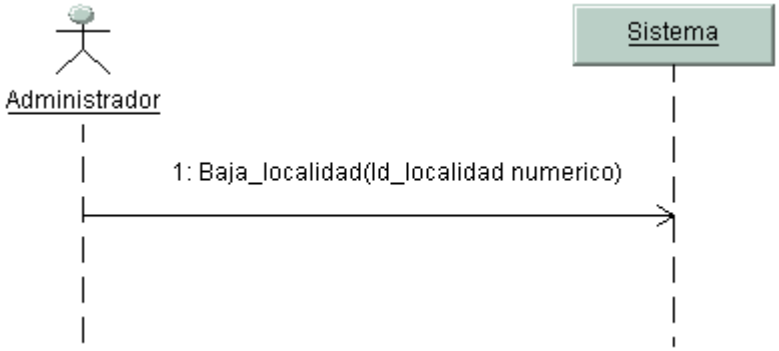


Diagrama de secuencia 6 del caso de uso Configuración del Sistema

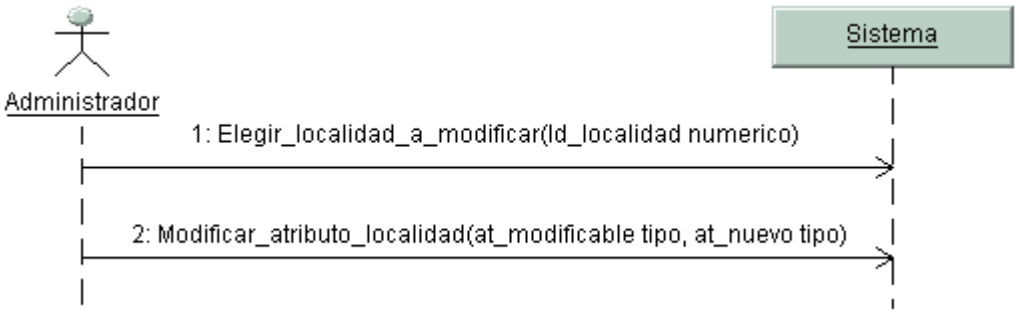


Diagrama de secuencia 7 del caso de uso Configuración del Sistema

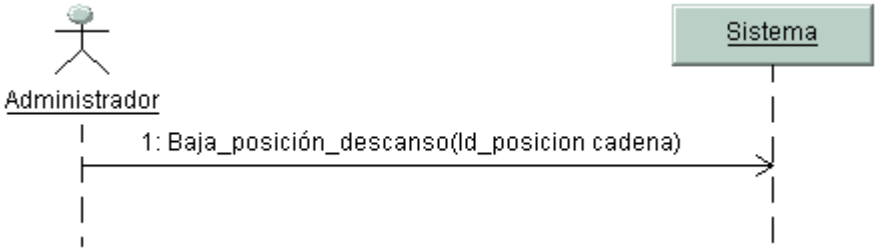


Diagrama de secuencia 8 del caso de uso Configuración del Sistema

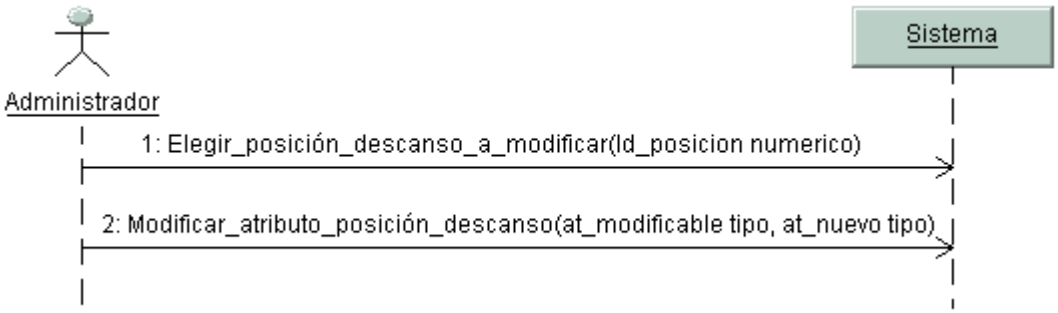


Diagrama de secuencia 9 del caso de uso Configuración del Sistema

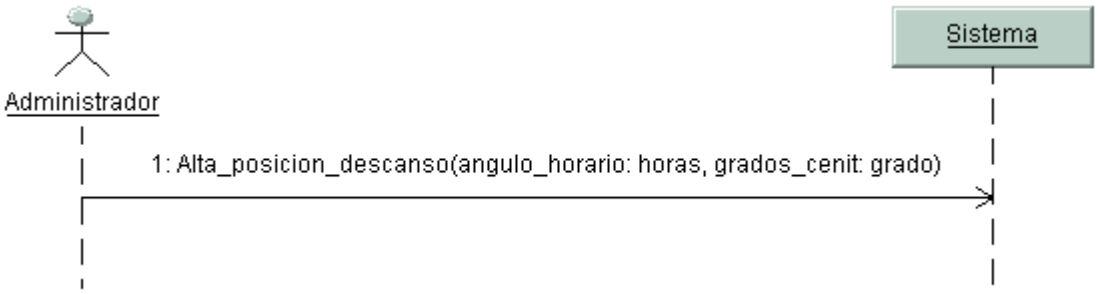


Diagrama de secuencia 10 del caso de uso Configuración del Sistema

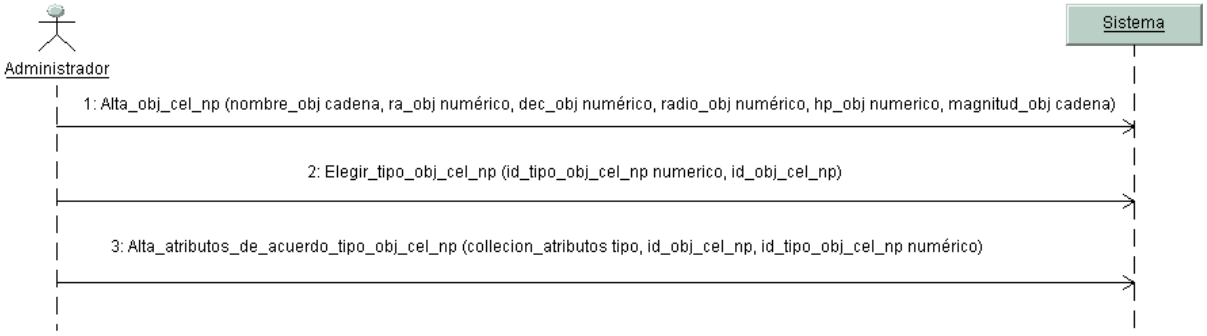


Diagrama de secuencia 1 del caso de uso Mantenimiento al catálogo de Objetos celestes NP

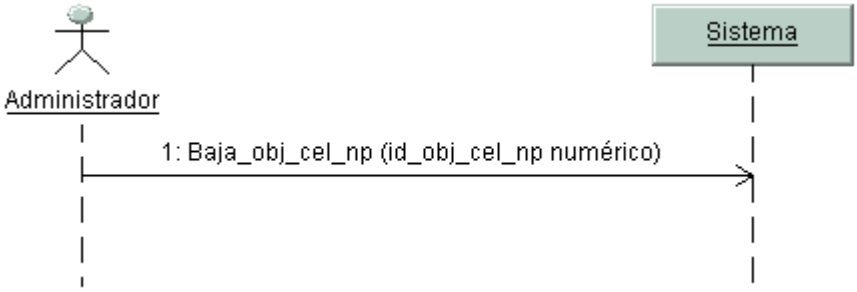


Diagrama de secuencia 2 del caso de uso Mantenimiento al catálogo de Objetos celestes NP

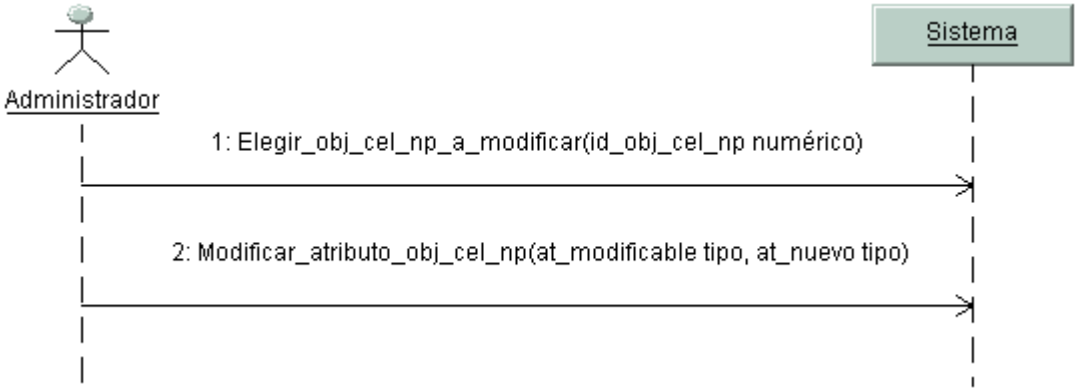


Diagrama de secuencia 3 del caso de uso Mantenimiento al catálogo de Objetos celestes NP

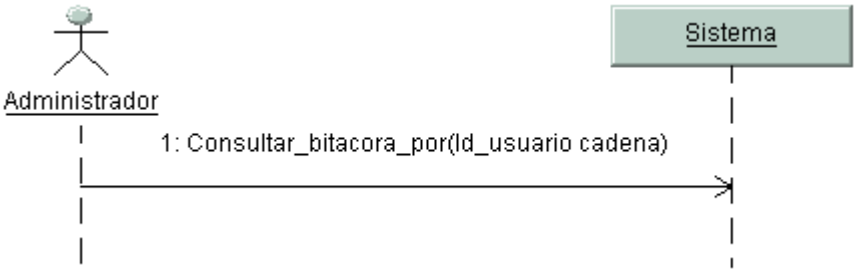


Diagrama de secuencia 1 del caso de uso Consultar Bitácora de Observación

Diagrama de secuencia 2 del caso de uso Consultar Bitácora de Observación

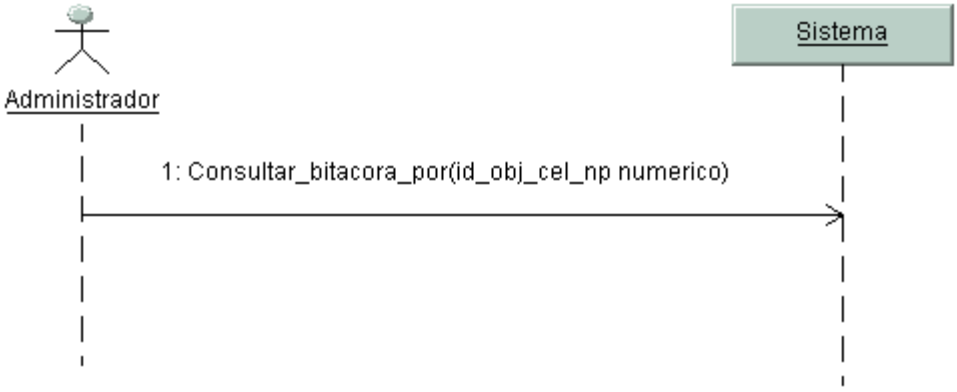


Diagrama de secuencia 3 del caso de uso Consultar Bitácora de Observación

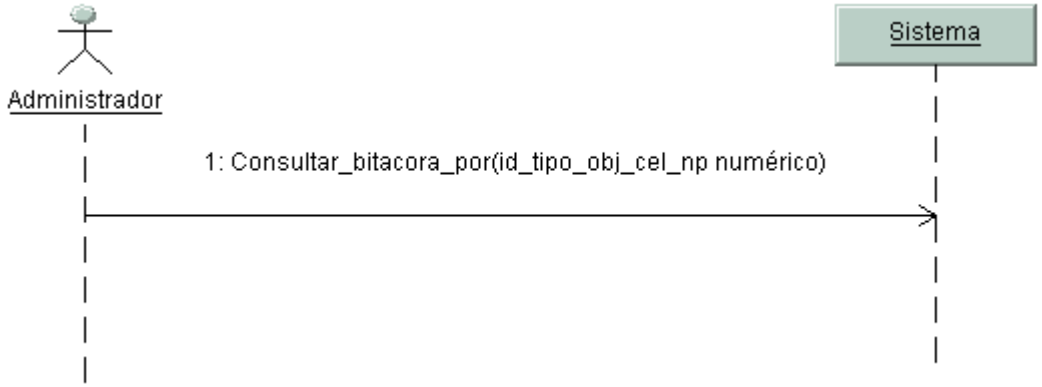


Diagrama de secuencia 4 del caso de uso Consultar Bitácora de Observación

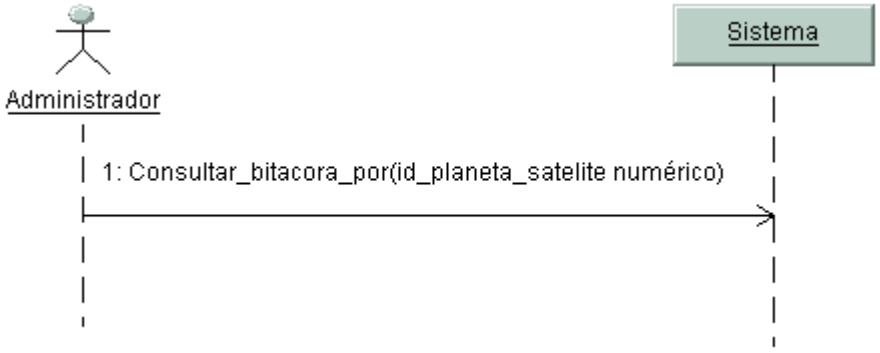


Diagrama de secuencia 5 del caso de uso Consultar Bitácora de Observación

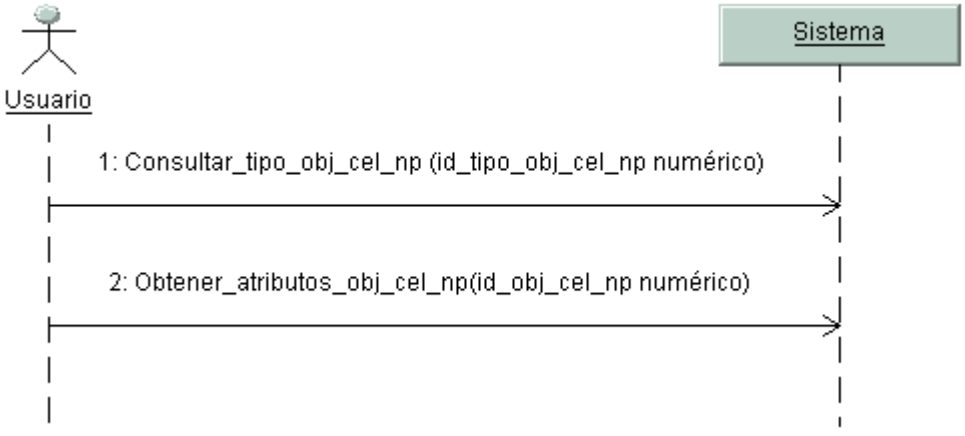


Diagrama de secuencia 1 del caso de uso Consultar Tipo de Objeto Celeste NP

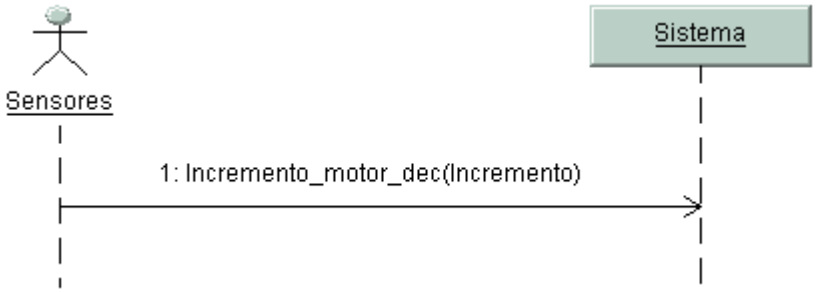


Diagrama de secuencia 1 del caso de uso Informar la Posición Actual del Telescopio

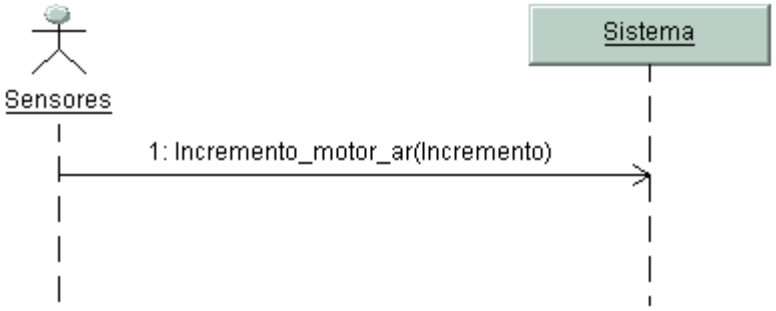


Diagrama de secuencia 2 del caso de uso Informar la Posición Actual del Telescopio

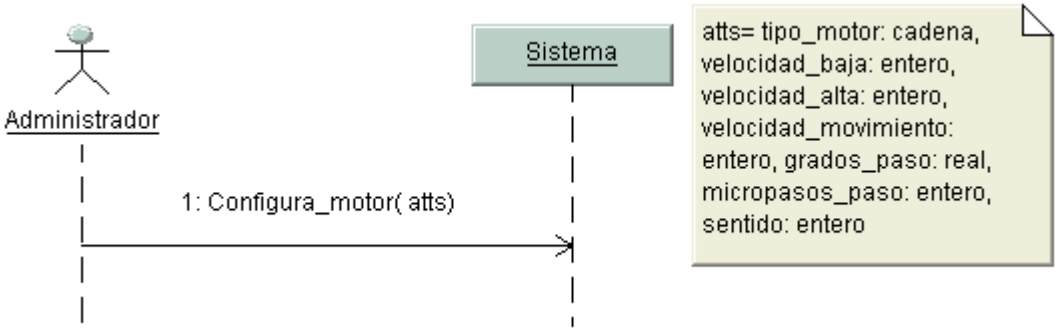


Diagrama de secuencia 1 del caso de uso Configurar Motores

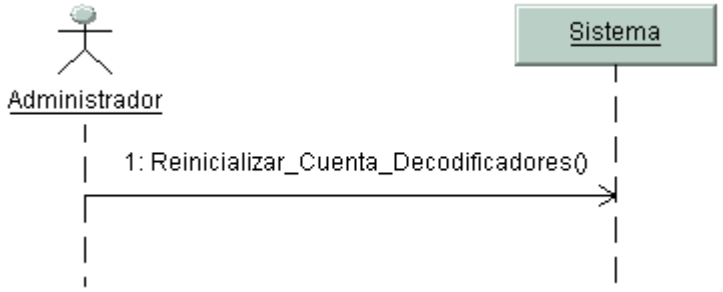


Diagrama de secuencia 1 del caso de uso Reinicialización

Diagrama de secuencia 2 del caso de uso Reinicialización

Diagrama de secuencia 1 del caso de uso Comunicaciones

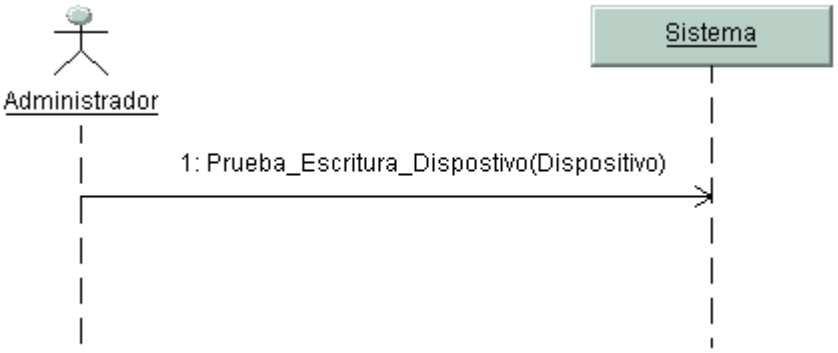


Diagrama de secuencia 2 del caso de uso Comunicaciones

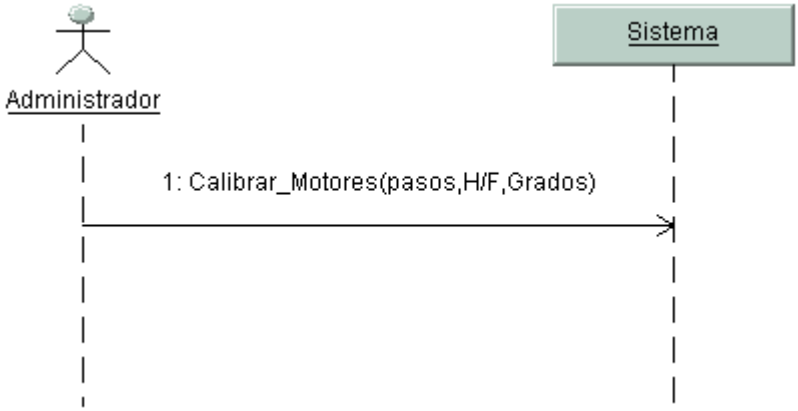


Diagrama de secuencia 1 del caso de uso Calibrar

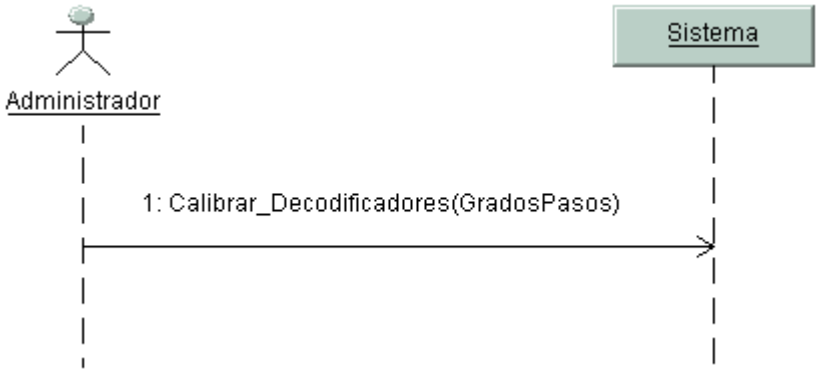


Diagrama de secuencia 2 del caso de uso Calibrar

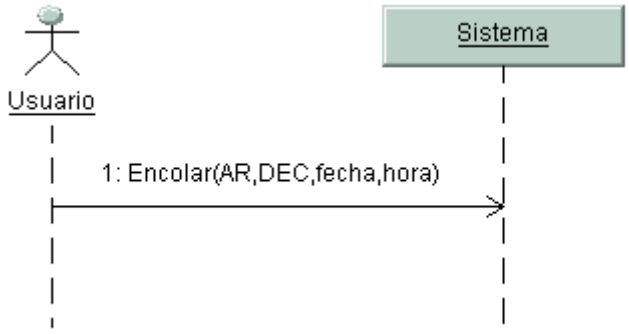


Diagrama de secuencia 1 del caso de uso Encolar Petición de Observación

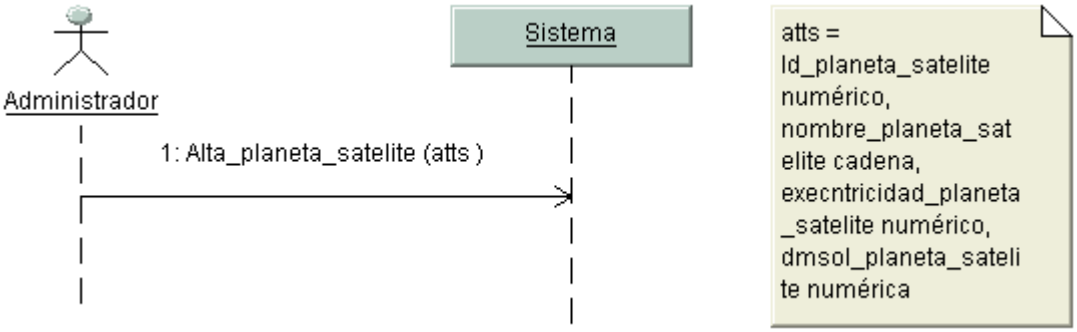


Diagrama de secuencia 1 del caso de uso mantenimiento al Catalogo de Planetas y Satélites

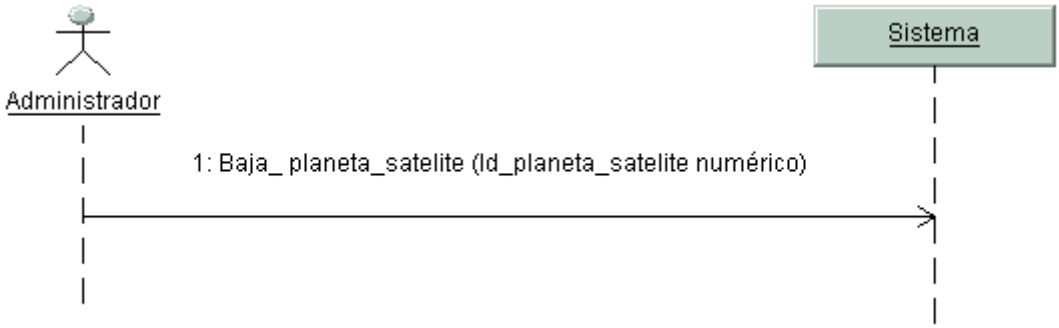


Diagrama de secuencia 2 del caso de uso mantenimiento al Catalogo de Planetas y Satélites

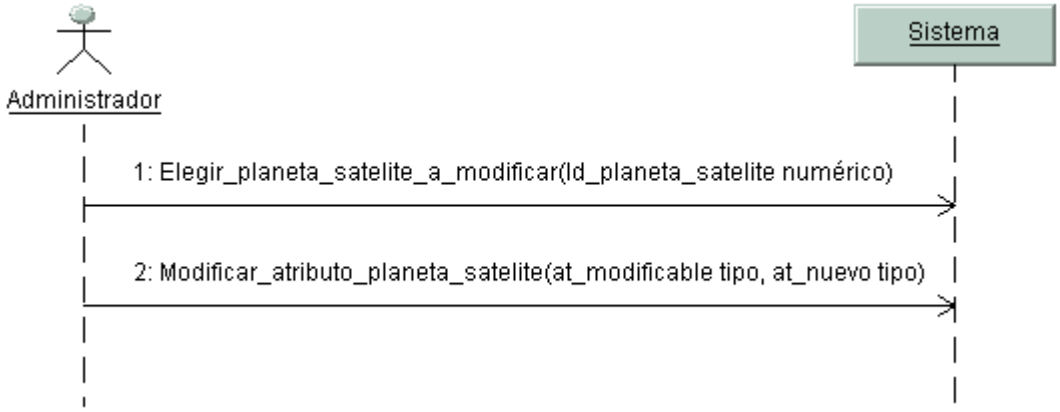


Diagrama de secuencia 3 del caso de uso mantenimiento al Catalogo de Planetas y Satélites

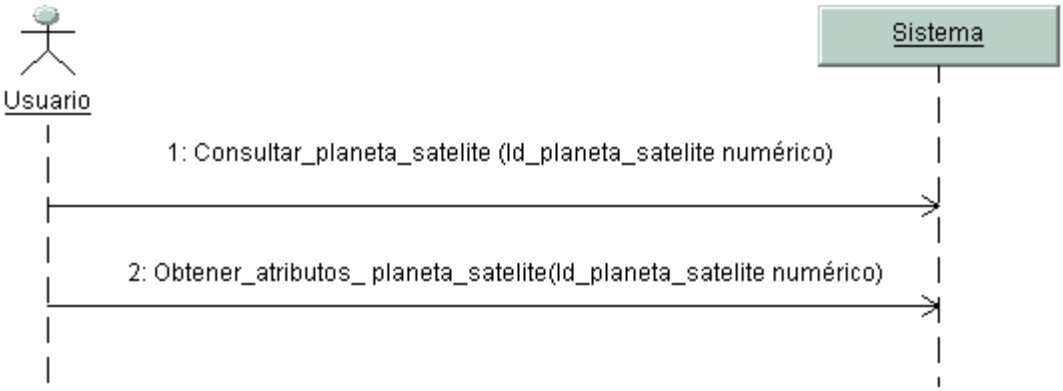


Diagrama de secuencia 1 del caso de uso mantenimiento al Consultar Planetas y Satélites

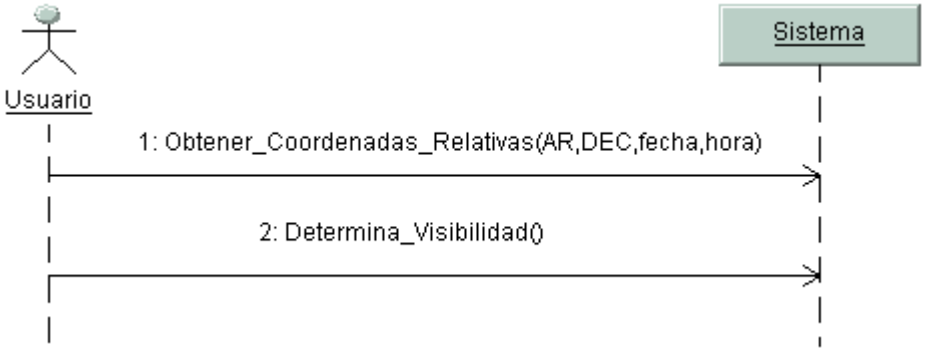


Diagrama de secuencia 1 del caso de uso mantenimiento al Obtener Coordenadas Relativas de Planetas y Satélites

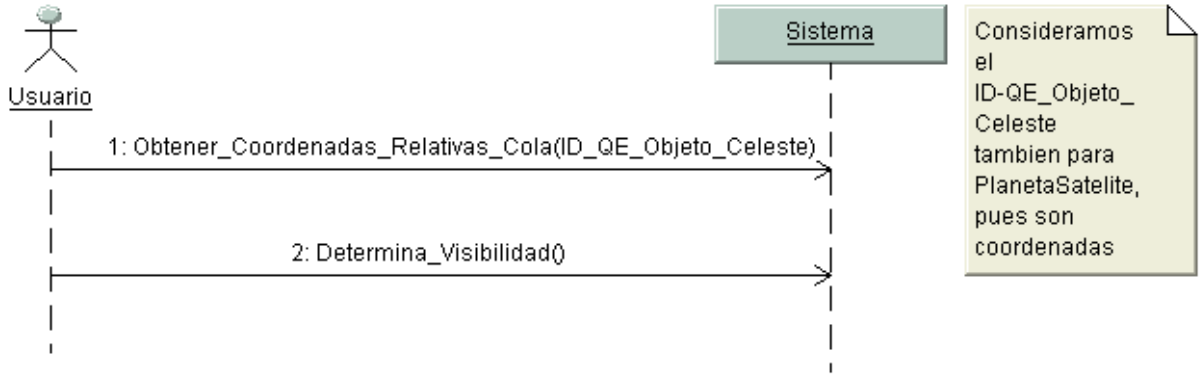


Diagrama de secuencia 2 del caso de uso mantenimiento al Obtener Coordenadas Relativas de Planetas y Satélites

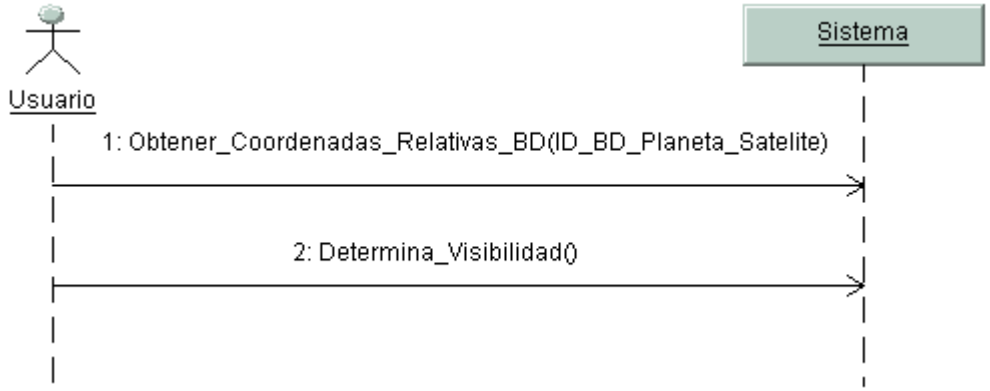


Diagrama de secuencia 3 del caso de uso mantenimiento al Obtener Coordenadas Relativas de Planetas y Satélites

Contratos

Contrato	CU001-DS001-OP001
Nombre	Alta_usuario(Id_usuario cadena,contraseña cadena, perfil numérico)
Responsabilidades	Dar de Alta un usuario.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del usuario es único, se registrará un nuevo usuario en el sistema
Excepciones	Que el usuario a ya exista.
Precondiciones	El atributo usuario identificador debe ser proporcionado
Poscondiciones	Se crea la instancia usuario, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU001-DS002-OP001
Nombre	Baja_usuario(Id_usuario cadena)
Responsabilidades	Eliminar usuario
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se borrara la instancia del usuario en todos los contextos
Excepciones	Que usuario no exista o haya sido borrado previamente
Precondiciones	Que exista la instancia del usuario.
Poscondiciones	Se borra o destruye la instancia de usuario de acuerdo al id_usuario

Contrato	CU001-DS003-OP001
Nombre	Elegir_usuario_a_modificar(Id_usuario cadena)
Responsabilidades	Buscar el usuario a modificar de acuerdo a su identificador.
Tipo / Referencias	Sistema

cruzadas / Salidas	
Notas	El identificador del usuario es único.
Excepciones	Que el usuario a modificar no exista.
Precondiciones	El atributo usuario identificador ya es conocido.
Poscondiciones	Se crea la instancia usuario, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU001-DS003-OP002
Nombre	Modificar_atributo_usuario(at_modificable tipo, at_nuevo tipo)
Responsabilidades	Elegir al atributo a modificar, proporcionar el nuevo valor del atributo y registrarlo.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Los atributos a modificar serán los de la instancia usuario generada en el contrato CU001-DS003-OP001. El tipo de dato de los atributos a modificar (nuevo y anterior) deberán ser iguales.
Excepciones	Que el nuevo valor del atributo no cumpla con el formato del atributo, sea un valor duplicado (Id's) o sea un valor incoherente (Tipo de dato)
Precondiciones	Que la instancia usuario haya sido creada previamente, tenga inicializados sus atributos(identificador, contraseña, perfil)
Poscondiciones	De la instancia usuario se elige el at_modificable, se verifica que el valor del at_nuevo coincida en tipo y se asigna el nuevo valor a la instancia.

Contrato	CU002-DS001-OP001
Nombre	Muestra_localidades()
Responsabilidades	Muestra las localidades dadas de alta en el sistema.
Tipo / Referencias cruzadas / Salidas	----------

Notas	----------
Excepciones	No muestra ninguna localidad por que no han sido dadas de alta al menos una.
Precondiciones	----------
Poscondiciones	El administrador observa las localidades disponibles. Se crea una colección de localidades disponibles, se asocian las localidades a la colección.

Contrato	CU002-DS001-OP002
Nombre	Establece_localidad(id_localidad: numero)
Responsabilidades	Establece la localidad actual del sistema
Tipo / Referencias cruzadas / Salidas	----------
Notas	----------
Excepciones	No se puede elegir una localidad por que no ha sido dada de alta.
Precondiciones	Debe haber al menos una localidad posible.
Poscondiciones	Al elegir la localidad actual la instancia ***localidad*** se actualiza con los datos de longitud, latitud, huso horario del lugar de observación elegido. Se elige una localidad de la colección de las localidades.

Contrato	CU002-DS002-OP001
Nombre	Alta_localidad(long: numero, latitud: numero, uso_horario: numero, nombre: cadena)
Responsabilidades	Da de alta una nueva localidad en el sistema
Tipo / Referencias cruzadas / Salidas	----------
Notas	----------

Excepciones	No se puede dar de alta por error en el tipo de datos. No se puede dar de alta por error en el rango de los datos.
Precondiciones	----------
Poscondiciones	Se crea una instancia temporal de ***localidad*** y se envía a la clase ***controladora*** para que la registre en el sistema.

Contrato	CU002-DS003-OP001
Nombre	Establece_rango_visibilidad(grados_horizonte: grado, grados_cenit: grado)
Responsabilidades	Establece el rango de visibilidad del telescopio.
Tipo / Referencias cruzadas / Salidas	----------
Notas	Este rango de visibilidad se medirá en grados Limite no visible 1: desde el horizonte hacia arriba (hasta las montañas o borde de la cúpula) Limite no visible 2: Y un circulo superior desde el cenit hacia abajo que no es visible por la arquitectura de la bóveda.
Excepciones	Errores por traslape de limites.
Precondiciones	----------
Poscondiciones	Los limites de visibilidad del telescopio se actualizarán y los tomará la instancia ***que corresponda(al parecer observación)***

Contrato	CU002-DS004-OP001
Nombre	Establece_rango_movimiento(grados_horizonte:grado, grados_cenit: grado)
Responsabilidades	Establece el rango de movimiento del telescopio.

Tipo / Referencias cruzadas / Salidas	----------
Notas	Este rango de movimiento se medirá en grados. Se debe a los limites mecánicos de movimiento del telescopio. Limite no visible 1: desde el horizonte hacia arriba (hasta las montañas o borde de la cúpula) Limite no visible 2: Y un circulo superior desde el cenit hacia abajo que no es visible por la arquitectura de la bóveda.
Excepciones	Errores por traslape de limites.
Precondiciones	----------
Poscondiciones	Los limites de movimiento del telescopio se actualizarán y los tomará la instancia ***que corresponda (al parecer observación)***

Contrato	CU002-DS005-OP001
Nombre	Establece_posicion_descanso(angulo_horario: horas, grados_cenit: grado)
Responsabilidades	Establece la posición de descanso del telescopio.
Tipo / Referencias cruzadas / Salidas	----------
Notas	La posición de descanso deberá establecerse después de configurar los limites de movimiento.
Excepciones	Errores al configurar por traslape de limites de movimiento.
Precondiciones	----------
Poscondiciones	La posición de descanso se actualizará. Limites o arquitectura.

Contrato	CU002-DS006-OP001
Nombre	Baja_localidad(Id_localidad numérico)
Responsabilidades	Eliminar localidad
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se borrara la instancia de la localidad en todos los contextos
Excepciones	Que la localidad no exista o haya sido borrado previamente
Precondiciones	Que exista la instancia de la localidad.
Poscondiciones	Se borra o destruye la instancia de localidad de acuerdo al id_localidad

Contrato	CU002-DS007-OP001
Nombre	Elegir_localidad_a_modificar(Id_localidad numerico)
Responsabilidades	Buscar la localidad a modificar de acuerdo a su identificador.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador de la localidad es único.
Excepciones	Que la localidad a modificar no exista.
Precondiciones	El atributo localidad identificador ya es conocido.
Poscondiciones	Se crea la instancia localidad, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU002-DS007-OP002
Nombre	Modificar_atributo_localida(at_modificable tipo, at_nuevo tipo)
Responsabilidades	Elegir al atributo a modificar, proporcionar el nuevo valor del atributo y registrarlo.
Tipo / Referencias cruzadas / Salidas	Sistema

Notas	Los atributos a modificar serán los de la instancia localidad generada en el contrato CU002-DS002-OP001. El tipo de dato de los atributos a modificar (nuevo y anterior) deberán ser iguales.
Excepciones	Que el nuevo valor del atributo no cumpla con el formato del atributo, sea un valor duplicado (Id's) o sea un valor incoherente (Tipo de dato)
Precondiciones	Que la instancia localidad haya sido creada previamente, tenga inicializados sus atributos
Poscondiciones	De la instancia localidad se elige el at_modificable, se verifica que el valor del at_nuevo coincida en tipo y se asigna el nuevo valor a la instancia.

Contrato	CU002-DS008-OP001
Nombre	Baja_posición_descanso(Id_posicion cadena)
Responsabilidades	Eliminar posición de descanso
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se borrara la instancia de la posición de descanso en todos los contextos
Excepciones	Que la posición de descanso no exista o haya sido borrado previamente
Precondiciones	Que exista la instancia de posición de descanso.
Poscondiciones	Se borra o destruye la instancia de posición de descanso de acuerdo al id_posición_descanso

Contrato	CU002-DS009-OP001
Nombre	Elegir_posición_descanso_a_modificar(Id_posicion numérico)
Responsabilidades	Buscar la POSICIÓN de descanso a modificar de acuerdo a su identificador.
Tipo / Referencias cruzadas / Salidas	Sistema

Notas	El identificador de la POSICIÓN de descanso es único.
Excepciones	Que la POSICIÓN de descanso a modificar no exista.
Precondiciones	El atributo POSICIÓN de desanso identificador ya es conocido.
Poscondiciones	Se crea la instancia POSICIÓN de descanso, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU002-DS009-OP002
Nombre	Modificar_atributo_posición_descanso(at_modificable tipo, at_nuevo tipo)
Responsabilidades	Elegir al atributo a modificar, proporcionar el nuevo valor del atributo y registrarlo.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Los atributos a modificar serán los de la instancia posicion de descanso generada en el contrato CU002-DS005-OP001. El tipo de dato de los atributos a modificar (nuevo y anterior) deberán ser iguales.
Excepciones	Que el nuevo valor del atributo no cumpla con el formato del atributo, sea un valor duplicado (Id's) o sea un valor incoherente (Tipo de dato)
Precondiciones	Que la instancia POSICIÓN de descanso haya sido creada previamente, tenga inicializados sus atributos
Poscondiciones	De la instancia localidad se elige el at_modificable, se verifica que el valor del at_nuevo coincida en tipo y se asigna el nuevo valor a la instancia.

Contrato	CU002-DS010-OP001
Nombre	Alta_posicion_descanso(angulo_horario: horas, grados_cenit: grado)
Responsabilidades	Da de alta la posición de descanso del telescopio.

Tipo / Referencias cruzadas / Salidas	Sistema
Notas	La posición de descanso deberá establecerse después de configurar los limites de movimiento.
Excepciones	Errores al configurar por traslape de limites de movimiento. Que la posición ya exista.
Precondiciones	El identificador de posición debe ser proporcionado
Poscondiciones	La posición de descanso se actualizará. Se crea la instancia posición

Contrato	CU003-DS001-OP001
Nombre	Alta_obj_cel_np (nombre_obj cadena, ra_obj numérico, dec_obj numérico, radio_obj numérico, hp_obj numérico, magnitud_obj cadena)
Responsabilidades	Registrar un nuevo objeto celeste np en el sistema.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se creara una instancia por cada objeto celeste np.
Excepciones	Que el objeto celeste np ya exista
Precondiciones	
Poscondiciones	Se creo una nueva instancia de objeto_celeste_np al asignarles valor a los atributos nombre_obj, ra_obj, dec_obj, radio_obj, hp_obj, magnitud_obj

Contrato	CU003-DS001-OP002
Nombre	Elegir_tipo_obj_cel_np (id_tipo_obj_cel_np numérico, id_obj_cel_np numérico)
Responsabilidades	Asignarle a un a instancia de objeto celeste np el tipo de objeto celeste np al que pertenece.

Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se asignara solo un tipo de objeto celeste np al objeto celeste np.
Excepciones	Que el objeto celeste np no haya sido creado previamente.
Precondiciones	Que exista la instancia de objeto celeste np
Poscondiciones	Se crea una instancia de Tipo de objeto celeste np de acuerdo al id_tipo_obj_cel_np y se asocia a la instancia de objeto celeste np por su id_obj_cel_np.

Contrato	CU003-DS001-OP003
Nombre	Alta_atributos_de_acuerdo_tipo_obj_cel_np (collecion_atributos tipo, id_obj_cel_np numérico, id_tipo_obj_cel_np numérico)
Responsabilidades	Asignarle a un a instancia de objeto celeste np la colección de atributos del tipo de objeto celeste np al que pertenece.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se asignara solo una colección de atributos del tipo de objeto celeste np al objeto celeste np.
Excepciones	Que el objeto celeste np no haya sido creado previamente o no tenga asignado un tipo de objeto celeste np
Precondiciones	Que exista la instancia de objeto celeste np y tenga asignado un tipo de objeto celeste np
Poscondiciones	Se agrega la colección de atributos a la instancia de objeto celeste np por su id_obj_cel_np de acuerdo al tipo de objeto celeste np id_tipo_obj_cel_np asignado previamente.

Contrato	CU003-DS002-OP001
Nombre	Baja_obj_cel_np (id_obj_cel_np numérico)
Responsabilidades	Eliminar un objeto celeste np.

Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se borrara la instancia del objeto celeste np en todos los contextos.
Excepciones	Que el objeto celeste np no exista o haya sido borrado previamente.
Precondiciones	Que exista la instancia del objeto celeste np.
Poscondiciones	Se borra o destruye la instancia de objeto_celeste_np de acuerdo al id_obj_cel_np, desasociándolo al mismo tiempo de su tipo de objeto celeste np asignado. La colección de atributos especiales de acuerdo al tipo de objeto celeste np se destruye con la instancia del objeto celeste np.

Contrato	CU003-DS003-OP001
Nombre	Elegir_obj_cel_np_a_modificar(id_obj_cel_np numérico)
Responsabilidades	Buscar el objeto celeste np a modificar de acuerdo a su identificador.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del objeto celeste np es único.
Excepciones	Que el objeto celeste a modificar no exista.
Precondiciones	El atributo objeto celeste np identificador ya es conocido.
Poscondiciones	Se crea la instancia del objeto celeste np, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU003-DS003-OP002
Nombre	Modificar_atributo_obj_cel_np(at_modificable tipo, at_nuevo tipo)
Responsabilidades	Elegir al atributo a modificar, proporcionar el nuevo valor del atributo y registrarlo.
Tipo / Referencias	Sistema

cruzadas / Salidas	
Notas	Los atributos a modificar serán los de la instancia objeto celeste np generada en el contrato CU003-DS003-OP001. El tipo de dato de los atributos a modificar (nuevo y anterior) deberán ser iguales.
Excepciones	Que el nuevo valor del atributo no cumpla con el formato del atributo, sea un valor duplicado (Id's) o sea un valor incoherente (Tipo de dato)
Precondiciones	Que la instancia objeto celeste np haya sido creada previamente, tenga inicializados sus atributos(nombre_obj cadena, ra_obj numérico, dec_obj numérico, radio_obj numérico, hp_obj numérico, magnitud_obj cadena y colección de atributos especiales)
Poscondiciones	De la instancia objeto celeste np se elige el at_modificable, se verifica que el valor del at_nuevo coincida en tipo y se asigna el nuevo valor a la instancia.

Contrato	CU004-DS001-OP001
Nombre	Consultar_bitacora_por(Id_usuario cadena)
Responsabilidades	Buscar en la bitácora de observación si el usuario proporcionado de acuerdo a su identificador, realizo observaciones.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del usuario es único y debe de estar almacenado en el objeto bitácora de observación.
Excepciones	Que el usuario no haya realizado observaciones.
Precondiciones	El atributo usuario identificador ya es conocido, ya existe la instancia de bitácora de observación.
Poscondiciones	Se obtiene de la instancia de bitácora de observación creada previamente, el usuario por su identificador Id_usuario cadena.

Contrato	CU004-DS002-OP001
Nombre	Consultar_bitacora_por(fecha)
Responsabilidades	Buscar en la bitácora de observación si sé, realizaron observaciones en la fecha estipulada.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	La fecha debe de estar almacenado en el objeto bitácora de observación.
Excepciones	Que no se hayan realizado observaciones en la fecha dada.
Precondiciones	La fecha ya es conocida, ya existe la instancia de bitácora de observación.
Poscondiciones	Se obtiene de la instancia de bitácora de observación creada previamente, las observaciones realizadas en la fecha proporcionada.

Contrato	CU004-DS003-OP001
Nombre	Consultar_bitacora_por(id_obj_cel_np numérico)
Responsabilidades	Buscar en la bitácora de observación si se realizaron observaciones del objeto celeste np.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del objeto celeste np es único y debe de estar almacenado en el objeto bitácora de observación.
Excepciones	Que el objeto celeste np no haya sido observado.
Precondiciones	El atributo objeto celeste np identificador ya es conocido, ya existe la instancia de bitácora de observación.
Poscondiciones	Se obtiene de la instancia de bitácora de observación creada previamente, el objeto celeste np por su identificador id_obj_cel_np numérico.

Contrato	CU004-DS004-OP001
Nombre	Consultar_bitacora_por(id_tipo_obj_cel_np numérico)
Responsabilidades	Buscar en la bitácora de observación si se realizaron observaciones del tipo de objeto celeste np proporcionado.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del tipo objeto celeste np es único y debe de estar almacenado en el objeto bitácora de observación.
Excepciones	Que el tipo de objeto celeste np no haya sido observado.
Precondiciones	El atributo tipo de objeto celeste np identificador ya es conocido, ya existe la instancia de bitácora de observación.
Poscondiciones	Se obtiene de la instancia de bitácora de observación creada previamente, el tipo objeto celeste np por su identificador id_tipo_obj_cel_np numérico.

Contrato	CU004-DS005-OP001
Nombre	Consultar_bitacora_por(id_planeta_satelite numérico)
Responsabilidades	Buscar en la bitácora de observación si se realizaron observaciones de planetas o satélites.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del planeta/satélite es único y debe de estar almacenado en el objeto bitácora de observación.
Excepciones	Que el planeta o satélite no haya sido observado.
Precondiciones	El atributo planeta/satélite identificador ya es conocido, ya existe la instancia de bitácora de observación.
Poscondiciones	Se obtiene de la instancia de bitácora de observación creada previamente, el planeta o satélite por su identificador id_planeta_satelite numérico.

Contrato	CU005-DS001-OP001

Nombre	Consultar_tipo_obj_cel_np (id_tipo_obj_cel_np numérico)
Responsabilidades	Consultar los tipos de objetos celestes np que existen.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del tipo de objeto celeste np es único, y esta operación es una especie de filtro para obtener los atributos de un objeto celeste np especifico.
Excepciones	Que el tipo de objeto celeste np exista.
Precondiciones	El atributo tipo de objeto celeste np identificador ya es conocido.
Poscondiciones	Se obtiene el tipo de objeto celeste np de acuerdo al identificador id_tipo_obj_cel_np numérico proporcionado.

Contrato	CU005-DS001-OP002
Nombre	Obtener_atributos_obj_cel_np(id_obj_cel_np numérico)
Responsabilidades	Obtener los atributos de un objeto celeste np de acuerdo al tipo de objeto celeste np que tenga asociado.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El objeto celeste np será buscado dentro del tipo especifico de tipo de objeto celeste np proporcionado por la operación anterior CU005-DS001-OP001.
Excepciones	Que el objeto celeste np no este asociado al tipo de objeto celeste np ya seleccionado. Que el objeto celeste np no exista.
Precondiciones	El atributo objeto celeste np identificador ya es conocido y ya se selecciono el tipo de objeto celeste np.
Poscondiciones	Se obtienen los atributos del objeto celeste np de acuerdo al identificador id_ obj_cel_np numérico proporcionado. Y del tipo de objeto celeste np al que esta asociado.

Contrato	CU006-DS001-OP001
Nombre	Obtener_Coordenadas_Relativas(AR, DEC, Fecha, Hora)
Responsabilidades	Obtener las coordenadas relativas del objeto celeste a observar (AH, DEC) a partir de coordenadas absolutas ingresadas por el usuario, desplegar el resultado.
Tipo / Referencias cruzadas / Salidas	
Notas	Las clases *Coordenadas* y *Localidad* que serán reutilizadas implementan ya la transformación de coordenadas. **NOTA:** Tal vez se debería cambiar el modelo conceptual y relacionar coordenadas a localidad directamente.
Excepciones	
Precondiciones	Debe existir una instancia de *Localidad*. Debe existir una instancia de *Usuario*.
Poscondiciones	Se creó una instancia de *coordenadas* pasando la AR y DEC Se creó una instancia de *Observación* Se asoció una *Observación* a *Usuario* Se asoció *Coordenadas* a *Localidad* Se asoció las *coordenadas* a una *observación* Se modificó los atributos *Observación.AR*, *Observación.DEC*, *Observación.AH , Observación.Fecha* y *Observación.Hora*

Contrato	CU006-DS001-OP002
Nombre	
Responsabilidades	Determina si el objeto celeste que se pretende observar es visible en ese momento con base en los límites de observación.
Tipo / Referencias cruzadas / Salidas	

Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Telescopio*
Poscondiciones	

Contrato	CU006-DS002-OP001
Nombre	Obtener_Coordenadas_Relativas_Cola(ID_QE_Objeto_Celeste)
Responsabilidades	Obtener las coordenadas relativas de una Cola de observaciones pendientes y Desplegarlas.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Cola de Observación* Deben existir elementos en la *Cola de Observación*
Poscondiciones	Se modificó la colección de objetos de *Cola de Observación*

Contrato	CU006-DS003-OP001
Nombre	Obtener_Coordenadas_Relativas_BD(ID_BD_Objeto_Celeste)
Responsabilidades	Obtener las coordenadas relativas de un Objeto Celeste contenido en la Base de Datos y desplegarlos.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Objeto Celeste*

Poscondiciones	Se creó una instancia de *coordenadas* a partir de los atributos de *Objeto Celeste* Se creó una instancia de *Observación* Se asoció una *Observación* a *Usuario* Se asoció *Observación* a *Localidad* Se asoció las *coordenadas* a una *observación* Se modificó los atributos *Observación.AR*, *Observación.DEC*, *observación.ah, Observación.Fecha* y *Observación.Hora*

Contrato	CU008-DS001-OP001
Nombre	Gira_motor_declinacion(pasos, sentido, tipopasos)
Responsabilidades	Mueve el motor de declinación tantos pasos de tipopasos, en el sentido especificado Tipo pasos especifica si los pasos son completos o micropasos y la velocidad.
Tipo / Referencias cruzadas / Salidas	Salida de datos hacia el puerto paralelo
Notas	Debe haber una notificación de inicio del movimiento y al final de procesar la petición de movimiento.
Excepciones	No se pueden enviar los datos al puerto paralelo. Se envían los datos pero no se mueve el motor.
Precondiciones	La comunicación entre el sistema y el motor debe estar activa.
Poscondiciones	El motor se mueve a la posición deseada.

Contrato	CU008-DS002-OP001
Nombre	Gira_motor_asencionrecta(pasos, sentido, tipopasos)
Responsabilidades	Mueve el motor de ascensión recta tantos pasos de tipopasos, en el sentido especificado Tipo pasos especifica si los pasos son completos o micropasos y la velocidad.
Tipo / Referencias cruzadas / Salidas	Salida de datos hacia el puerto paralelo
Notas	Debe haber una notificación de inicio del movimiento y al final de procesar la petición de movimiento.
Excepciones	No se pueden enviar los datos al puerto paralelo. Se envían los datos pero no se mueve el motor.
Precondiciones	La comunicación entre el sistema y el motor debe estar activa.
Poscondiciones	El motor se mueve a la posición deseada.

Contrato	CU009-DS001-OP001
Nombre	Incremento_motor_dec(Incremento: entero con signo)
Responsabilidades	Reporta al sistema cualquier cambio en la posición del eje de declinación por medio de un valor de incremento positivo o negativo dependiendo del sentido.
Tipo / Referencias cruzadas / Salidas	
Notas	El valor lo envía uno de los sensores construidos con el ratón. Debe establecerse cual es el sentido positivo o negativo de giro en el sistema de acuerdo a la orientación del sensor. Los incrementos se reportan por el puerto serial, en paquetes de información de acuerdo al estudio que ya se hizo del uso

	del mouse por el puerto serie.
Excepciones	No se reportan cambios en la posición por parte de los sensores. No se reflejan los cambios en la posición reportados por los sensores. La actualización de la ***posición actual*** no corresponde al cambio en la posición reportados.
Precondiciones	La relación de incremento/grados para los sensores de ascensión recta y declinación debe estar configurada. La ***posición actual*** debe estar inicializada. La comunicación entre el sistema y los sensores debe estar activa.
Poscondiciones	La variable de ***posición actual*** se deba actualizar de acuerdo al incremento obtenido y debe reportar la posición actual real.

Contrato	CU009-DS002-OP001
Nombre	Incremento_motor_ar(Incremento: entero con signo)
Responsabilidades	Reporta al sistema cualquier cambio en la posición del eje de ascensión recta por medio de un valor de incremento positivo o negativo dependiendo del sentido.
Tipo / Referencias cruzadas / Salidas	
Notas	El valor lo envía uno de los sensores construidos con el ratón. Debe establecerse cual es el sentido positivo o negativo de giro en el sistema de acuerdo a la orientación del sensor. Los incrementos se reportan por el puerto serial, en paquetes de información de acuerdo al estudio que ya se hizo del uso del mouse por el puerto serie.
Excepciones	No se reportan cambios en la posición por parte de los sensores. No se reflejan los cambios en la posición reportados por los

	sensores. La actualización de la ***posición actual*** no corresponde al cambio en la posición reportados.
Precondiciones	La relación de incremento/grados para los sensores de ascensión recta y declinación debe estar configurada. La ***posición actual*** debe estar inicializada. La comunicación entre el sistema y los sensores debe estar activa.
Poscondiciones	La variable de ***posición actual*** se deba actualizar de acuerdo al incremento obtenido y debe reportar la posición actual real.

Contrato	CU010-DS001-OP001
Nombre	Configura_motor(tipo_motor: cadena, velocidad_baja: entero, velocidad_alta: entero, velocidad_movimiento: entero, grados_paso: real, micropasos_paso: entero, sentido: entero)
Responsabilidades	Establecer los valores los valores característicos del motor para poder calcular los movimientos a objetos celestes.
Tipo / Referencias cruzadas / Salidas	
Notas	Tipo_motor: Especifica si es de declinación o ángulo horario. Por lo tanto habrá dos instancias del objeto motor. Velocidad_baja: Es una relación que especifica la velocidad baja de desplazamiento del motor, **(pasos x segundo)**, que será mayor o igual al limite mecánico del telescopio, dado por el motor y engrane en conjunto para una velocidad baja Velocidad_alta: Es una relación que especifica la velocidad alta de desplazamiento del motor, **(pasos x segundo)**, que será menor o igual al limite mecánico del telescopio, dado por el motor y engrane en conjunto para una velocidad alta Velocidad_alta: Es una relación que especifica la velocidad alta de desplazamiento normal del motor, **(pasos x segundo)**, que estará entre la velocidad baja y alta Grados_paso: Es una relación que especifica cuantos grados

	gira el telescopio en el eje del motor por paso **(grados/paso)**. Grados_paso: Especifica cuantos micropasos tiene un paso cuando se habilitan micropasos para el motor. Si no ha micropasos la relación es 1 **(micropaso/paso)** Sentido: especifica el sentido positivo de giro: 1: Sentido positivo es el horario -1: El sentido positivo es el antihorario Todas estas variables deben medirse y probarse físicamente antes de configurarlas.
Excepciones	No se pueden configurar los valores por errores en rango.
Precondiciones	----------
Poscondiciones	La instancia motor se actualiza con los nuevos valores.

Contrato	CU012-DS001-OP001
Nombre	Reinicializar_Cuenta_Decodificadores()
Responsabilidades	Reinicializar la cuenta interna de los decodificadores.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Decodificador*.
Poscondiciones	Se modificó la cuenta en *Decodificador.Tick*

Contrato	CU012-DS002-OP001
Nombre	Reinicializar_Posición_Telescopio()

Responsabilidades	Reinicializar las coordenadas que describen la posición actual del telescopio.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Decodificador*. Debe existir una instancia de *Telescopio*.
Poscondiciones	Se modificó las coordenadas en *Telescopio.AH* y *Telescopio.dec* Se modificó las cuenta en *Decodificar.Tick*

Contrato	CU013-DS001-OP001
Nombre	Prueba_Lectura_Dispostivo(Dispositivo)
Responsabilidades	Lee los datos de prueba al dispositivo E/S que sé específica, los datos son desplegados.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Decodificador*.
Poscondiciones	Se modificó la cuenta en *Decodificador.Tick*

Contrato	CU013-DS002-OP001
Nombre	Prueba_Escritura_Dispostivo(Dispositivo)
Responsabilidades	Envía datos de prueba al dispositivo E/S que sé específica, los datos son desplegados.
Tipo / Referencias	

cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Motor*.
Poscondiciones	Se modificaron los atributos *Motor.PasosH-F, Motor.Dirección, Motor.Numpasos*

Contrato	CU014-DS001-OP001
Nombre	Calibrar_Motores(pasos, H/F, Grados)
Responsabilidades	Establece la relación grados-pasos que utilizará el motor
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Motor*.
Poscondiciones	Se modificó la relación en *Motor.GradosPasos*

Contrato	CU014-DS002-OP001
Nombre	Calibrar_Decodificadores(pasos, H/F, Grados)
Responsabilidades	Establece la relación grados-pasos que utilizará el decodificador
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Decodificador*.
Poscondiciones	Se modificó la relación en *Decodificador.GradosPasos*

Contrato	CU015-DS001-OP001
Nombre	Encolar(AR, DEC, fecha, hora)
Responsabilidades	Encolar una petición de observación para mover el telescopio después.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Localidad* . Debe existir una instancia de *Usuario*.
Poscondiciones	Se creó una instancia de *coordenadas* pasando la AR y DEC Se creó una instancia de *Observación* Se asoció una *Observación* a *Usuario* Se asoció *coordenadas* a *Localidad* Se asoció las *coordenadas* a una *observación* Se modificó los atributos *Observación.AR*, *Observación.DEC*, *observación.ah, Observación.Fecha* y *Observación.Hora*

Contrato	CU016-DS001-OP001
Nombre	Alta_planeta_satelite (Id_planeta_satelite numérico, nombre_planeta_satelite cadena, execntricidad, distancia media al sol)
Responsabilidades	Registrar un nuevo planeta _satélite en el sistema.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se creara una instancia por cada planeta _satélite.

Excepciones	Que el planeta o satélite np ya exista
Precondiciones	
Poscondiciones	Se creo una nueva instancia de objeto planeta o satélite al asignarles valor a los nombre_planeta_satelite cadena, execntricidad_planeta_satelite numérico, dmsol_planeta_satelite numérica

Contrato	CU016-DS002-OP001
Nombre	Baja_ planeta_satelite (Id_planeta_satelite numérico)
Responsabilidades	Eliminar un objeto planeta_satelite.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Se borrara la instancia del objeto planeta_satelite en todos los contextos.
Excepciones	Que el objeto planeta_satelite no exista o haya sido borrado previamente.
Precondiciones	Que exista la instancia del objeto planeta_satelite.
Poscondiciones	Se borra o destruye la instancia del objeto planeta_satelite de acuerdo al Id_planeta_satelite,

Contrato	CU016-DS003-OP001
Nombre	Elegir_planeta_satelite_a_modificar(Id_planeta_satelite numérico)
Responsabilidades	Buscar el planeta_satelite a modificar de acuerdo a su identificador.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del objeto planeta_satelite es único.
Excepciones	Que el planeta_satelite a modificar no exista.
Precondiciones	El atributo planeta_satelite identificador ya es conocido.

Poscondiciones	Se crea la instancia del objeto planeta_satelite, la cual tiene los atributos que al corresponden a su identificador proporcionado.

Contrato	CU016-DS003-OP002
Nombre	Modificar_atributo_planeta_satelite (at_modificable tipo, at_nuevo tipo)
Responsabilidades	Elegir al atributo a modificar, proporcionar el nuevo valor del atributo y registrarlo.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	Los atributos a modificar serán los de la instancia planeta_satelite generada en el contrato CU016-DS003-OP001. El tipo de dato de los atributos a modificar (nuevo y anterior) deberán ser iguales.
Excepciones	Que el nuevo valor del atributo no cumpla con el formato del atributo, sea un valor duplicado (Id's) o sea un valor incoherente (Tipo de dato)
Precondiciones	Que la instancia planeta_satelite haya sido creada previamente, tenga inicializados sus atributos(, nombre_planeta_satelite cadena, excentricidad, distancia media al sol).
Poscondiciones	De la instancia planeta_satelite se elige el at_modificable, se verifica que el valor del at_nuevo coincida en tipo y se asigna el nuevo valor a la instancia.

Contrato	CU017-DS001-OP001
Nombre	Consultar_planeta_satelite (Id_planeta_satelite numérico)
Responsabilidades	Consultar los planetas o satélites que existen.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	El identificador del planeta_satelite es único, y esta operación es una especie de filtro para obtener los atributos de un planeta o satélite especifico.

Excepciones	Que el planeta_satélite no exista.
Precondiciones	El atributo planeta_satelite identificador ya es conocido.
Poscondiciones	Se obtiene el planeta_satelite de acuerdo al identificador id_ planeta_satelite numérico proporcionado.

Contrato	CU017-DS001-OP002
Nombre	Obtener_atributos_planeta_satelite (id_planeta_satelite numérico)
Responsabilidades	Obtener los atributos de un planeta o satélite de acuerdo al su id.
Tipo / Referencias cruzadas / Salidas	Sistema
Notas	
Excepciones	Que el planeta_satelite no exista.
Precondiciones	El atributo planeta_satelite identificador ya es conocido
Poscondiciones	Se obtienen los atributos del planeta_satelite de acuerdo al identificador id_ planeta_satelite numérico proporcionado.

Contrato	CU018-DS001-OP001
Nombre	Obtener_Coordenadas_Relativas(AR, DEC, Fecha, Hora)
Responsabilidades	Obtener las coordenadas relativas del planeta_satelite a observar (AH, DEC) a partir de coordenadas absolutas ingresadas por el usuario o por el catálogo, desplegar el resultado.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Localidad*.

	Debe existir una instancia de *Usuario*.
Poscondiciones	Se creó una instancia de *coordenadas* pasando la AR y DEC Se creó una instancia de *Observación* Se asoció una *Observación* a *Usuario* Se asoció *Coordenadas* a *Localidad* Se asoció las *coordenadas* a una *observación* Se modificó los atributos *Observación.AR*, *Observación.DEC*, *observación.ah, Observación.Fecha* y *Observación.Hora*

Contrato	CU018-DS001-OP002
Nombre	Determina Visibilidad
Responsabilidades	Determina si el planeta_satélite que se pretende observar es visible en ese momento con base en los límites de observación.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Telescopio*
Poscondiciones	

Contrato	CU018-DS002-OP001
Nombre	Obtener_Coordenadas_Relativas_Cola(ID_QE_Objeto_Celeste)
Responsabilidades	Obtener las coordenadas relativas de una Cola de observaciones pendientes y Desplegarlas.
Tipo / Referencias cruzadas / Salidas	

Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Cola de Observación* Deben existir elementos en la *Cola de Observación*
Poscondiciones	Se modificó la colección de objetos de *Cola de Observación*

Contrato	CU018-DS003-OP001
Nombre	Obtener_Coordenadas_Relativas_BD(ID_BD_Planeta_Satelite)
Responsabilidades	Obtener las coordenadas relativas de un Planeta_Satelite contenido en la Base de Datos y desplegarlos.
Tipo / Referencias cruzadas / Salidas	
Notas	
Excepciones	
Precondiciones	Debe existir una instancia de *Planeta_Satelite*
Poscondiciones	Se creó una instancia de *coordenadas* a partir de los atributos de *Planeta_Saltelite* Se creó una instancia de *Observación* Se asoció una *Observación* a *Usuario* Se asoció *Observación* a *Localidad* Se asoció las *coordenadas* a una *observación* Se modificó los atributos *Observación.AR*, *Observación.DEC*, *observación.ah, Observación.Fecha* y *Observación.Hora*

Diagramas de Colaboración

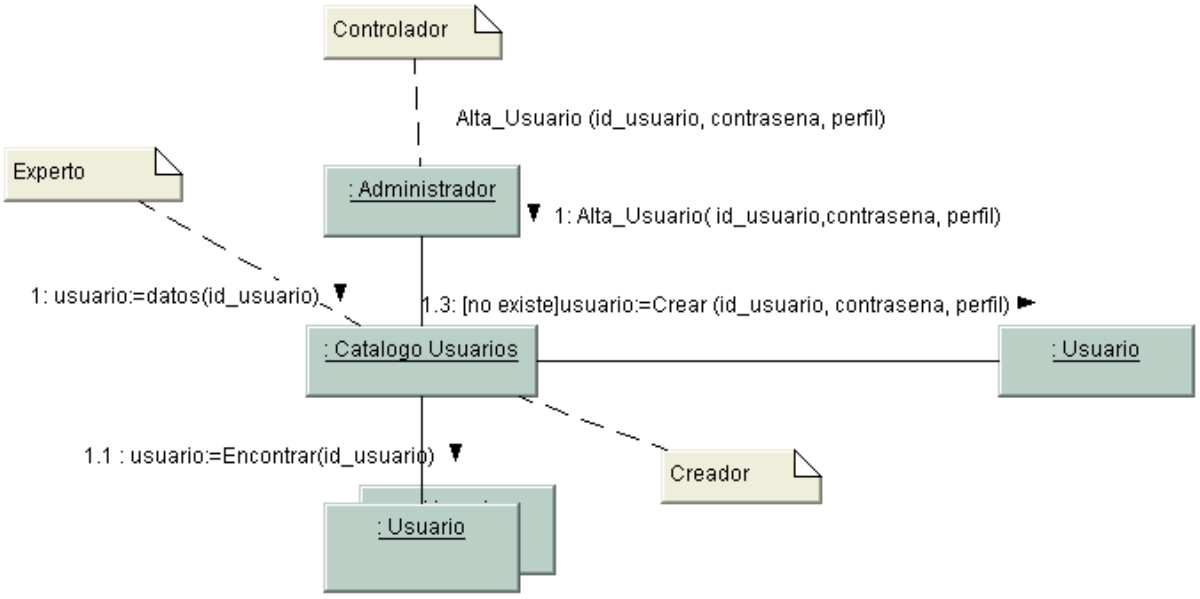


Diagrama de colaboración según el contrato CU001-DS001-OP001

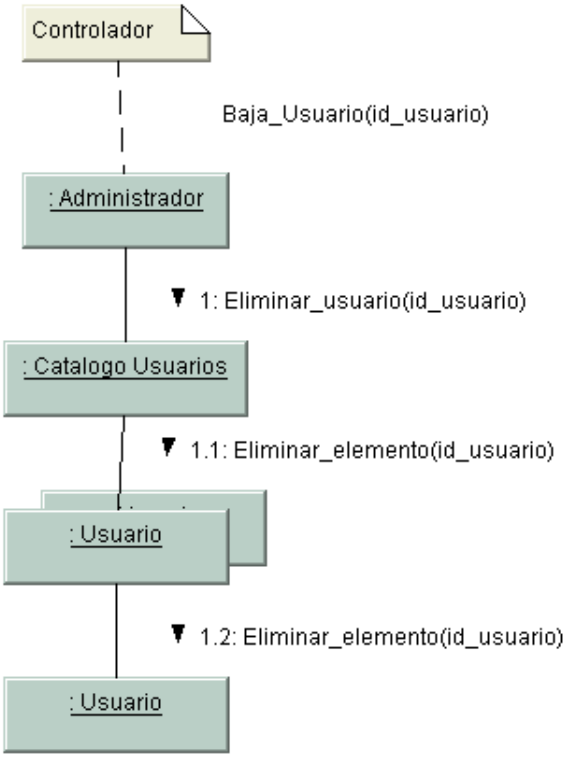


Diagrama de colaboración según el contrato CU001-DS002-OP001

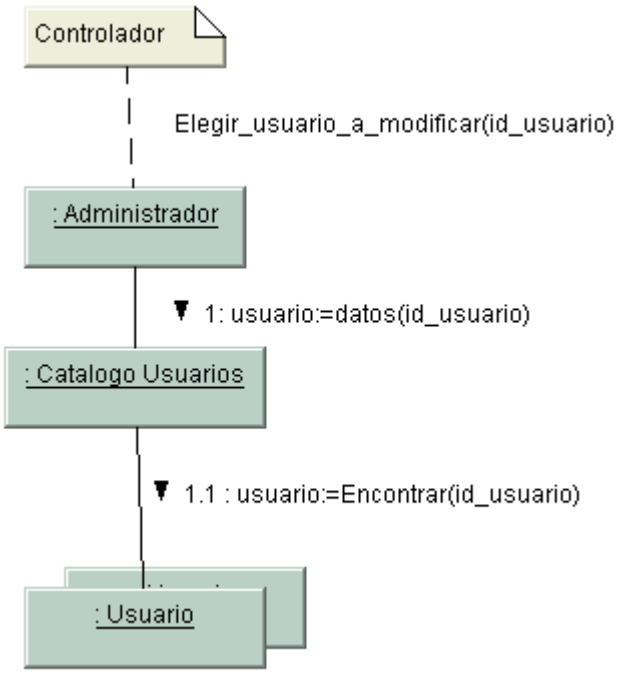


Diagrama de colaboración según el contrato CU001-DS003-OP001

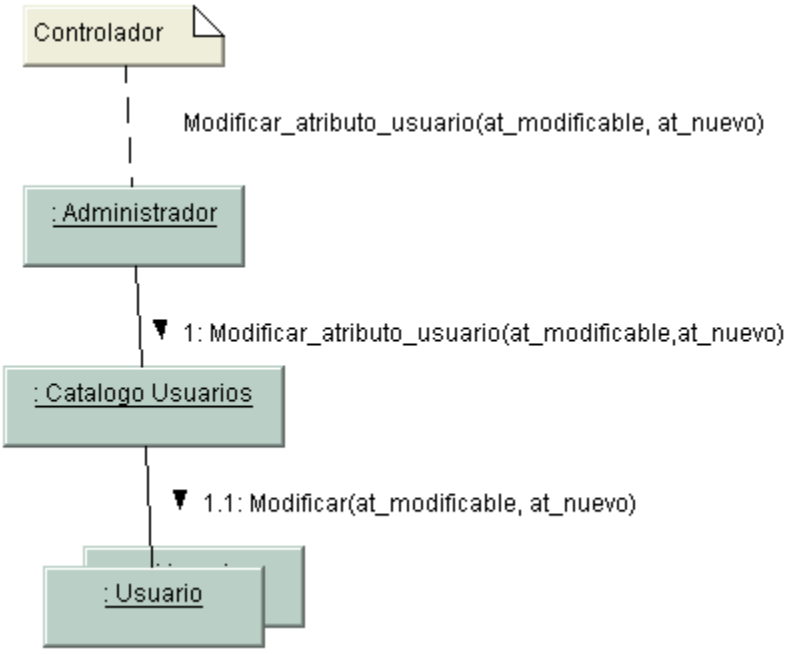


Diagrama de colaboración según el contrato CU001-DS003-OP002

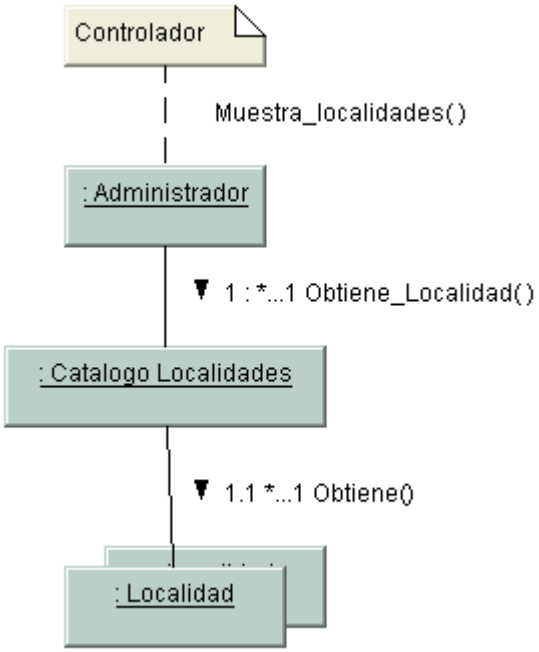


Diagrama de colaboración según el contrato CU002-DS001-OP001

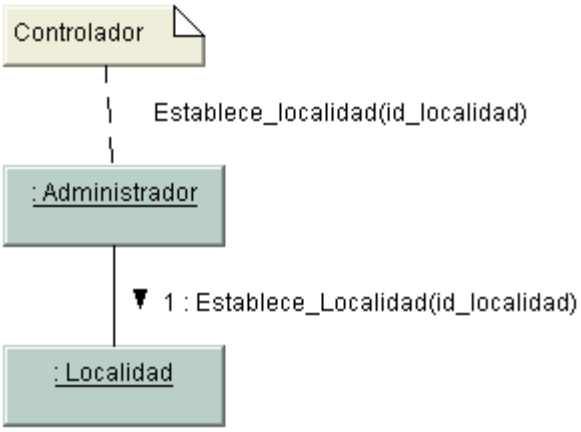


Diagrama de colaboración según el contrato CU002-DS001-OP002

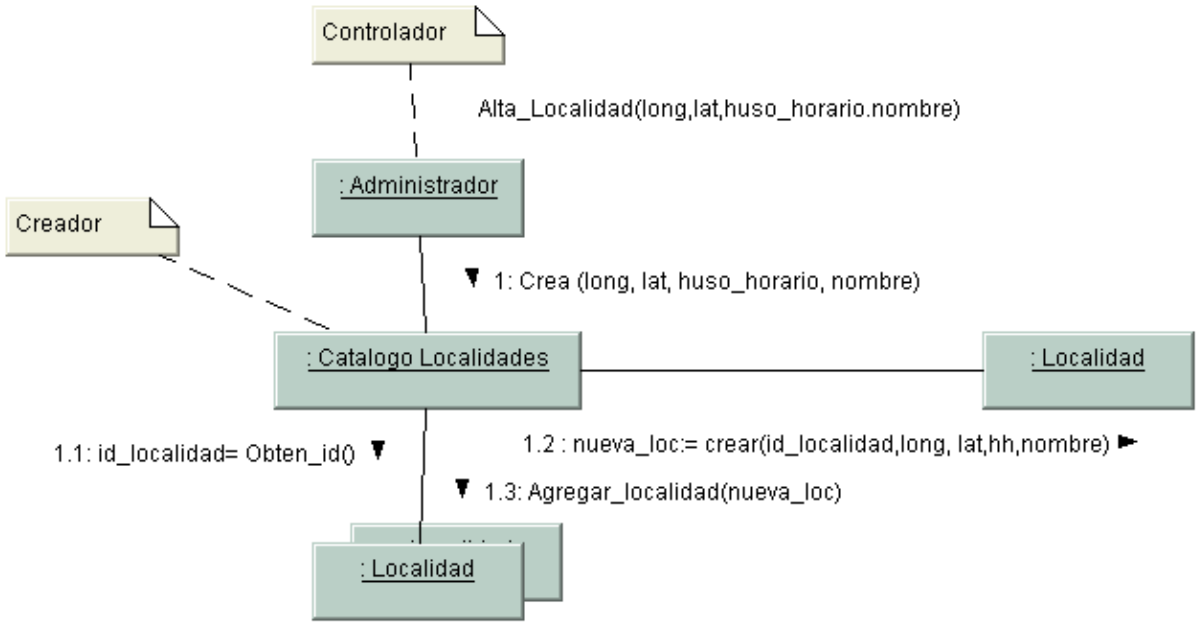


Diagrama de colaboración según el contrato CU002-DS002-OP001

Diagrama de colaboración según el contrato CU002-DS003-OP001

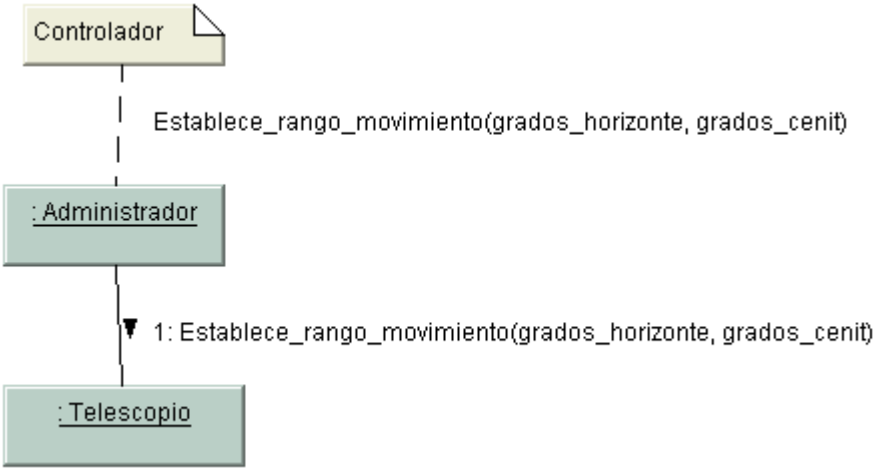


Diagrama de colaboración según el contrato CU002-DS004-OP001

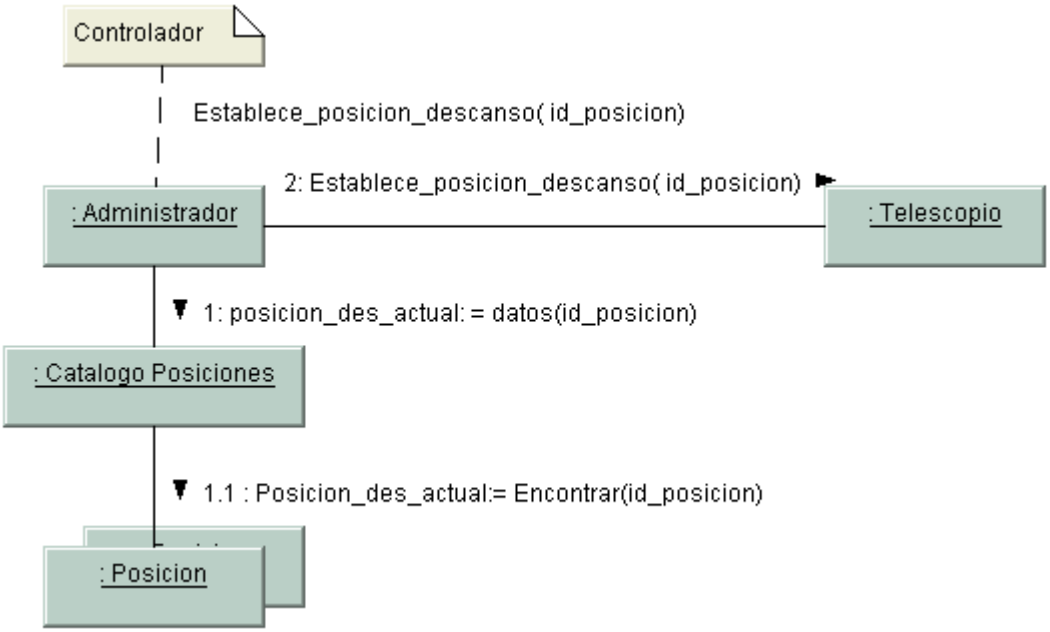


Diagrama de colaboración según el contrato CU002-DS005-OP001

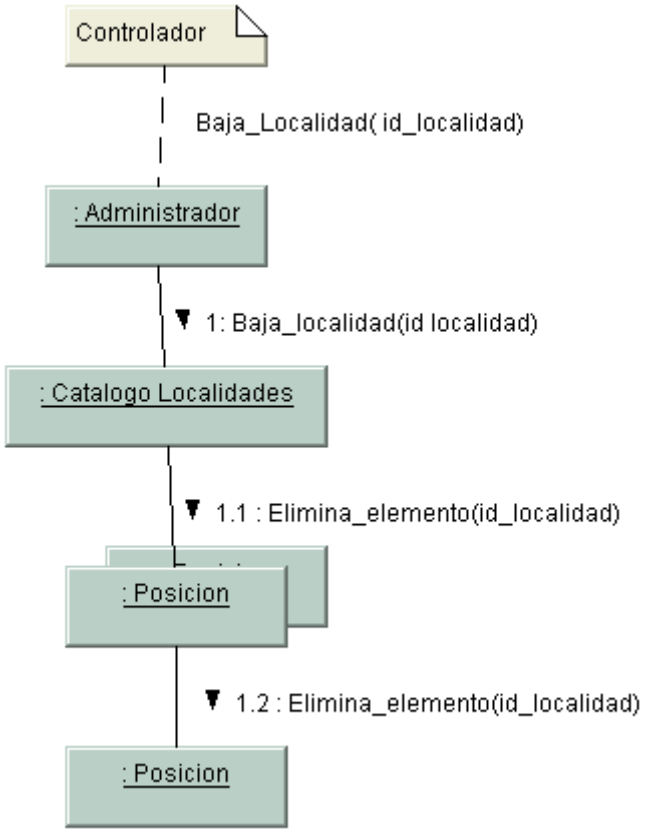


Diagrama de colaboración según el contrato CU002-DS006-OP001

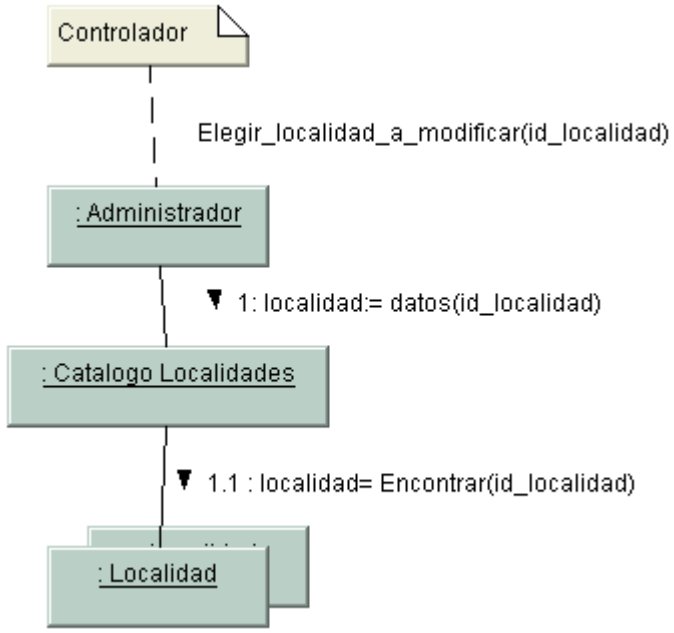


Diagrama de colaboración según el contrato CU002-DS007-OP001

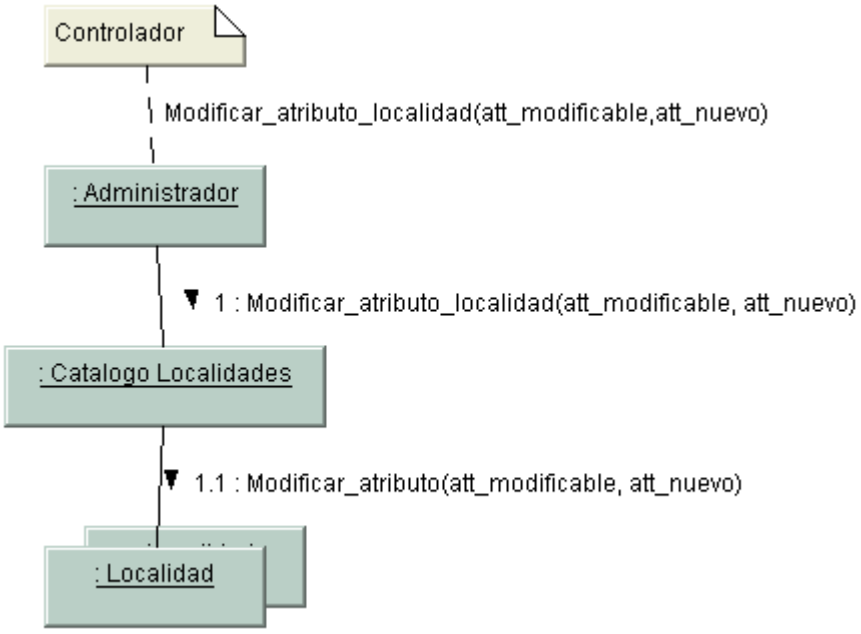


Diagrama de colaboración según el contrato CU002-DS007-OP002

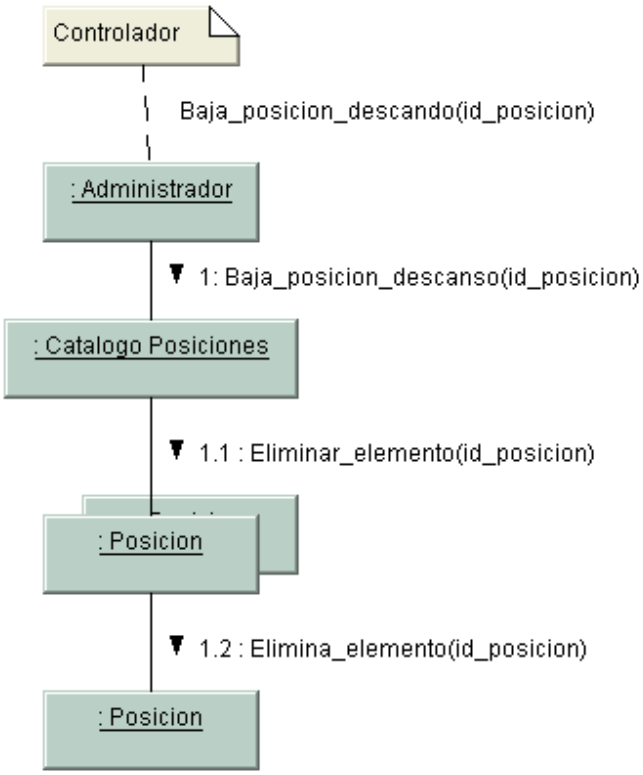


Diagrama de colaboración según el contrato CU002-DS008-OP001

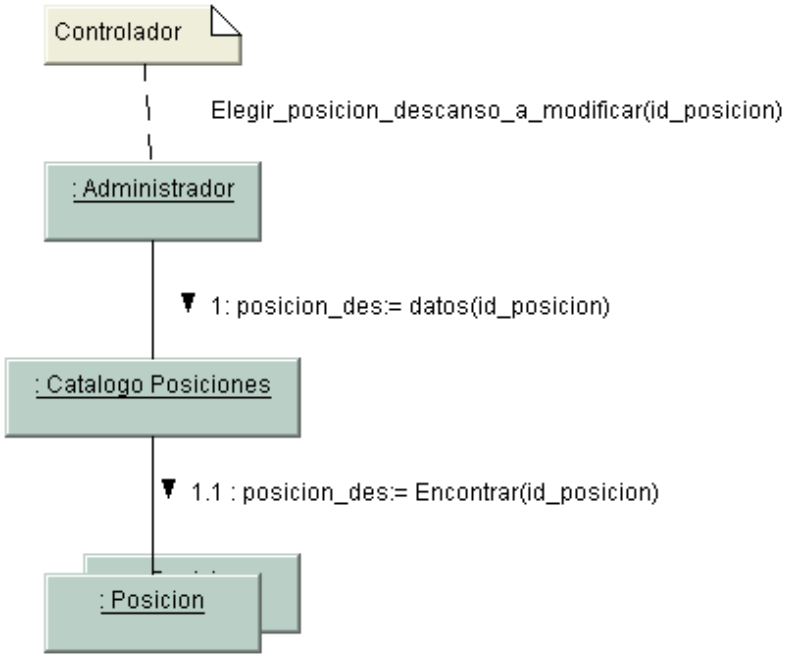


Diagrama de colaboración según el contrato CU002-DS009-OP001

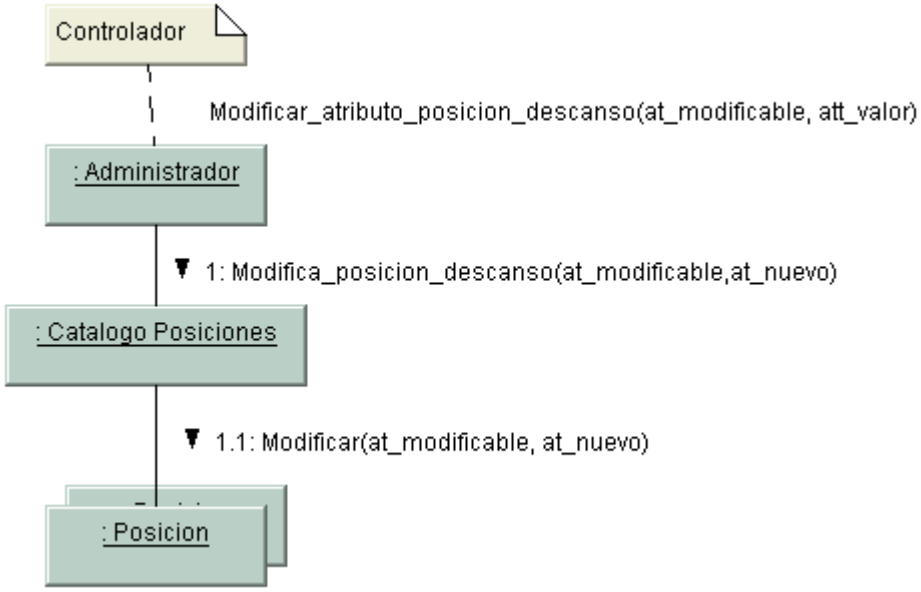


Diagrama de colaboración según el contrato CU002-DS009-OP002

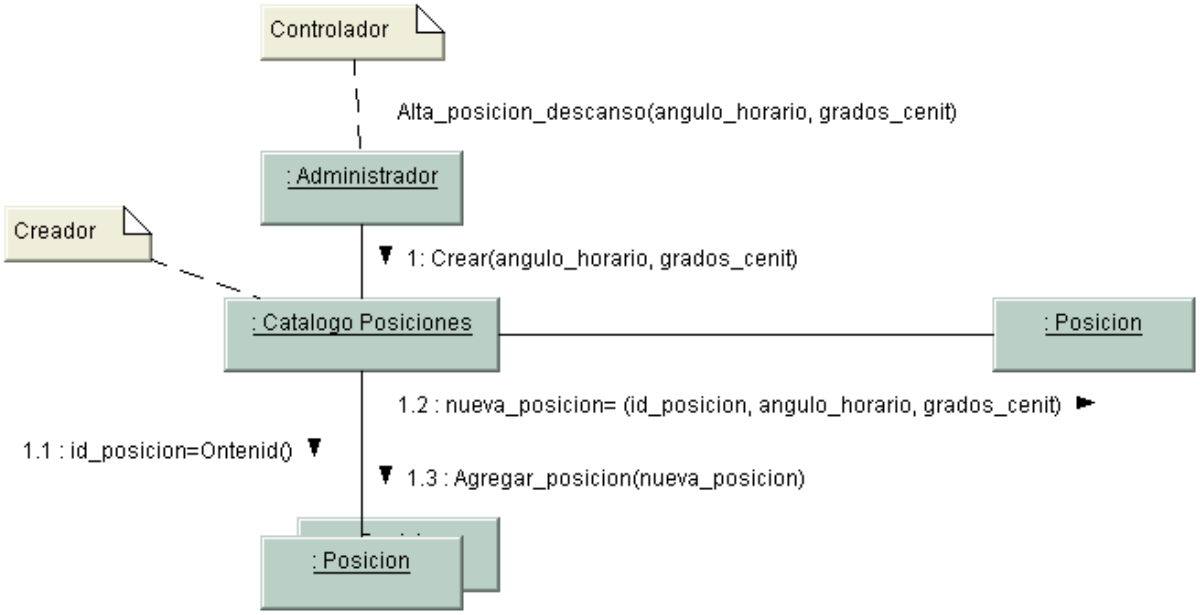


Diagrama de colaboración según el contrato CU002-DS010-OP001

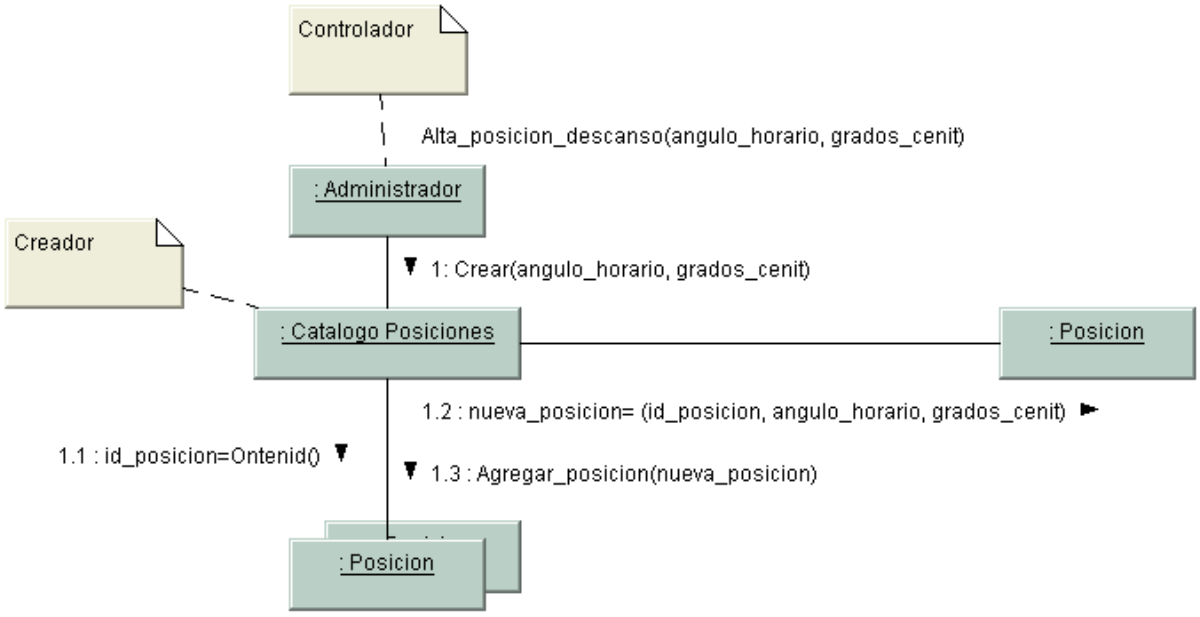


Diagrama de colaboración según el contrato CU002-DS010-OP002

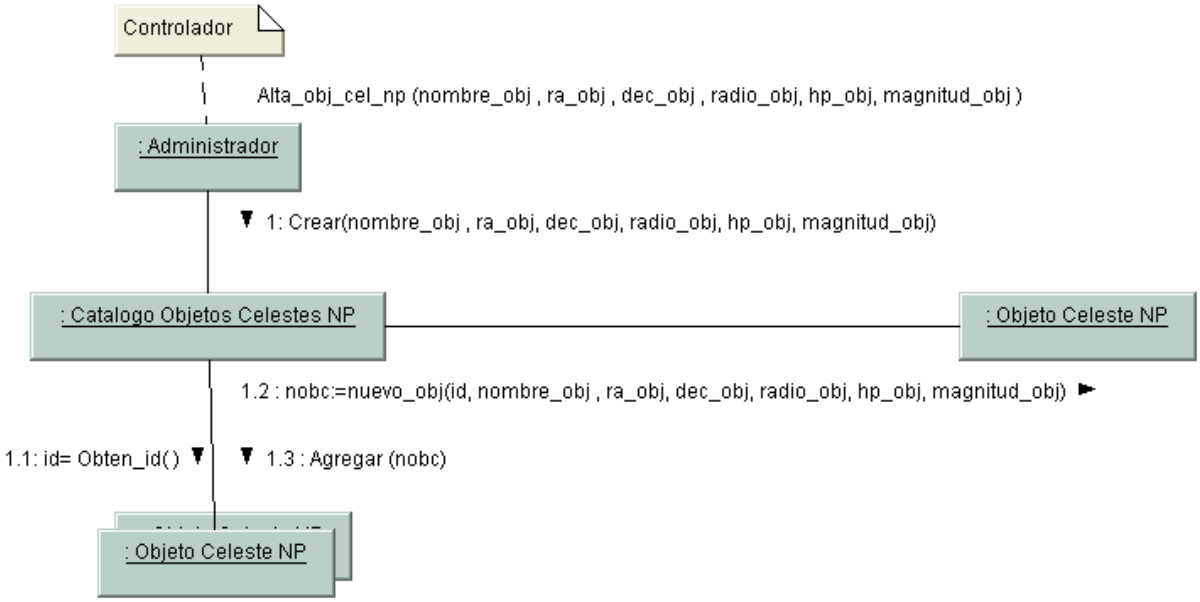


Diagrama de colaboración según el contrato CU003-DS001-OP001

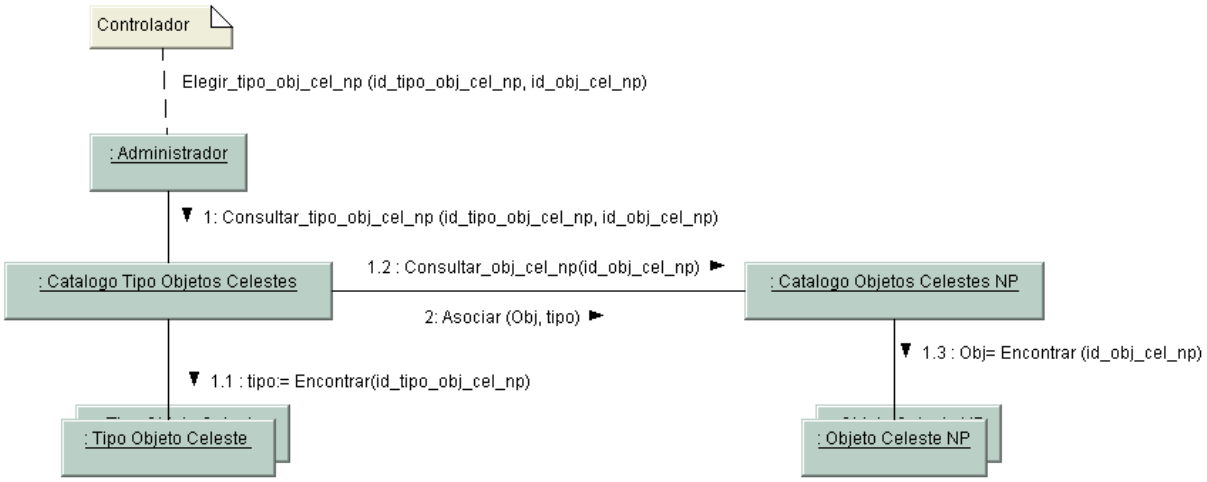


Diagrama de colaboración según el contrato CU003-DS001-OP002

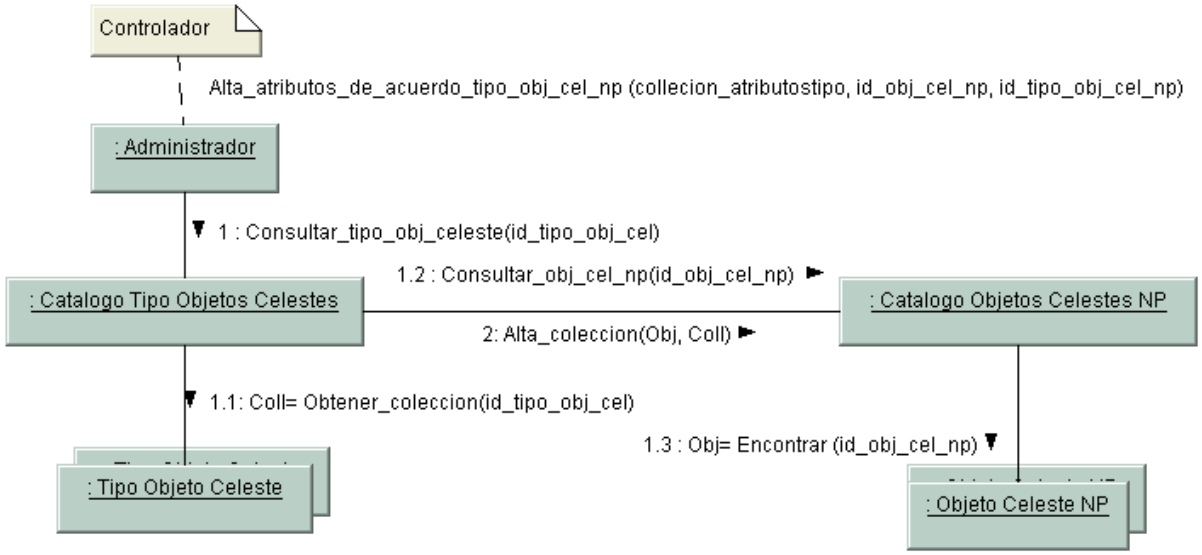


Diagrama de colaboración según el contrato CU003-DS001-OP003

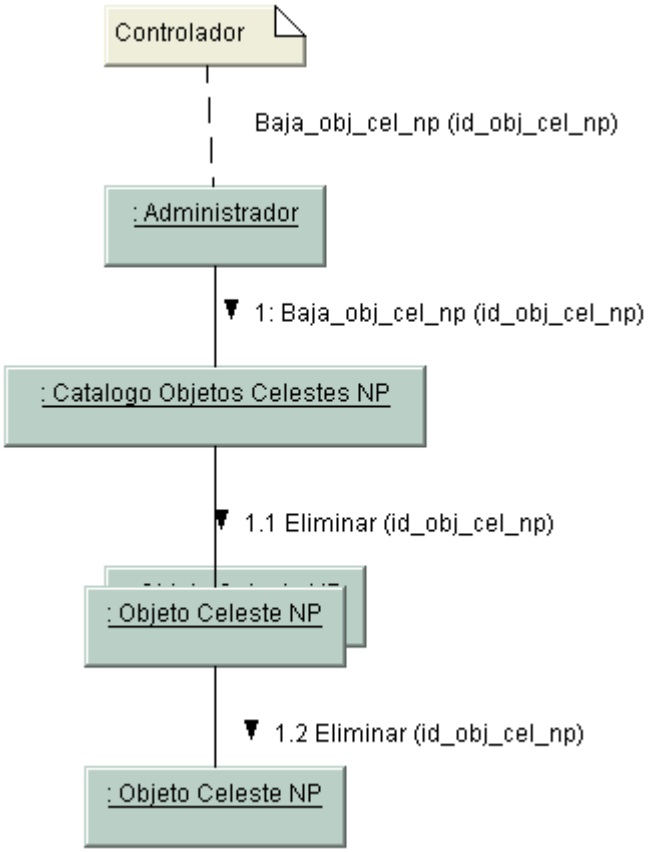


Diagrama de colaboración según el contrato CU003-DS002-OP001

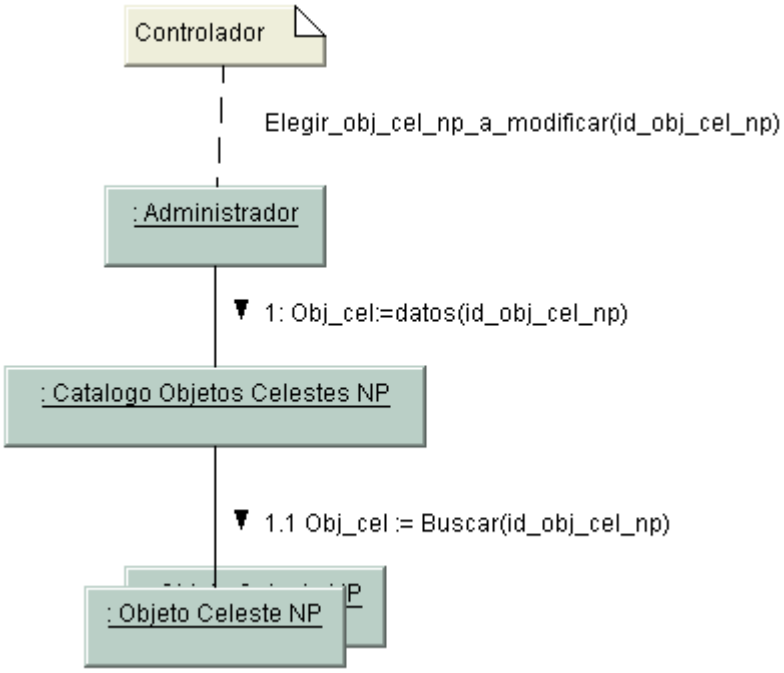


Diagrama de colaboración según el contrato CU003-DS003-OP001

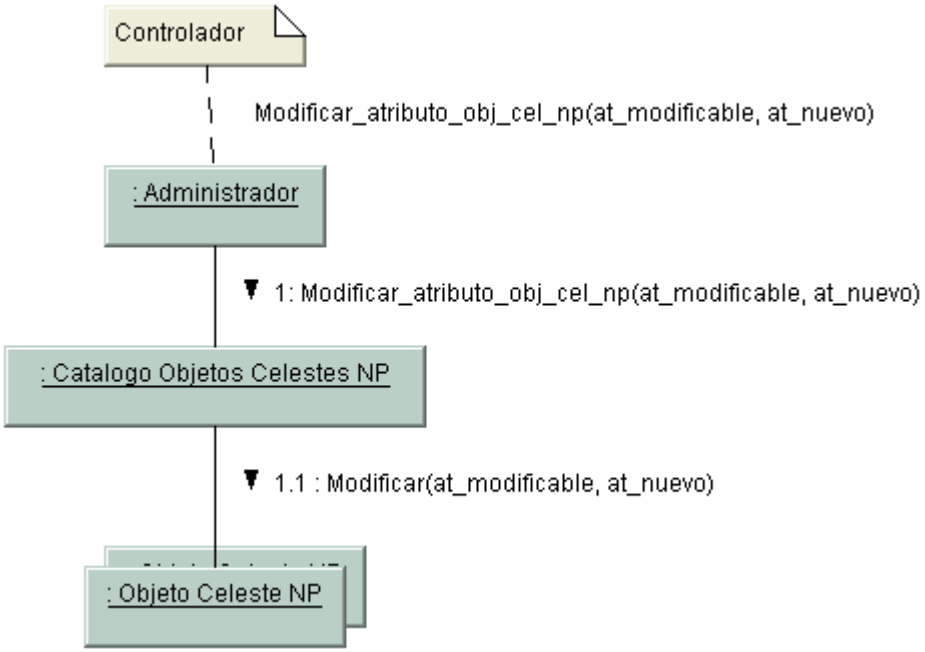


Diagrama de colaboración según el contrato CU003-DS003-OP002

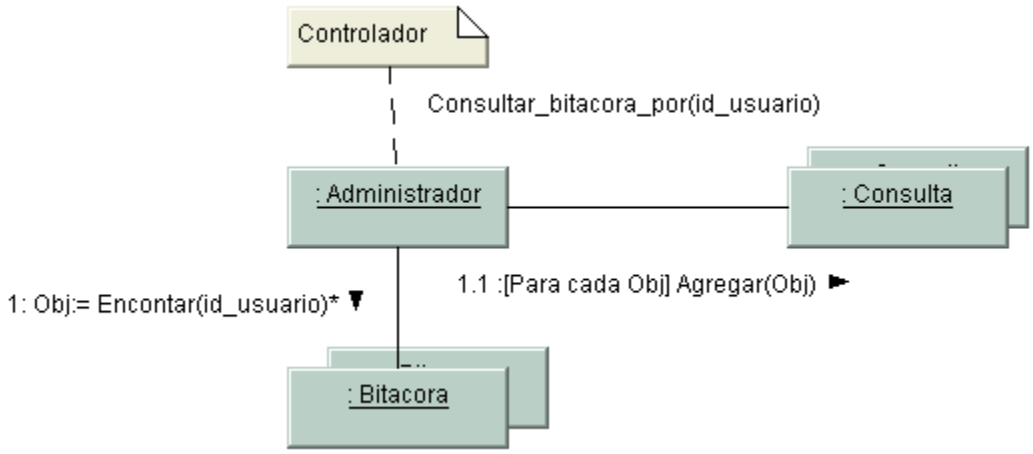


Diagrama de colaboración según el contrato CU004-DS001-OP001

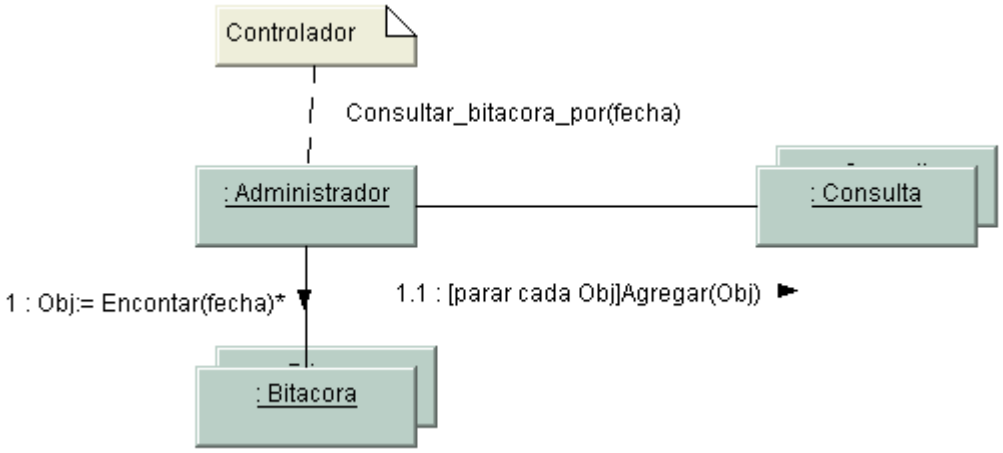


Diagrama de colaboración según el contrato CU004-DS002-OP001

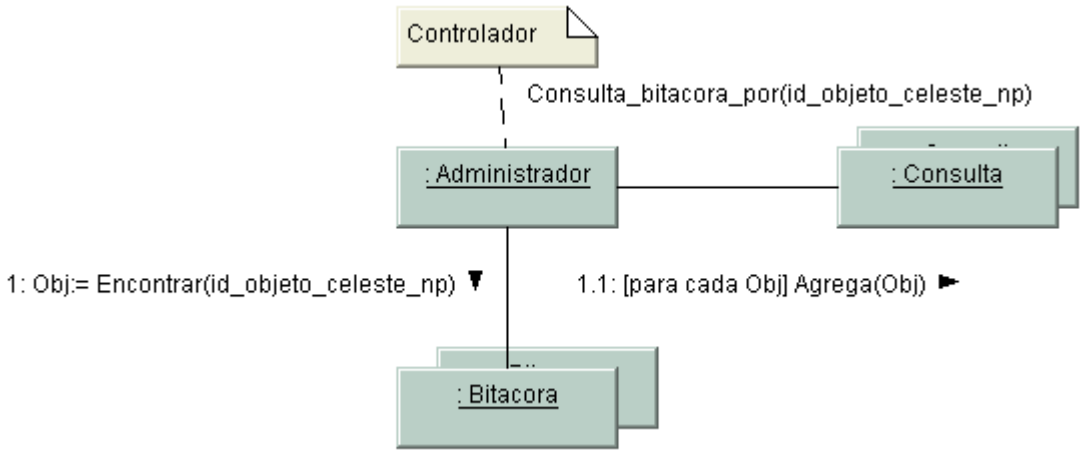


Diagrama de colaboración según el contrato CU004-DS003-OP001

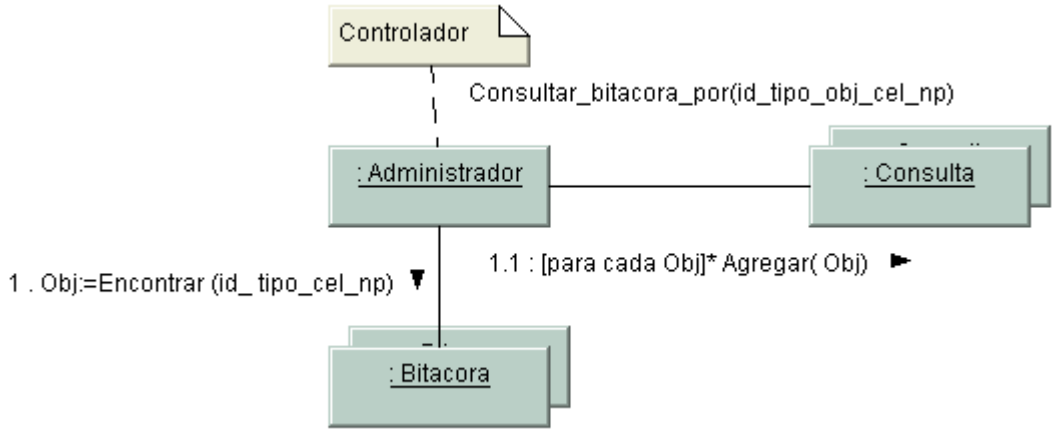


Diagrama de colaboración según el contrato CU004-DS004-OP001

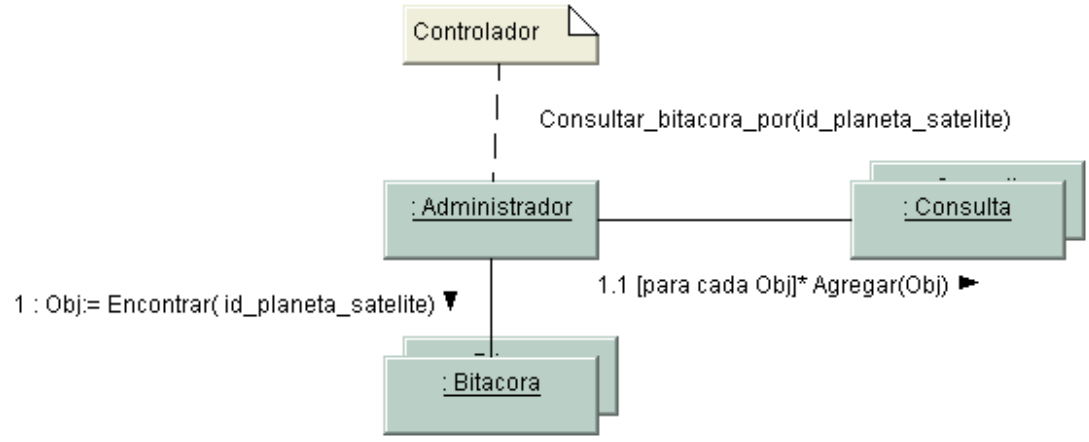


Diagrama de colaboración según el contrato CU004-DS005-OP001

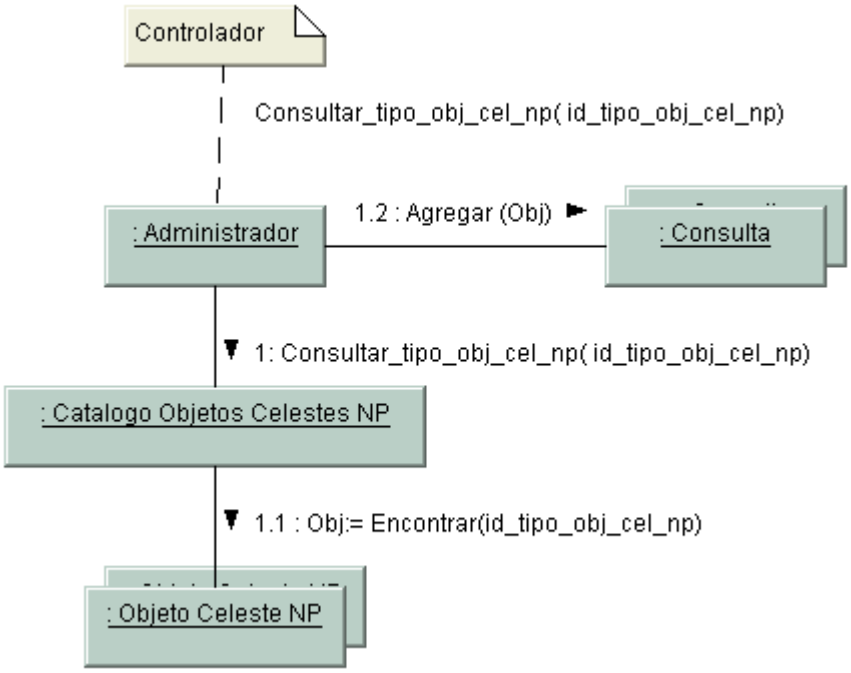


Diagrama de colaboración según el contrato CU005-DS001-OP001

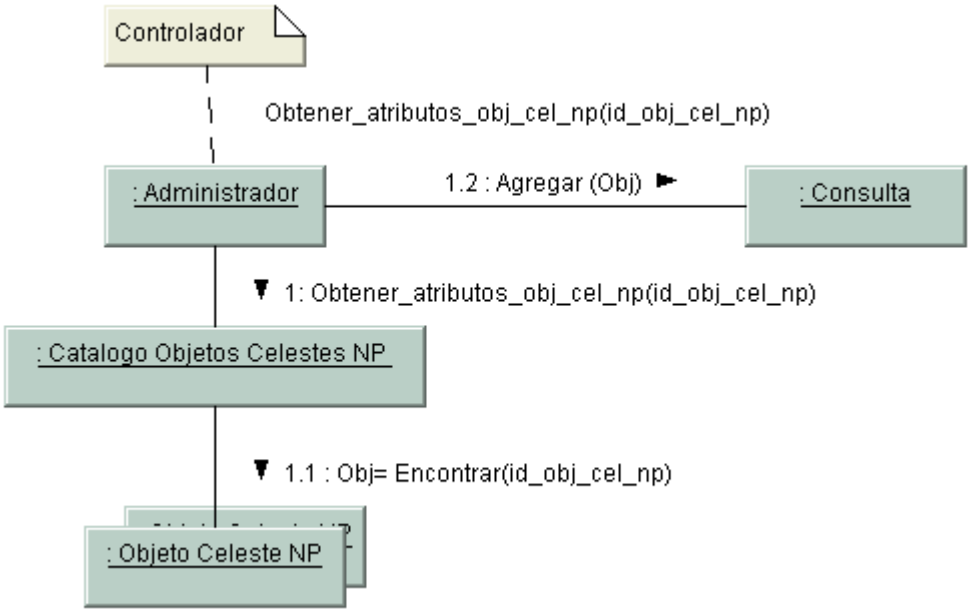


Diagrama de colaboración según el contrato CU005-DS001-OP002

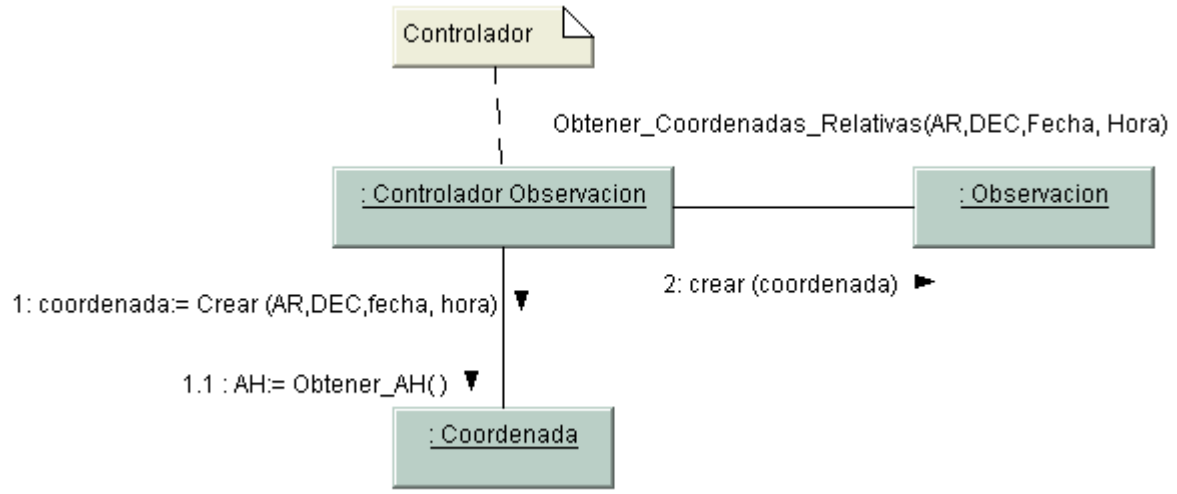


Diagrama de colaboración según el contrato CU006-DS001-OP001

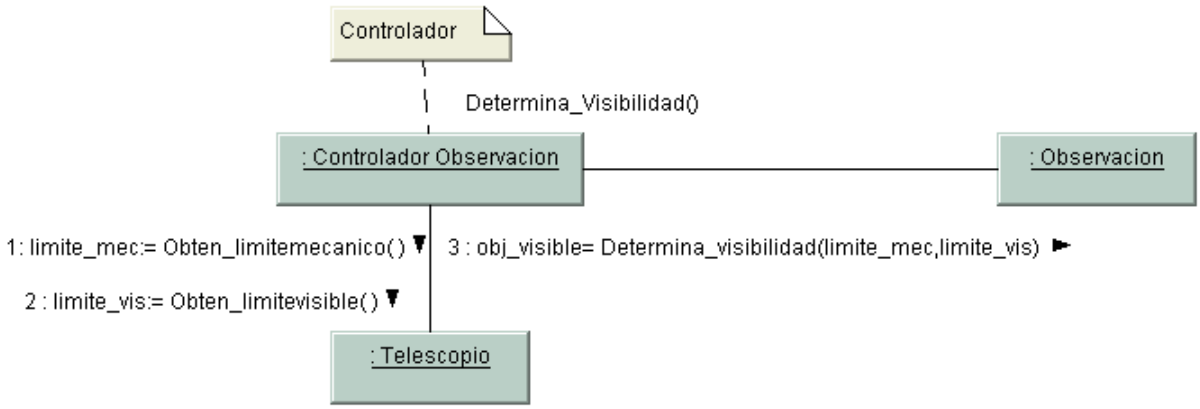


Diagrama de colaboración según el contrato CU006-DS001-OP002

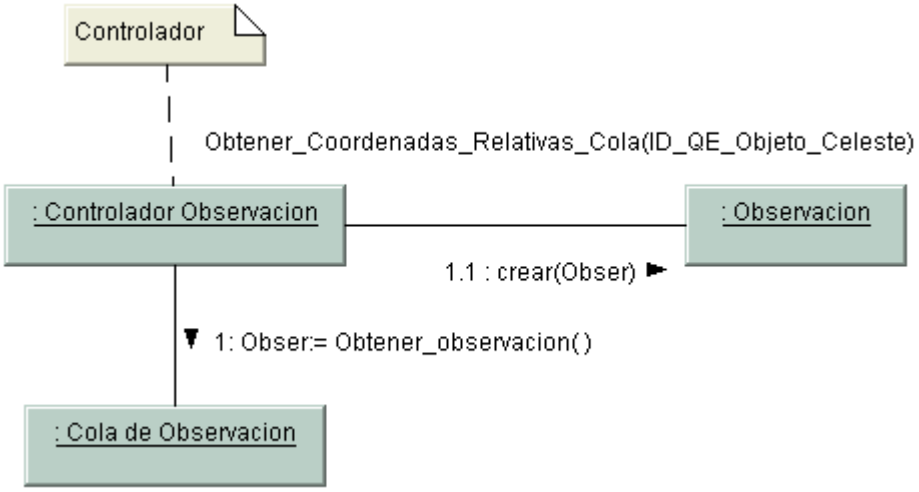


Diagrama de colaboración según el contrato CU006-DS002-OP001

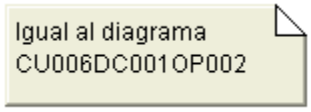


Diagrama de colaboración según el contrato CU006-DS002-OP002

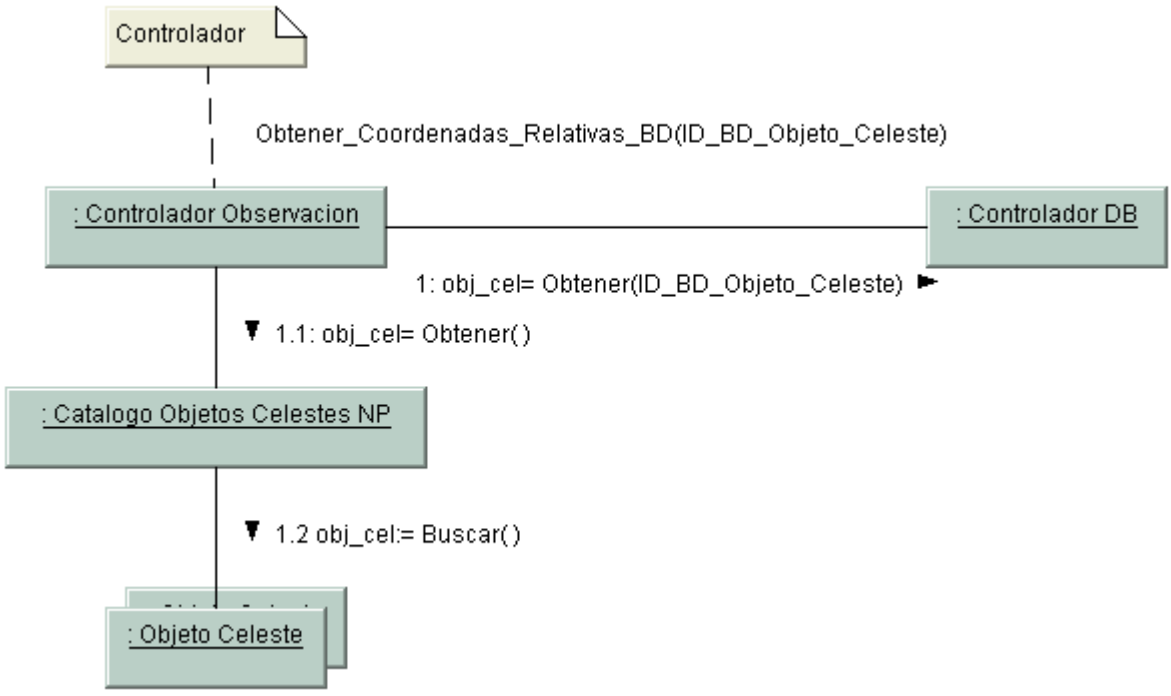


Diagrama de colaboración según el contrato CU006-DS003-OP001

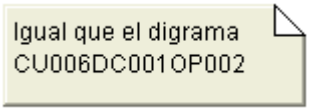


Diagrama de colaboración según el contrato CU006-DS003-OP002

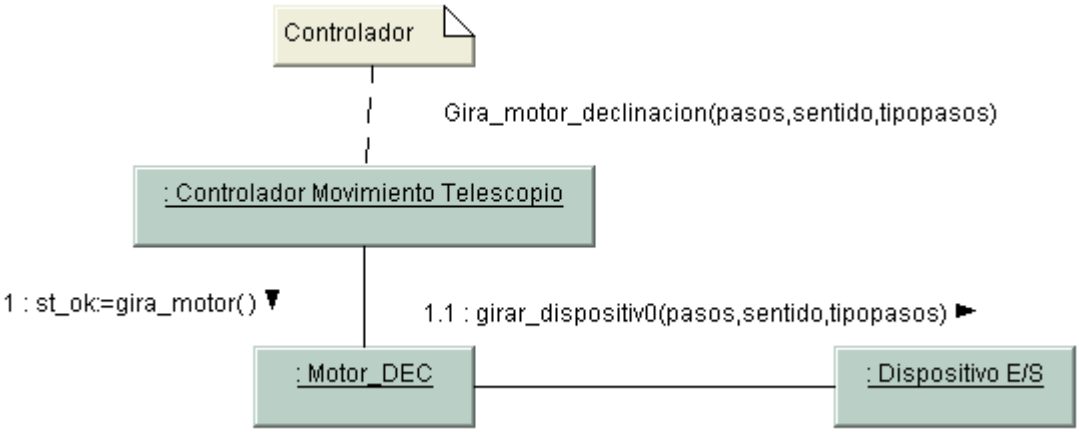


Diagrama de colaboración según el contrato CU008-DS001-OP001

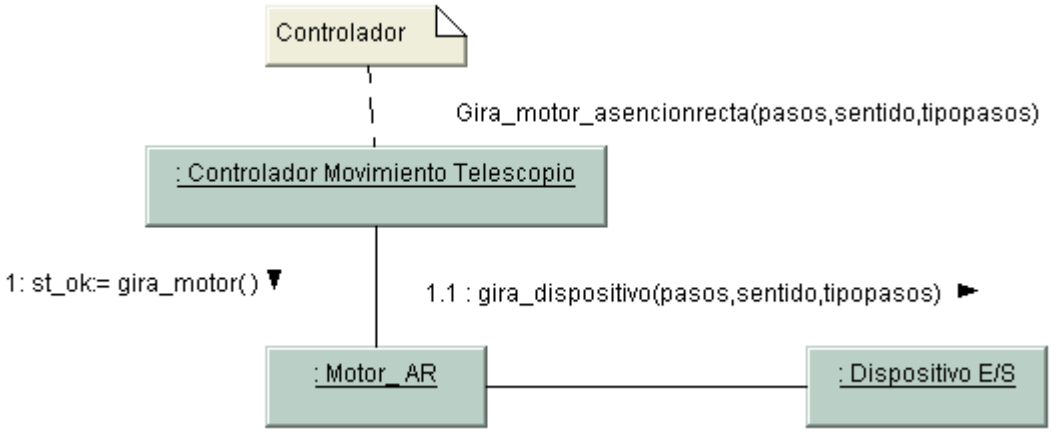


Diagrama de colaboración según el contrato CU008-DS002-OP001

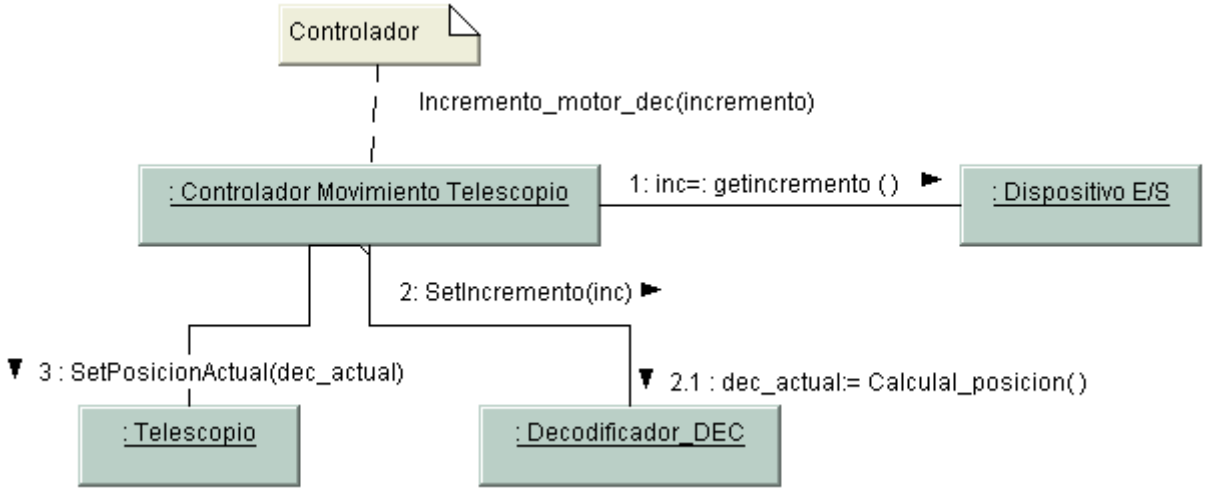


Diagrama de colaboración según el contrato CU009-DS001-OP001

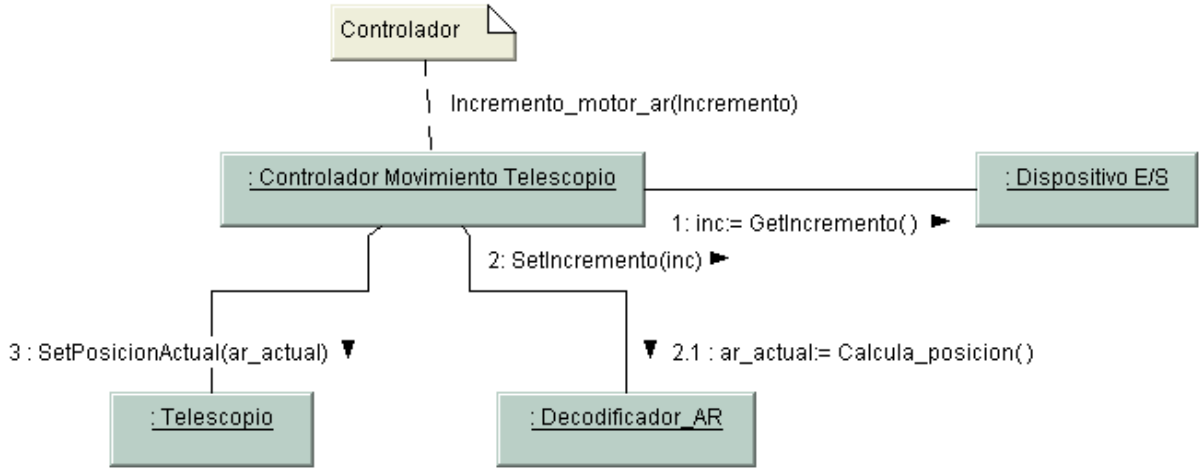


Diagrama de colaboración según el contrato CU009-DS002-OP001

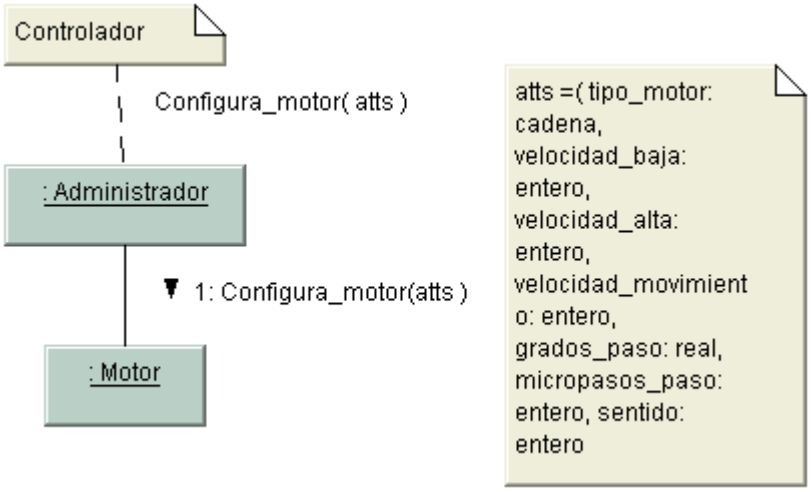


Diagrama de colaboración según el contrato CU010-DS001-OP001

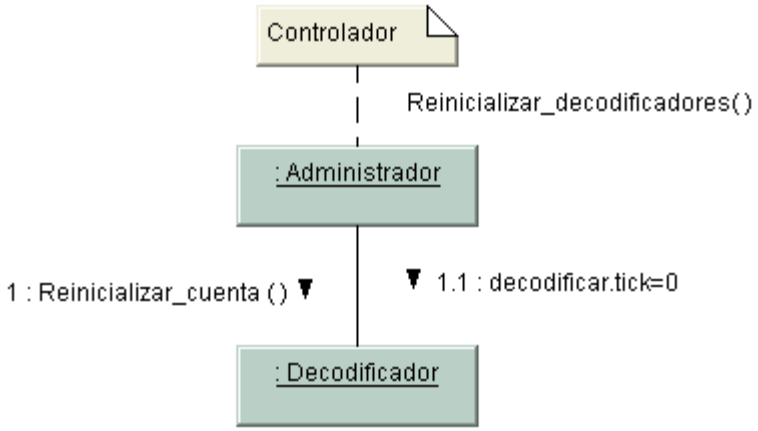


Diagrama de colaboración según el contrato CU012-DS001-OP001

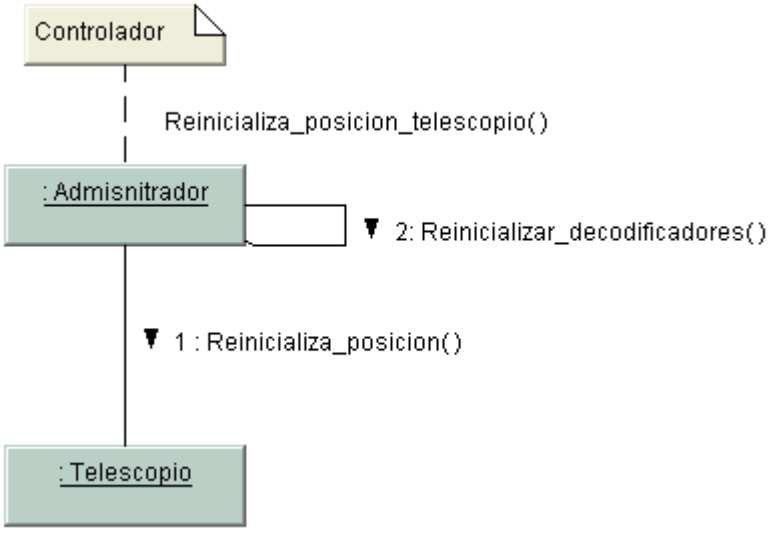


Diagrama de colaboración según el contrato CU012-DS002-OP001

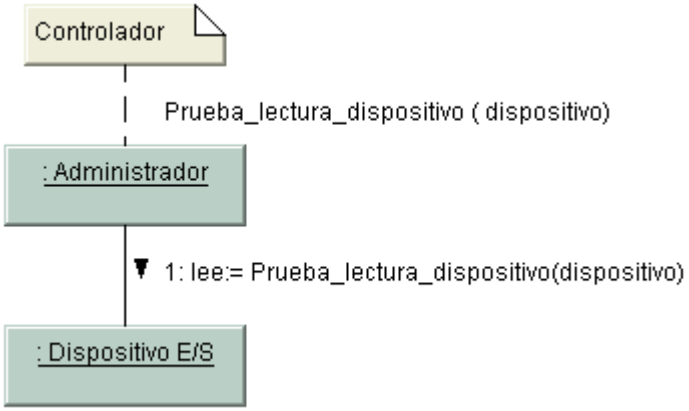


Diagrama de colaboración según el contrato CU013-DS001-OP001

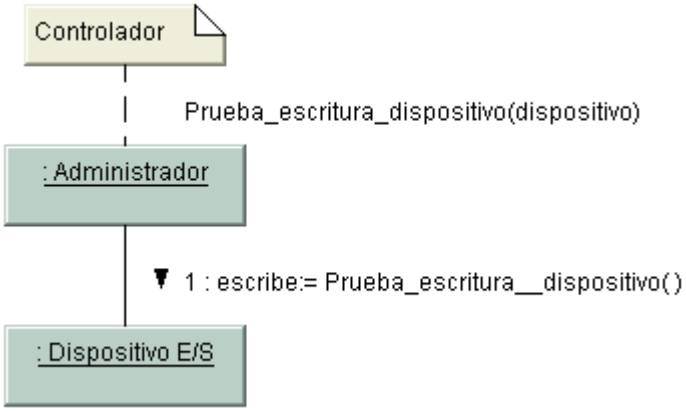


Diagrama de colaboración según el contrato CU013-DS002-OP001

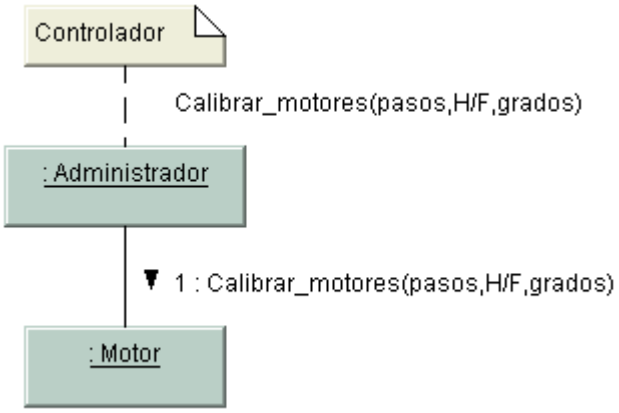


Diagrama de colaboración según el contrato CU014-DS001-OP001

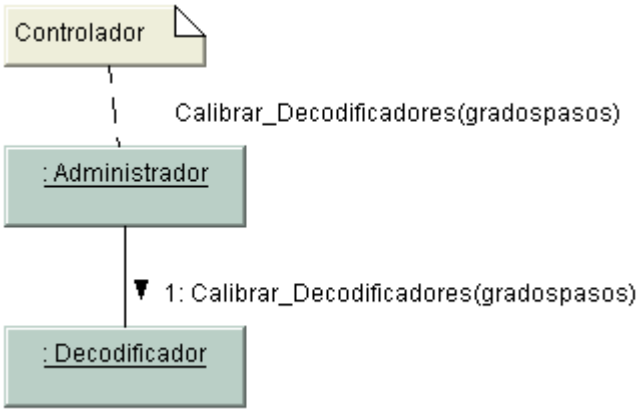


Diagrama de colaboración según el contrato CU014-DS002-OP001

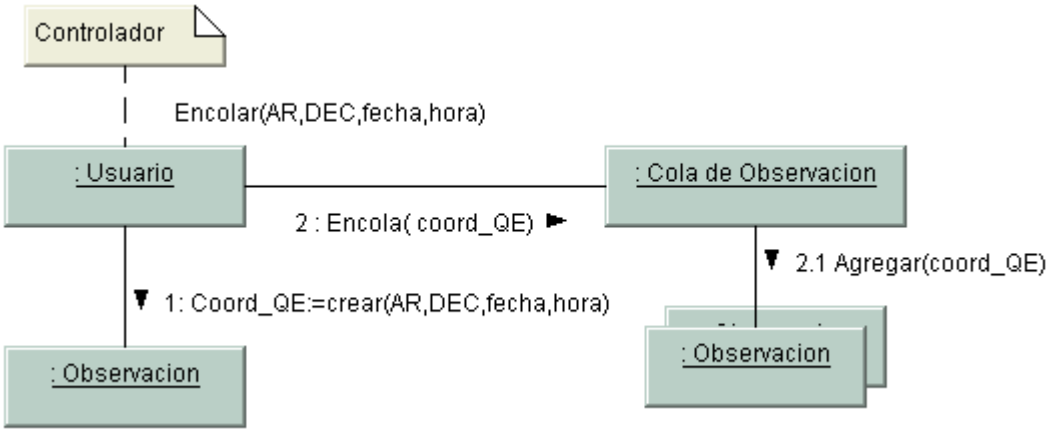


Diagrama de colaboración según el contrato CU015-DS001-OP001

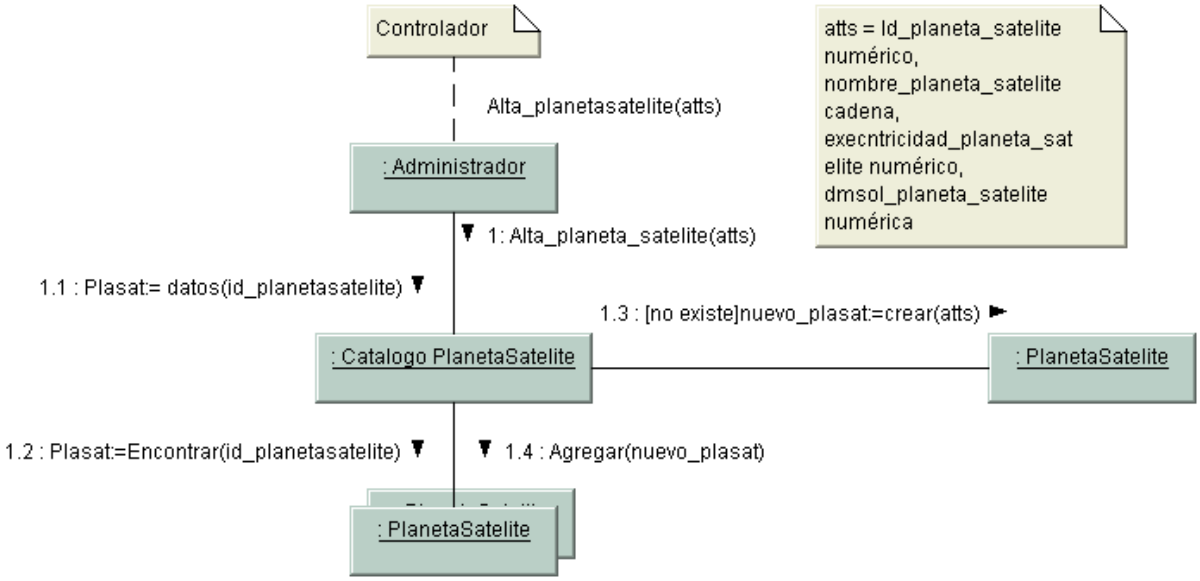


Diagrama de colaboración según el contrato CU016-DS001-OP001

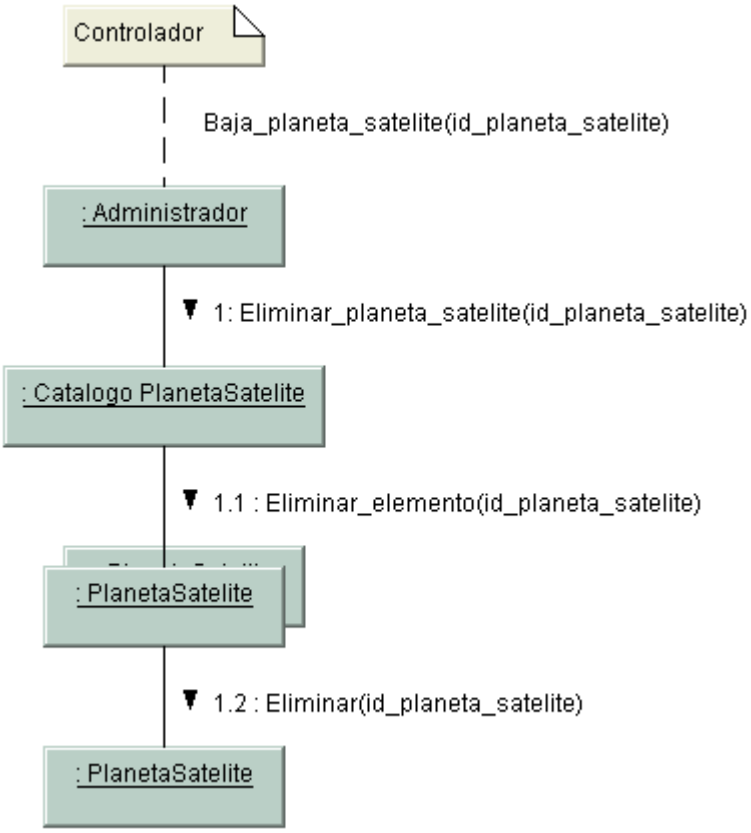


Diagrama de colaboración según el contrato CU016-DS002-OP001

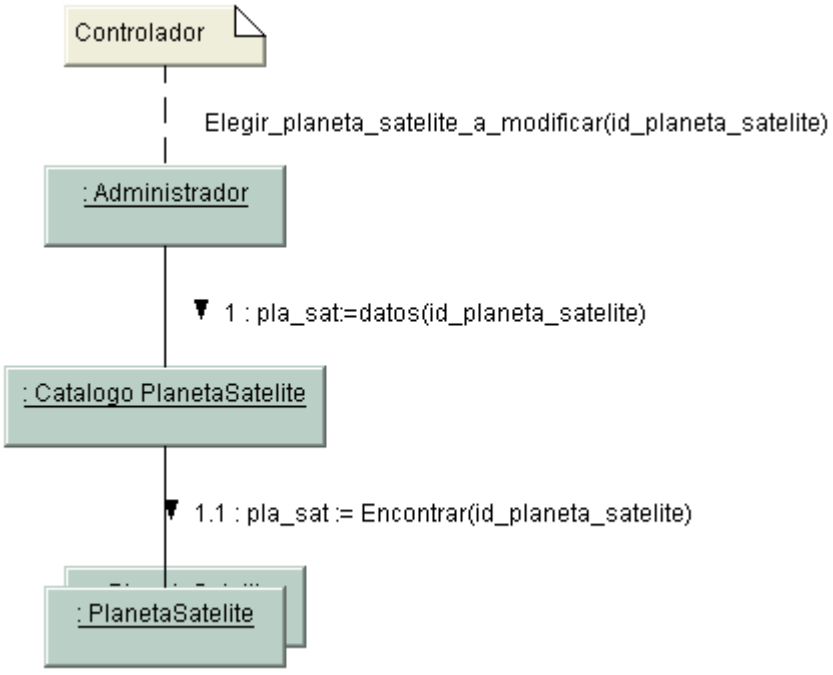


Diagrama de colaboración según el contrato CU016-DS003-OP001

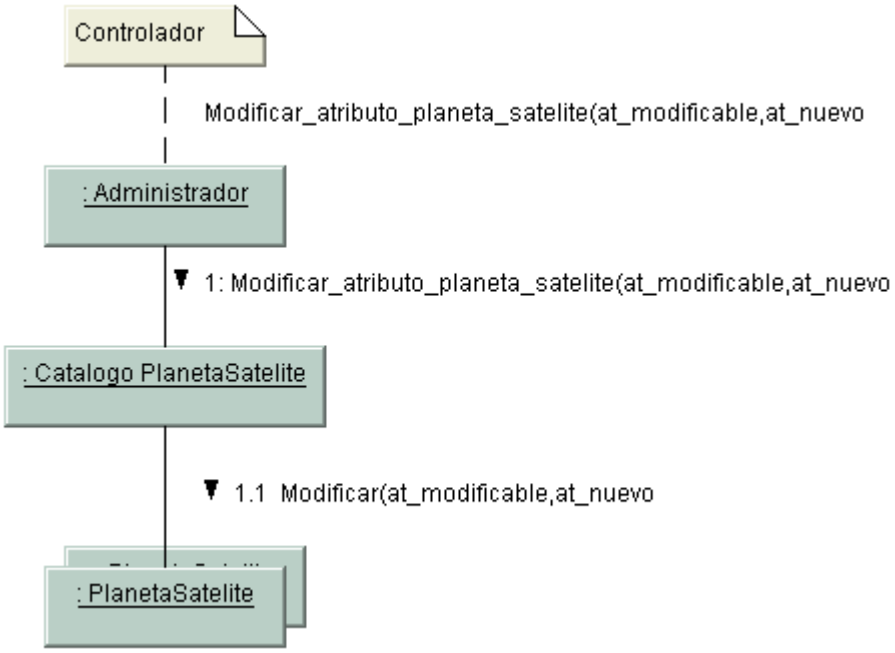


Diagrama de colaboración según el contrato CU016-DS003-OP002

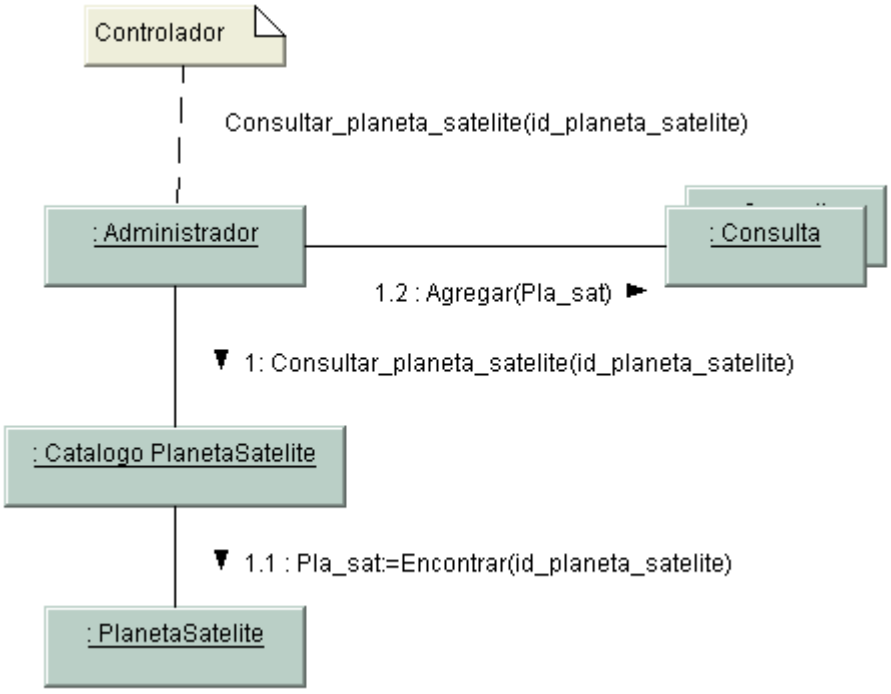


Diagrama de colaboración según el contrato CU017-DS001-OP001

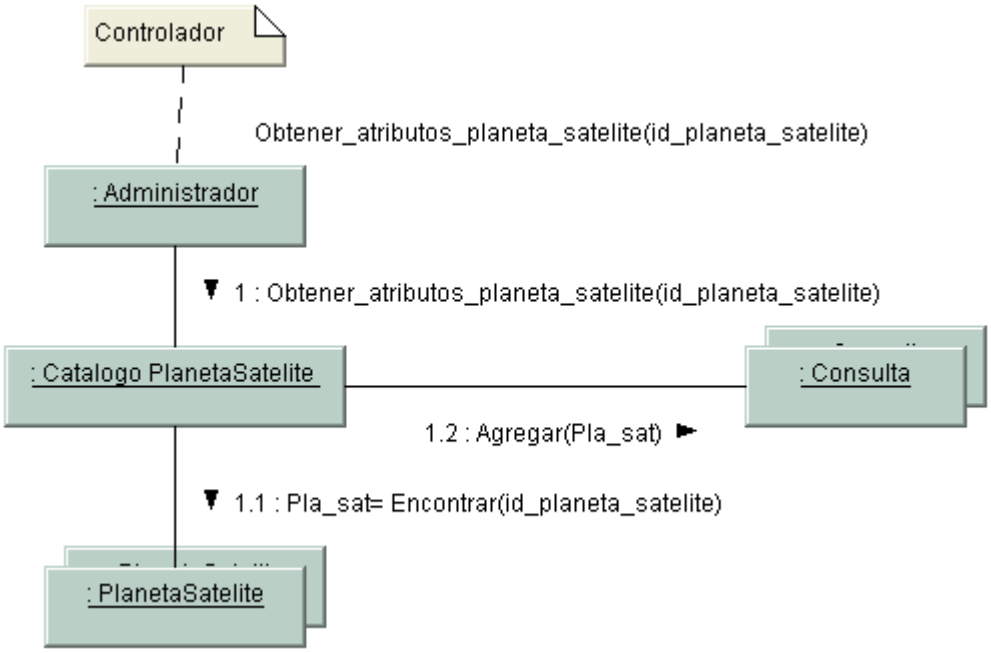


Diagrama de colaboración según el contrato CU017-DS001-OP002

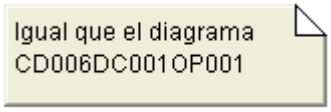


Diagrama de colaboración según el contrato CU018-DS00-OP001

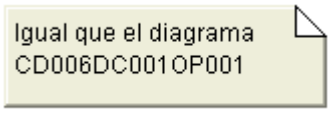


Diagrama de colaboración según el contrato CU018-DS001-OP002

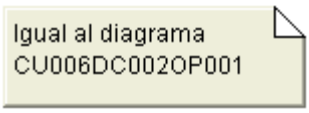


Diagrama de colaboración según el contrato CU018-DS002-OP001

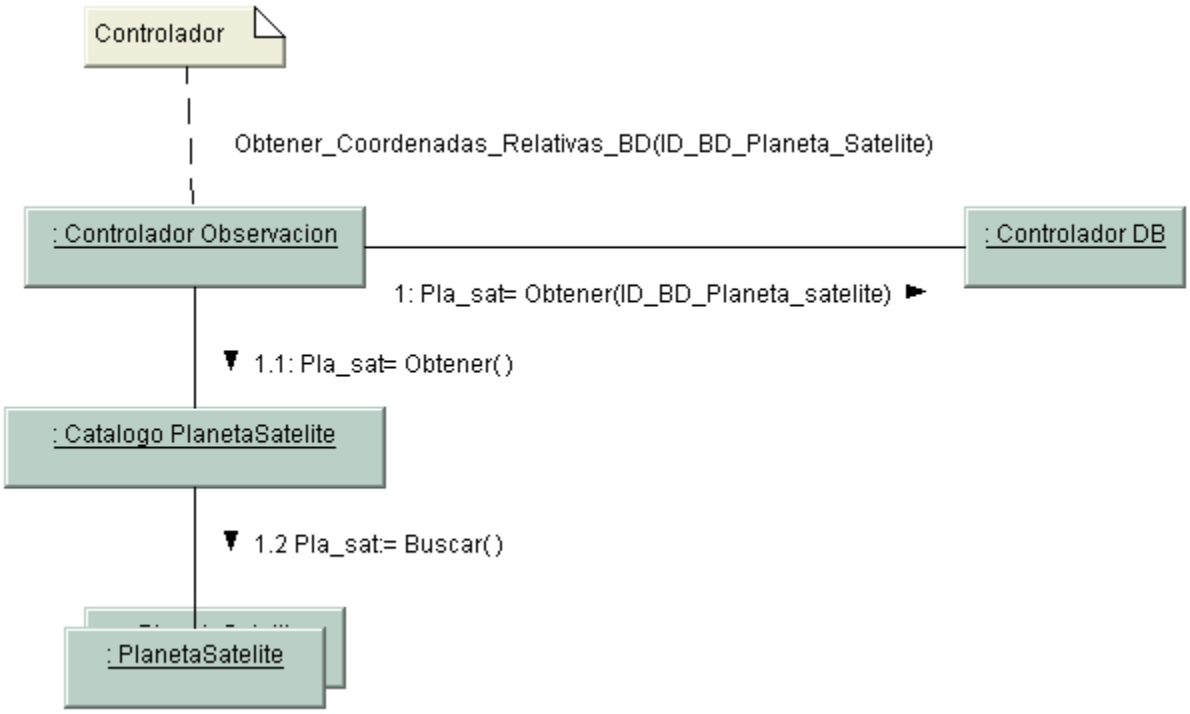


Diagrama de colaboración según el contrato CU018-DS003-OP001

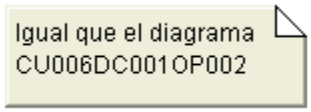


Diagrama de colaboración según el contrato CU018-DS003-OP00

Anexo C: Base de Datos

Diccionario de Datos

Tabla	Atributo	Tipo de dato	Opción NULL	Definición	PK	FK
Btcra	Lcldad_clve	SMALLINT	NULL	Clave numérica de la Localidad.	No	Sí
	Objto_Clste_clve	INT	NULL	Clave del Objeto Celeste.	No	Sí
	Usrio_lgin	VARCHAR(20)	NULL	Clave tipo texto del usuario, con la que entrara al sistema.	No	Sí
	Btcra_hra	TIME	NULL	Hora que se realizo la observación	No	No
	Btcra_fcha	DATE	NULL	Fecha que se realizo la observación	No	No
	Btcra_clve	INT	NOT NULL	Clave de la Bitácora, como un índice.	Sí	No
Cnstlcion	Cnstlcion_nmntvo	VARCHAR(25)	NOT NULL	Nombre de la constelación.	No	No
	Cnstlcion_gntvo	VARCHAR(25)	NOT NULL	Genitivo de la constelación.	No	No
	Cnstlcion_dscrpcion	VARCHAR(255)	NULL	Descripción de la constelación.	No	No
	Cnstlcion_clve	TINYINT	NULL	Clave numérica de la constelación. Autoincremental	Sí	No
	Cnstlcion_abrvtra	CHAR(3)	NOT NULL	Abreviatura de la constelación.	No	No
Lclddes	Lcldad_lngtud_mntos	SMALLINT	NULL	Latitud de la Localidad, minutos.	No	No
	Lcldad_alttud	SMALLINT	NULL	Altitud de la Localidad.	No	No

	Lcldad_clve	SMALLINT	NOT NULL	Clave numérica de la Localidad. Autoincremental	Sí	No
	Lcldad_lngtud_grdos	SMALLINT	NULL	Longitud de la Localidad, grados.	No	No
	Lcldad_hrrio_vrno	SMALLINT	NULL	Horario verano, 1 si usa, 0 no usa.	No	No
	Lcldad_nmbre	VARCHAR(50)	NULL	Nombre dela Localidad.	No	No
	Lcldad_lttud_sgndos	SMALLINT	NULL	Latitud de la Localidad, segundos.	No	No
	Lcldad_lttud_mntos	SMALLINT	NULL	Latitud de la Localidad, minutos.	No	No
	Lcldad_lttud_grdos	SMALLINT	NULL	Latitud de la Localidad, grados.	No	No
	Lcldad_lngtud_sgndos	SMALLINT	NULL	Latitud de la Localidad, segundos.	No	No
	Lcldad_lngtud_ldo	CHAR(1)	NOT NULL	Nos indica el sentido de la longitud(Este E u Oeste)	No	No
	Lcldad_lttud_ldo	CHAR(1)	NOT NULL	Nos indica el sentido de la latitud (Norte N o Sur S)	No	No
Objto_Clste	Objto_Clste_tpo_objto	CHAR(1)	NULL	Tipo de Objeto celeste: estrella E, Galaxia G, cúmulo C, nebulosa N.	No	No
	Objto_Clste_NGC	SMALLINT	NULL	Numero del objeto en el NGC(New General catalog)	No	No
	Objto_Clste_nmbre	VARCHAR(20)	NULL	Nombre del objeto celeste.	No	No
	Objto_Clste_RA_hras	FLOAT	NULL	Horas ascensión recta del objeto celeste.	No	No
	Objto_Clste_RA_mntos	FLOAT	NULL	Minutos ascensión recta del objeto celeste.	No	No

	Objto_clste_RA_sgndos	FLOAT	NULL	Segundos ascensión recta del objeto celeste.	No	No
	Objto_Clste_tpo_espctral	VARCHAR(15)	NULL	Tipo espectral dela estrella.	No	No
	Objto_Clste_tpo_glxia	VARCHAR(15)	NULL	Si el objeto es una galaxia, describir su tupo, ejemplo, espiral, barra, etc.	No	No
	Objto_Clste_NBS	SMALLINT	NULL	Numero del objeto en el NBS(New Brilliant catalog)	No	No
	Objto_Clste_mgntud	TINYINT	NULL	Brillantes del objeto	No	No
	Objto_Clste_Mssier	SMALLINT	NULL	Clave numérica del catalogo Messier	No	Si
	Objto_Clste_clve	INT	NOT NULL	Clave del Objeto Celeste. Autoincrementable.	Si	No
	Objto_Clste_DEC_grdos	FLOAT	NULL	Grados declinación del objeto celeste.	No	No
	Cnstlcion_clve	TINYINT	NOT NULL	Clave numérica de la constelación.	No	Si
	Objto_Clste_DEC_mntos	FLOAT	NULL	Minutos declinación del objeto celeste.	No	No
	Objto_Clste_DEC_sgndos	FLOAT	NULL	Segundos declinación del objeto celeste.	No	No
	Objto_Clste_mrflgia	CHAR(10)	NULL	Morfología del Objeto celeste	No	No
Plnta	Plnta_dstncia_mdia_sol	FLOAT	NULL	Distancia media al sol	No	No
	Plnta_rdio_mdio	INT	NULL	Planeta radio medio.	No	No
	Plnta_excntricdad	FLOAT	NULL	Excentricidad del planeta.	No	No
	Plnta_msa	INT	NULL	Masa del planeta en Unidades astronómicas.	No	No

	Plnta_nmbre	VARCHAR(20)	NOT NULL	Nombre del Planeta.	No	No
	Plnta_prdo_rtcion	FLOAT	NULL	Periodo de rotación en días.	No	No
	Plnta_clve	TINYINT	NOT NULL	Clave numérica del Planeta. Autoincrementable.	Si	No
	Plnta_dnsdad	FLOAT	NULL	Densidad del planeta.	No	No
	Plnta_dscrpcion	VARCHAR(255)	NULL	Descripción del Planeta.	Si	Si
Plnta_Crdndas	Plnta_vrcion_hrria_RA	FLOAT	NULL	Variación horaria de la ascensión recta.	No	No
	Plnta_mes	TINYINT	NOT NULL	Fecha del año, mes.	Si	No
	Plnta_RA_hras	SMALLINT	NULL	ascensión recta del planeta, horas.	No	No
	Plnta_RA_mntos	SMALLINT	NULL	ascensión recta del planeta, minutos	No	No
	Plnta_HPM_sgndos	SMALLINT	NULL	Hora de paso por el meridiano 90° W.G., segundos.	No	No
	Plnta_HPM_mntos	SMALLINT	NULL	Hora de paso por el meridiano 90° W.G., minutos.	No	No
	Plnta_HPM_hras	SMALLINT	NULL	Hora de paso por el meridiano 90° W.G., horas.	No	No
	Plnta_RA_sgndos	SMALLINT	NULL	Ascensión recta del planeta, segundos.	No	No
	Plnta_DEC_grdos	SMALLINT	NULL	Declinación, grados del planeta.	No	No
	Plnta_vrcion_hrria_DEC	FLOAT	NULL	Edad en días, solo aplica para la luna.	No	No

	Plnta_DEC_mntos	SMALLINT	NULL	Declinación, minutos del planeta.	No	No
	Plnta_DEC_sgndos	SMALLINT	NULL	Declinación, segundos del planeta.	No	No
	Plnta_DG	FLOAT	NULL	Distancia geocéntrica.	No	No
	Plnta_dia	TINYINT	NOT NULL	Fecha del año, día,	Sí	No
	Plnta_clve	TINYINT	NOT NULL	paralaje horizontal Lunar	No	No
Plnta_Crdndas_eclptcas	Plnta_dia	TINYINT	NOT NULL	Fecha del año, día.	Sí	No
	Plnta_mes	TINYINT	NOT NULL	Fecha del año, mes.	No	No
	Plnta_clve	TINYINT	NOT NULL	Clave numérica del Planeta.	No	Sí
	Plnta_dstncia_trra	INT	NULL	Distancia a la Tierra.	No	No
	Plnta_elngcion_plnta	INT	NULL	Elongación Planeta Sol.	No	No
	Plnta_elngcion_sntdo	CHAR(1)	NULL	Sentido de la elongación, Este (E) u Oeste (O)	No	No
	Plnta_lngtud_eclptca	FLOAT	NULL	Longitud Eclíptica del Planeta.	No	No
	Plnta_lttud_eclptca	FLOAT	NULL	Latitud Eclíptica del Planeta.	No	No
	Plnta_smdmtro_lnar	FLOAT	NULL	Semidiametro Lunar	No	No
	Plnta_prlje_hrzntal_lnar	INT	NULL	Paralaje horizontal Lunar	No	No
Pscion	Pscion_DEC_grdos	FLOAT	NULL	Grados, declinación de la posición	No	No
	Pscion_AGH_hras	FLOAT	NULL	Horas, Angulo Horario de la posición	No	No

	Pscion_clve	INT	NOT NULL	Clave de la posición.	Sí	No
	Pscion_AGH_mntos	FLOAT	NULL	Minutos, Angulo Horario de la posición	No	No
	Pscion_AGH_sgndos	FLOAT	NULL	Segundos, Angulo Horario de la posición	No	No
	Pscion_DEC_mntos	FLOAT	NULL	Minutos, declinación de la posición	No	No
	Pscion_DEC_sgndos	FLOAT	NULL	Segundos, declinación de la posición	No	No
Stlte	Stlte_nmbre	VARCHAR(20)	NOT NULL	Nombre del satélite.	No	No
	Stlte_msa	FLOAT	NULL	Masa.	No	No
	Stlte_excntrcdad	FLOAT	NULL	Excentricidad.	No	No
	Stlte_dstncia_plnta	FLOAT	NULL	Distancia al planeta.	No	No
	Stlte_clve	SMALLINT	NOT NULL	Clave numérica del satélite.	Sí	No
	Plnta_clve	SMALLINT	NOT NULL	Clave del planeta al que pertenece el satélite.	Sí	Sí
	Stlte_rdio	FLOAT	NULL	Radio.	No	No
	Stlte_prdo_sdral	FLOAT	NULL	Periodo Sideral del satélite.	No	No
Usrios	Usrio_prfil	TINYINT	NULL	Perfil del usuario que definirá sus privilegios para usar el sistema. 0 Administrador, 1 Usuario.	No	No
	Usrio_cntrsna	VARCHAR(20)	NULL	Contraseña de tipo cadena del usuario, la cual se validara cuando ingrese al sistema.	No	No
	Usrio_cmntrio	VARCHAR(20)	NULL	Comentario u observaciones acerca del usuario.	No	No

	Usrio_nmbre	VARCHAR(50)	NULL	Nombre completo del usuario.	No	No
	Usrio_lgin	VARCHAR(20)	NOT NULL	Clave tipo texto del usuario, con la que entrara al sistema.	Sí	No

Scripts SQL

Creación

```
USE boveda

CREATE TABLE Plnta (
       Plnta_clve           TINYINT NOT NULL AUTO_INCREMENT,
       Plnta_nmbre          VARCHAR(20) NOT NULL,
       Plnta_rdio_mdio      INT NULL,
       Plnta_msa            INT NULL,
       Plnta_excntrcdad    FLOAT NULL,
       Plnta_dnsdad         FLOAT NULL,
       Plnta_prdo_rtcion    FLOAT NULL,
       Plnta_dstncia_mdia_sol FLOAT NULL,
       Plnta_dscrpcion      VARCHAR(255) NULL,
       PRIMARY KEY (Plnta_clve)
);


CREATE TABLE Plnta_Crdndas (
       Plnta_clve           TINYINT NOT NULL,
       Plnta_HPM_hras       SMALLINT NULL,
       Plnta_HPM_mntos      SMALLINT NULL,
       Plnta_HPM_sgndos     SMALLINT NULL,
       Plnta_DEC_grdos      SMALLINT NULL,
       Plnta_DEC_mntos      SMALLINT NULL,
       Plnta_DEC_sgndos     SMALLINT NULL,
       Plnta_RA_hras        SMALLINT NULL,
       Plnta_RA_mntos       SMALLINT NULL,
       Plnta_RA_sgndos      SMALLINT NULL,
       Plnta_edad           FLOAT NULL,
       Plnta_vrcion_hrria_DEC FLOAT NULL,
       Plnta_vrcion_hrria_RA FLOAT NULL,
       Plnta_mes            TINYINT NOT NULL,
       Plnta_dia            TINYINT NOT NULL,
       Plnta_DG             FLOAT NULL,
       PRIMARY KEY (Plnta_mes, Plnta_dia, Plnta_clve),
       FOREIGN KEY (Plnta_clve)
                             REFERENCES Plnta
);
```

```
CREATE TABLE Plnta_Crdndas_eclptcas (
       Plnta_mes            TINYINT NOT NULL,
       Plnta_dia            TINYINT NOT NULL,
       Plnta_clve           TINYINT NOT NULL,
       Plnta_lngtud_eclptca FLOAT NULL,
       Plnta_lttud_eclptca  FLOAT NULL,
       Plnta_smdmtro_lnar   FLOAT NULL,
       Plnta_dstncia_trra   INT NULL,
       Plnta_elngcion_sntdo CHAR(1) NULL,
       Plnta_elngcion_plnta INT NULL,
       Plnta_prlje_hrzntal_lnar INT NULL,
       PRIMARY KEY (Plnta_mes, Plnta_dia, Plnta_clve),
       FOREIGN KEY (Plnta_clve)
                            REFERENCES Plnta
);

CREATE TABLE Cnstlcion (
       Cnstlcion_clve       TINYINT NOT NULL AUTO_INCREMENT,
       Cnstlcion_nmntvo     VARCHAR(25) NOT NULL,
       Cnstlcion_gntvo      VARCHAR(25) NOT NULL,
       Cnstlcion_abrvtra    CHAR(3) NOT NULL,
       Cnstlcion_dscrpcion  VARCHAR(255) NULL,
       PRIMARY KEY (Cnstlcion_clve)
);


CREATE TABLE Objto_Clste (
       Objto_Clste_clve     INT NOT NULL AUTO_INCREMENT,
       Cnstlcion_clve       TINYINT  NULL,
       Objto_Clste_Mssier   SMALLINT NULL,
       Objto_Clste_nmbre    VARCHAR(20) NULL,
       Objto_Clste_RA_hras  FLOAT NULL,
      Objto_Clste_RA_mntos FLOAT NULL,
      Objto_clste_RA_sgndos FLOAT NULL,
       Objto_Clste_DEC_grdos FLOAT NULL,
      Objto_Clste_DEC_mntos FLOAT NULL,
       Objto_Clste_DEC_sgndos FLOAT NULL,
       Objto_Clste_NBS      SMALLINT NULL,
       Objto_Clste_NGC       SMALLINT NULL,
       Objto_Clste_mgntud   TINYINT NULL,
       Objto_Clste_tpo_espctral VARCHAR(15) NULL,
      Objto_Clste_tpo_glxia CHAR(15) NULL,
       Objto_Clste_tpo_objto CHAR(1) NULL,
       Objto_Clste_mrflgia   Char(10) NULL,
       PRIMARY KEY (Objto_Clste_clve),
       FOREIGN KEY (Cnstlcion_clve)
                            REFERENCES Cnstlcion
);


CREATE TABLE Lclddes (
       Lcldad_clve          SMALLINT NOT NULL AUTO_INCREMENT,
       Lcldad_nmbre         VARCHAR(50) NULL,
       Lcldad_lttud_grdos   SMALLINT NULL,
       Lcldad_lngtud_grdos  SMALLINT NULL,
       Lcldad_lttud_mntos   SMALLINT NULL,
```

```
	Lcldad_alttud		SMALLINT NULL,
	Lcldad_lttud_sgndos	SMALLINT NULL,
	Lcldad_hrrio_vrno	 SMALLINT NULL,
	Lcldad_lngtud_mntos	SMALLINT NULL,
	Lcldad_lngtud_sgndos SMALLINT NULL,
	PRIMARY KEY (Lcldad_clve)
);


CREATE TABLE Usrios (
	Usrio_lgin		VARCHAR(20) NOT NULL,
	Usrio_nmbre		VARCHAR(50) NULL,
	Usrio_cntrsna		VARCHAR(20) NULL,
	Usrio_prfil		TINYINT NULL,
	Usrio_cmntrio		VARCHAR(20) NULL,
	PRIMARY KEY (Usrio_lgin)
);


CREATE TABLE Btcra (
	Btcra_clve		INT NOT NULL,
	Objto_Clste_clve	INT NULL,
	Lcldad_clve		SMALLINT NULL,
	Btcra_fcha		DATE NULL,
	Usrio_lgin		VARCHAR(20) NULL,
	Btcra_hra		TIME NULL,
	PRIMARY KEY (Btcra_clve),
	FOREIGN KEY (Objto_Clste_clve)
				 REFERENCES Objto_Clste,
	FOREIGN KEY (Lcldad_clve)
				 REFERENCES Lclddes,
	FOREIGN KEY (Usrio_lgin)
				 REFERENCES Usrios
);


CREATE TABLE Pscion (
	Pscion_clve		INT NOT NULL,
	Pscion_DEC_grdos	FLOAT NULL,
	Pscion_AGH_hras		FLOAT NULL,
	Pscion_DEC_mntos	FLOAT NULL,
	Pscion_DEC_sgndos	FLOAT NULL,
	Pscion_AGH_mntos	FLOAT NULL,
	Pscion_AGH_sgndos	FLOAT NULL,
	PRIMARY KEY (Pscion_clve)
);


CREATE TABLE Stlte (
	Stlte_clve		SMALLINT NOT NULL AUTO_INCREMENT,
	Plnta_clve		SMALLINT NOT NULL,
	Stlte_nmbre		VARCHAR(20) NOT NULL,
	Stlte_dstncia_plnta	FLOAT NULL,
	Stlte_prdo_sdral	FLOAT NULL,
	Stlte_excntrcdad	FLOAT NULL,
	Stlte_msa		FLOAT NULL,
```

```
       Stlte_rdio           FLOAT NULL,
       PRIMARY KEY (Stlte_clve, Plnta_clve),
       FOREIGN KEY (Plnta_clve)
                            REFERENCES Plnta
);

CREATE TABLE Prmtros (
Prmtro_clve_num         INT NOT NULL,
Prmtro_clve_prmtro      VARCHAR(20) NOT NULL,
Prmtro_dscrpcion  VARCHAR(100) NOT NULL,
Prmtro_tpo_dto          INT NOT NULL,
Prmtro_mdfcble          INT NOT NULL,
Prmtro_vlor       VARCHAR(200) NOT NULL,
PRIMARY KEY (Prmtro_clve_num)
);
```

Inserción de datos

```
use boveda

insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (1,'Andromeda','Andromedae','And','Andromeda, hija de Casiopea y
Cefeo')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (2,'Antlia','Antliae','Ant','Maquina neumatica')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (3,'Apus','Apodis','Aps','Ave del Paraiso')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (4,'Aquarius','Aquarii','Aqr','Aguador')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (5,'Aquila','Aquilae','Aql','Aguila')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (6,'Ara','Arae','Ara','Altar')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (7,'Aries','Arietis','Ari','Carnero')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (8,'Auriga','Aurigae','Aur','Cochero')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
```

```
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (9,'Bootes','Bootis','Boo','Boyero o pastor')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (10,'Caelum','Caeli','Cae','Buril')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (11,'Camelopardalis','Camaleopardalis','Cam','Jirafa')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (12,'Cancer','Cancri','Cnc','Cangrejo')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (13,'Canes Venatici','Canum Venaticorum','CVn','Lebreles o perros de
caza')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (14,'Canis Major','Canis Majoris','CMa','Can mayor')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (15,'Canis Minor','Canis Minoris','CMi','Can menor')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (16,'Capricornus','Capricorni','Cap','Cabra marina')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (17,'Carina','Carinae','Car','Carena o quilla')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (18,'Cassiopea','Cassiopeiae','Cas','Casiopea, reina')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (19,'Centaurus','Centauri','Cen','Centauro')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (20,'Cepheus','Cephei','Cep','Cefeo, rey')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (21,'Cetus','Ceti','Cet','cetaceo o ballena')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (22,'Chamaleon','Chamaleontis','Cha','Camaleon')
;
```

```
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (23,'Circinus','Circini','Cir','Compas')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (24,'Columba','Columbae','Col','Paloma')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (25,'Coma Berenices','Comae Berenices','Com','Cebellera de Berenice')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (26,'Corona Australis','Coronae Australis','CrA','Corona Austral')
;
insert into Cnstlcion (Cnstlcion_clve,Cnstlcion_nmntvo,Cnstlcion_gntvo,
Cnstlcion_abrvtra,Cnstlcion_dscrpcion)
values (27,'Corona Borealis','Coronae Borealis','CrB','Corona Boreal')
;


use boveda

insert into Lclddes(Lcldad_clve,Lcldad_nmbre,Lcldad_lttud_grdos,
Lcldad_lttud_mntos,Lcldad_lttud_sgndos,Lcldad_lngtud_grdos,
Lcldad_lngtud_mntos,Lcldad_lngtud_sgndos,Lcldad_alttud,Lcldad_uso_hrrio)
values(1,'México,D.F.',19,25,59,99,11,7,2277,NULL)
;
insert into Lclddes(Lcldad_clve,Lcldad_nmbre,Lcldad_lttud_grdos,
Lcldad_lttud_mntos,Lcldad_lttud_sgndos,Lcldad_lngtud_grdos,
Lcldad_lngtud_mntos,Lcldad_lngtud_sgndos,Lcldad_alttud,Lcldad_uso_hrrio)
values(2,'Mexicali,Baja California',31,52,8,115,11,35,0,NULL)
;
insert into Lclddes(Lcldad_clve,Lcldad_nmbre,Lcldad_lttud_grdos,
Lcldad_lttud_mntos,Lcldad_lttud_sgndos,Lcldad_lngtud_grdos,
Lcldad_lngtud_mntos,Lcldad_lngtud_sgndos,Lcldad_alttud,Lcldad_uso_hrrio)
values(3,'Xochimilco,D.F',19,15,44,99,6,7,2274,NULL)
;
insert into Lclddes(Lcldad_clve,Lcldad_nmbre,Lcldad_lttud_grdos,
Lcldad_lttud_mntos,Lcldad_lttud_sgndos,Lcldad_lngtud_grdos,
Lcldad_lngtud_mntos,Lcldad_lngtud_sgndos,Lcldad_alttud,Lcldad_uso_hrrio)
values(4,'Guadalajara,Jalisco',20,42,32,103,23,9,1567,NULL)
;
insert into Lclddes(Lcldad_clve,Lcldad_nmbre,Lcldad_lttud_grdos,
Lcldad_lttud_mntos,Lcldad_lttud_sgndos,Lcldad_lngtud_grdos,
Lcldad_lngtud_mntos,Lcldad_lngtud_sgndos,Lcldad_alttud,Lcldad_uso_hrrio)
values(5,'Monterrey,Nuevo León',25,40,11,100,18,26,538,NULL)
;



use boveda
```

```
insert into Objto_Clste(Cnstlcion_clve,Objto_Clste_Mssier,Objto_Clste_nmbre,
Objto_Clste_RA_hras,Objto_Clste_RA_mntos,Objto_clste_RA_sgndos,
Objto_Clste_DEC_grdos,Objto_Clste_DEC_mntos,Objto_Clste_DEC_sgndos,
Objto_Clste_NBS,Objto_Clste_IC,Objto_Clste_NGC,Objto_Clste_mgntud,
Objto_Clste_tpo_espctral,Objto_Clste_tpo_glxia,Objto_Clste_tpo_objto,
Objto_Clste_mrflgia)

values
(1,110,'Satelite de
M31',0,40,24,41,41,37,null,null,205,8,null,'E','G','E6'),

(1,32,'Satelite de M31',0,42,42,40,52,36,null,null,221,8,null,'E','G','E2'),

(1,31,'Galaxia de
Andromeda',0,42,42,41,16,36,null,null,224,4,null,'S','G','S'),
(18,103,null,1,33,12,60,42,8,null,null,581,4,null,null,'C','ca'),

(21,77,'Galaxia
Seyfert',2,42,42,0,1,22,null,null,1068,9,null,'S','G','Sbp'),

(8,38,null,5,28,42,35,50,15,null,null,1912,6,null,null,'C','ca'),

(12,44,'El pesebre o La
Colmena',8,40,1,19,59,1,null,null,2632,4,null,null,'C','ca'),
(25,88,'Espiral, cumulo de
Virgo',12,32,0,14,25,3,null,null,4501,10,null,'S','G','Sc'),
(1,null,'alpha',0,8,27,29,5,56,15,null,null,2,'B8IVp',null,'E',null),
(18,null,'betha',0,9,9,59,9,30,21,null,null,2,'F2III',null,'E',null),
(21,null,'betha',0,43,38,-17,58,42,188,null,null,2,'K0III',null,'E',null),
(7,null,'betha',1,54,43,20,48,55,553,null,null,3,'A5V',null,'E',null),
(7,null,'etha',2,59,18,21,20,47,887,null,null,5,'A2Vs',null,'E',null),
(11,null,'2',4,40,5,53,28,34,1466,null,null,5,'A8V',null,'E',null),
(24,null,'betha',5,51,0,-35,46,5,2040,null,null,3,'K2III',null,'E',null)
;



use boveda

CREATE TABLE Objto_Clste (
       Objto_Clste_clve     INT NOT NULL AUTO_INCREMENT,
       Cnstlcion_clve       TINYINT  NULL,
       Objto_Clste_Mssier   SMALLINT NULL,
       Objto_Clste_nmbre    VARCHAR(20) NULL,
       Objto_Clste_RA_hras  FLOAT NULL,
      Objto_Clste_RA_mntos FLOAT NULL,
      Objto_clste_RA_sgndos FLOAT NULL,
       Objto_Clste_DEC_grdos FLOAT NULL,
      Objto_Clste_DEC_mntos FLOAT NULL,
       Objto_Clste_DEC_sgndos FLOAT NULL,
       Objto_Clste_NBS      SMALLINT NULL,
       Objto_Clste_NGC      SMALLINT NULL,
       Objto_Clste_mgntud   TINYINT NULL,
       Objto_Clste_tpo_espctral VARCHAR(15) NULL,
```

```
      Objto_Clste_tpo_glxia CHAR(15) NULL,
       Objto_Clste_tpo_objto CHAR(1) NULL,
       Objto_Clste_mrflgia   Char(10) NULL,
       PRIMARY KEY (Objto_Clste_clve),
       FOREIGN KEY (Cnstlcion_clve)
                           REFERENCES Cnstlcion
);

insert into Objto_Clste(Cnstlcion_clve,Objto_Clste_Mssier,Objto_Clste_nmbre,
Objto_Clste_RA_hras,Objto_Clste_RA_mntos,Objto_clste_RA_sgndos,
Objto_Clste_DEC_grdos,Objto_Clste_DEC_mntos,Objto_Clste_DEC_sgndos,
Objto_Clste_NBS,Objto_Clste_NGC,Objto_Clste_mgntud,
Objto_Clste_tpo_espctral,Objto_Clste_tpo_glxia,Objto_Clste_tpo_objto,
Objto_Clste_mrflgia)

values
(1,110,'Satelite de M31',0,40,24,41,41,37,null,205,8,null,'E','G','E6'),

(1,32,'Satelite de M31',0,42,42,40,52,36,null,221,8,null,'E','G','E2'),

(1,31,'Galaxia de Andromeda',0,42,42,41,16,36,null,224,4,null,'S','G','S'),
(18,103,null,1,33,12,60,42,8,null,581,4,null,null,'C','ca'),

(21,77,'Galaxia Seyfert',2,42,42,0,1,22,null,1068,9,null,'S','G','Sbp'),

(8,38,null,5,28,42,35,50,15,null,1912,6,null,null,'C','ca'),

(12,44,'El pesebre o La
Colmena',8,40,1,19,59,1,null,2632,4,null,null,'C','ca'),
(25,88,'Espiral, cumulo de
Virgo',12,32,0,14,25,3,null,4501,10,null,'S','G','Sc'),
(1,null,'alpha',0,8,27,29,5,56,15,null,2,'B8IVp',null,'E',null),
(18,null,'betha',0,9,9,59,9,30,21,null,2,'F2III',null,'E',null),
(21,null,'betha',0,43,38,-17,58,42,188,null,2,'K0III',null,'E',null),
(7,null,'betha',1,54,43,20,48,55,553,null,3,'A5V',null,'E',null),
(7,null,'etha',2,59,18,21,20,47,887,null,5,'A2Vs',null,'E',null),
(11,null,'2',4,40,5,53,28,34,1466,null,5,'A8V',null,'E',null),
(24,null,'betha',5,51,0,-35,46,5,2040,null,3,'K2III',null,'E',null)
;

insert into Prmtros (
Prmtro_clve_num,Prmtro_clve_prmtro,Prmtro_dscrpcion,
Prmtro_tpo_dto,Prmtro_mdfcble,Prmtro_vlor)
values(1,"prueba","XXXXXXXXX",1,0,"prueba");




use boveda

insert into Usrios(Usrio_lgin,Usrio_nmbre,Usrio_cntrsna,
Usrio_prfil,Usrio_cmntrio)
values('admin','Administrador del sistema','admin',0,NULL)
;
insert into Usrios(Usrio_lgin,Usrio_nmbre,Usrio_cntrsna,
```

```
Usrio_prfil,Usrio_cmntrio)
values('jluis','José Luis Iturbide','jluis',1,NULL)
;
insert into Usrios(Usrio_lgin,Usrio_nmbre,Usrio_cntrsna,
Usrio_prfil,Usrio_cmntrio)
values('yovi','Yoatzin Solis','yovi',0,NULL)
;
insert into Usrios(Usrio_lgin,Usrio_nmbre,Usrio_cntrsna,
Usrio_prfil,Usrio_cmntrio)
values('greedo','Luis G Osorio','greedo',1,NULL)
;
insert into Usrios(Usrio_lgin,Usrio_nmbre,Usrio_cntrsna,
Usrio_prfil,Usrio_cmntrio)
values('invitado','Invitado del sistema','invitado',1,NULL
```