Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Departamento de Automática

TRABAJO DE DIPLOMA

Análisis de las posibilidades del enfoque orientado a objetos para el desarrollo de software educativo

Autor: Yorky Corcho Suárez

Tutor: (Dr.) Víctor Giraldo Valdés Pardo

Profesor Titular, Facultad de Ing. Eléctrica UCLV.

Santa Clara

2004

"Año del 45 aniversario del triunfo de la revolución"

Hago constar que el presente trabajo fue realizado en la Universidad Central "Marta Abreu" de las Villas (UCLV) como parte de la culminación de los estudios de la carrera de Ingeniería en Automática. Autorizo a que el mismo sea utilizado por la UCLV para los fines que se estimen convenientes, tanto de forma parcial como total y que además no pueda ser presentado en evento, ni publicado, sin la autorización expresa de la UCLV.

_____
Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado con la aprobación de la Dirección de nuestro centro y que el mismo cumple con los requisitos que debe tener un trabajo de esta naturaleza, referido a la temática señalada.

_____ _____
Firma del Tutor Firma del Jefe de Dpto.

_____
Firma del Responsable de
Información Científico-Técnica

Pensamiento

Libre, y para mi sagrado, es el derecho de pensar... La educación es fundamental para la felicidad social; es el principio en el que descansan la libertad y el engrandecimiento de los pueblos.

Benito Juárez

Dedicatoria

Dedicatoria

A las personas que más quiero en este mundo, mis abuelos, Carmela Bolaños y Miguel Suárez, por toda la dedicación, por todo el cariño y por todo el apoyo que siempre me han dado.

A la persona que le debo la vida, mi mamá, Gladys Suárez, por todo lo que ha hecho por mi y por lo mucho que me quiere.

A mis tíos, Leonel, Nelson y Luis Suárez, que siempre me han ayudado y querido como un hijo.

A mi hermanita Yoslenys Corcho, por todo el cariño que me tiene.

A quien se mantuvo en todo momento a mi lado y me ofreció todo su amor, Yessi del Río.

A mis amigos, los del aula y los de siempre, en especial a Eidel Capote y Ariel López.

A mi perrita Naomi, por darme tanta alegría.

Agradecimientos

Agradecimientos

Quiero agradecer a: Mi familia por todo el apoyo que me han dado siempre y todo su sacrificio, en especial mis abuelos, mi mamá y mis tíos.

Mi tutor, el profesor Dr. Victor Giraldo Valdés Pardo, por toda su ayuda incondicional, su disposición y su confianza en mí.

Mi amigo y compañero de estudios, Royd Reyes, por haberme ayudado siempre que lo necesite durante estos cinco años.

Todos los maestros y profesores que han tenido la bondad de transmitirme sus conocimientos a lo largo de mi vida de estudiante.

La profesora MSc. Maria del Carmen Hernández, por toda su ayuda, sus consejos y su ejemplo.

El profesor Dr. Jorge Sánchez, por su ayuda y colaboración.

La persona que siempre me animó en los momentos difíciles, me dio fuerzas para continuar y me ayudó en todo, mi novia.

Todos mis amigos y compañeros del aula, con los que compartí mi vida durante estos años.

Todas las personas e instituciones que de una forma u otra hicieron posible la realización de este trabajo.

Muchas gracias a todos.

Resumen

Resumen

En el presente trabajo se abordan las características del software educativo en sus diferentes modalidades, así como los pasos fundamentales del diseño instruccional y algunos de sus principales modelos, con el fin de proporcionar una panorámica general de las características y formas de desarrollar el software educativo.

Se analizan las etapas de análisis y diseño durante el desarrollo del software según el enfoque orientado a objetos y se describen algunas de las metodologías propuestas con esta finalidad. Se señalan las ventajas y facilidades asociadas al paradigma orientado a objetos, destacando especialmente la reusabilidad de los objetos. Se describen las principales características del lenguaje UML y las posibilidades que aporta, así como las características y ventajas de la herramienta CASE Rational Rose.

Se abordan los objetos de aprendizaje y algunos de los estándares para su elaboración, en calidad de elementos novedosos que prometen mejoras significativas en relación con el desarrollo del software educativo en la modalidad de *e-learning*..

Finalmente se integra el enfoque orientado a objetos al análisis y diseño del software educativo, tomando como ejemplo el caso del software de modelación y simulación, y particularmente los micromundos interactivos, haciendo uso de las facilidades del UML.

Tarea Técnica

Tarea Técnica

Para alcanzar los objetivos propuestos en este trabajo se ejecutan las siguientes tareas:

1. Realizar una búsqueda, análisis, sistematización e interpretación crítica de la literatura técnica relativa al área objeto de estudio.
2. Determinar las características del software educativo en sus diferentes modalidades.
3. Analizar las facilidades que ofrece la notación UML y la herramienta Rational Rose como medios para el desarrollo de productos de software educativo.
4. Describir una metodología de desarrollo y su correspondiente herramienta de apoyo para el análisis y diseño de software educativo, según el enfoque orientado a objeto.
5. Ilustrar la aplicación del enfoque dirigido por casos de uso para la elaboración de software educativo en diferentes modalidades.
6. Elaboración del informe final.

Índice

Índice

Introducción

Introducción

La utilización de las Tecnologías de la Información y la Comunicación (TIC) en calidad de recurso didáctico con el fin de apoyar procesos de enseñanza aprendizaje ha sido un área del conocimiento muy investigada, tanto teórica como experimentalmente, desde hace varias décadas. Su introducción en numerosas instituciones educativas, que ha llegado incluso hasta el hogar, muestra un espectacular aumento en los últimos años, por lo cual la demanda de software educativo con alta calidad es cada vez mayor.

Para poder disponer de software aplicable al proceso docente educativo que reúna condiciones adecuadas, se requiere utilizar metodologías que faciliten y garanticen la satisfacción de las necesidades educativas identificadas en un determinado contexto. Entre los enfoques que han sido aplicados a la producción de software educativo se destaca la metodología **orientada a objetos**, la cual permite lograr una aproximación adecuada a las características estructurales y funcionales del entorno que se modela.

El objeto de este trabajo está constituido por el **proceso de desarrollo** del software educativo, de acuerdo al paradigma de la orientación a objetos, el cual es actualmente uno de los más utilizados para la elaboración de sistemas informáticos de muy diversa índole.

Cuando se habla de "software educativo orientado a objetos" es necesario distinguir dos conceptos fundamentales, ellos son, en primer lugar, el "software educativo" -en su generalidad y como elemento de un proceso pedagógico- y por otra parte la "orientación a objetos", como paradigma para la elaboración de soluciones informáticas. Al combinarse ambos conceptos dan lugar a uno nuevo, con cualidades distintivas respecto a otros tipos de software, el cual constituye el **campo de acción** principal de nuestro trabajo.

Un propósito fundamental del presente trabajo es analizar las posibilidades del enfoque orientado a objetos con relación al desarrollo de software educativo, considerando todo el ciclo de vida del mismo. Se pretende además destacar las ventajas que presenta dicho enfoque cuando se aplica a la producción de materiales didácticos soportados en las TIC, proponer una metodología para el análisis y diseño orientado a objetos de dichos materiales, así como proponer la utilización de una herramienta CASE que sirva de apoyo a las tareas vinculadas a las diferentes fases del ciclo de vida de estos productos informáticos.

Para la realización de este trabajo nos planteamos como hipótesis que la aplicación del enfoque orientado a objetos al proceso de desarrollo del software educativo facilita las actividades vinculadas a las diferentes fases del ciclo de vida del mismo y contribuye a alcanzar una adecuada calidad, tanto en el proceso de desarrollo en sí como en el producto final obtenido.

En el presente trabajo como fundamentos metodológicos se utiliza el **enfoque de sistemas**, a fin de modelar el objeto bajo estudio, descomponiéndolo en sus elementos constituyentes, así como considerando las relaciones entre los mismos. Dichas relaciones establecen tanto la estructura del objeto como la dinámica del mismo.

Los métodos utilizados son:

El **análisis**, con vista a descomponer el objeto de estudio en sus elementos constituyentes, a fin de poder abordarlos con mayor facilidad y profundidad.

La **síntesis**, en relación con los resultados obtenidos previamente a través del análisis, con el fin de interrelacionar los diferentes componentes de la metodología propuesta así como poder aplicar integralmente la herramienta informática de apoyo a dicha metodología.

La **revisión documental,** acerca de la literatura técnica relacionada con el objeto de investigación. Como resultado de dicha revisión se incluyen en este informe más de un centenar de referencias a documentos en diferentes soportes.

Se puede afirmar que las tareas técnicas han sido realizadas de forma adecuada en su totalidad, atendiendo a las exigencias originalmente planteadas a nuestro trabajo, el cual constituye, en opinión del autor, una fuente de información importante para todos aquellas personas interesadas en investigar en esta dirección, ya que se ha logrado integrar sistemáticamente un gran volumen de información, que anteriormente se encontraba dispersa. Consideramos que este es el principal **valor científico** de nuestro trabajo.

Seguidamente daremos una breve descripción de cómo está distribuido el contenido en los diferentes capítulos.

En el Capítulo I se muestran de forma abreviada, los aspectos psicopedagógicos fundamentales que afectan al software educativo y que de hecho determinan la forma en que éste se concibe, diseña y desarrolla. Se alude a las formas en que se aprende, lo que se conoce como teorías del aprendizaje, así como a las distintas modalidades de software educativo existentes. También se presenta el proceso de desarrollo que se requiere llevar a

cabo durante el diseño instruccional de dicho software, así como las diferentes formas de estructurar este proceso. Una de las fases del diseño instruccional, el análisis de requisitos, es tratada con marcado interés, debido a la importancia que se le atribuye. Al igual que la técnica de prototipado, que forma parte de este análisis de requisitos. Finalmente se aborda el diseño de software educativo, de manera específica, debido a la relación del mismo con el tema central del trabajo.

En el Capítulo II se trata el paradigma orientado a objetos; se describen metodologías de análisis y diseño (ADOO) según distintos puntos de vista y queda claro en que consisten el análisis y el diseño orientado a objetos. Se aborda detenidamente la notación UML, así como su utilización para el análisis y diseño orientado a objetos. Debido a que los aspectos más importantes, en el paradigma OO son los objetos y sus relaciones, se hacen algunas recomendaciones para reconocer y definir objetos, así como para el diseño de jerarquías de clase. En este capítulo se muestran las ventajas del diseño, análisis e implementación según el enfoque OO, así como de las posibilidades de reutilización asociada. Por último se describe la herramienta Rational Rose y sus posibles facilidades en conexión con los aspectos antes señalados.

En el Capítulo III se pretende integrar los principales conceptos analizados en los dos capítulos anteriores, y es por ello que se trata el proceso de desarrollo del software educativo orientado a objetos, especialmente del software de modelación y simulación OO. Aquí se abordan los aspectos relativos a estructuras de clases, ampliabilidad, reutilización e integración, la cual se enfatiza particularmente. Se describen los conceptos fundamentales sobre objetos de aprendizaje, como un ejemplo del estado del arte actual en cuanto a las técnicas de elaboración de materiales didácticos soportados en TIC, especialmente para entornos de educación a distancia (e-learning). Se resaltan las ventajas asociadas a estos aspectos y otros, como la modularidad. En este contexto resulta imprescindible hablar de los estándares existentes actualmente y que rigen el proceso de creación y aplicación de los mencionados objetos de aprendizaje. Como punto culminante en este trabajo se presenta el análisis y diseño OO para micromudos interactivos usando notación (UML), atendiendo a un enfoque constructivista.

Capítulo 1

1. Fundamentos pedagógicos y metodológicos para el desarrollo del software educativo

Objetivo específico.

Exponer las bases teórico-conceptuales para la utilización de las Tecnologías de la Información y la Comunicación (TIC) en calidad de recursos didácticos y los conceptos claves asociados al diseño instruccional y del software educativo.

1.1 Bases psicopedagógicas del desarrollo del software educativo.

Aunque queda fuera del alcance de este trabajo el tratamiento de las diferentes aproximaciones pedagógicas y teorías educativas existentes, resulta ineludible exponer, aunque sea de forma abreviada, los aspectos psicopedagógicos fundamentales que afectan al software educativo y que de hecho determinan la forma en que éste se concibe, diseña y desarrolla.

El software educativo, como cualquier otro sistema instruccional, está destinado a apoyar el proceso de enseñanza aprendizaje de un determinado contenido. En el marco del debate contemporáneo entre *objetivismo* y *constructivismo*, algunos autores conciben la instrucción como una senda por la cual el conocimiento es "transmitido" de forma directa al estudiante [38] [99] , mientras que para otros el aprendizaje consiste en la "construcción" del conocimiento por parte del estudiante, o el "aprendizaje por descubrimiento" [26] [75] .

No obstante, consideramos que tanto el aprendizaje como la enseñanza van tomados de la mano y el término enseñanza debe ser entendido como algo más general. De hecho, nos adscribimos a la noción de la ***enseñanza*** considerada como "la creación y utilización de entornos en los cuales el aprendizaje es facilitado o promovido" [1] [54] . En cualquier caso, para nuestros propósitos, no vamos a establecer una ruptura entre ambos términos, sino más bien a considerarlos como una dualidad, ya que en cualquier caso el objetivo final del

software educativo es lograr que los estudiantes hagan suyos determinados conocimientos, habilidades y actitudes.

1.1.1 Teorías del aprendizaje

Diferentes teorías del aprendizaje se han desarrollado a lo largo del tiempo, como fruto de numerosos estudios. Estas teorías se orientan a describir las maneras como se aprenden nuevas ideas y conceptos (a menudo, explican la relación entre la información que se conoce y la nueva información que se debe aprender). Las teorías del aprendizaje [95] que predominan hoy en día son: conductismo, cognitivismo y constructivismo. En rasgos generales sus características son:

El ***Conductismo*** [36] [89] [101] : considera que el aprendizaje da por resultado cambios observables en la conducta del sujeto. Se enfoca hacia la repetición de patrones de conducta, hasta que estos se realicen de manera automática. Algunas personas claves en el desarrollo de esta teoría han sido Pavlov, Watson, Thorndike y Skinner.

El ***Cognitivismo*** [5] [45] : considera que el aprendizaje ocurre cuando los aprendices son capaces de incorporar nuevos conceptos e ideas a su estructura cognitiva, al reconocer una relación entre algo que ya conocen y aquello que están aprendiendo. El foco de los cognitivistas se centra en las entradas del proceso de aprendizaje y en los procesos del pensamiento que subyacen a la conducta. Los cambios en la conducta les sirven como indicadores para entender lo que está pasando en la mente del que aprende.

Tanto el conductismo como el cognitivismo se basan, para determinar si ocurre el aprendizaje, en datos “objetivos” acerca de los cambios en la conducta del que aprende. Es por ello que ambas teorías se incluyen en la categoría más abarcadora de ***objetivismo***. Una diferencia esencial entre ambas radica en que para los conductistas la mente humana es una "caja negra", mientras que los cognitivistas intentan verla más como una "caja transparente".

El ***Constructivismo*** [54] [64] : se sustenta en la premisa de que cada persona construye su propia perspectiva del mundo que lo rodea, a través de sus propias experiencias y esquemas mentales desarrollados. El constructivismo se enfoca hacia la preparación del que aprende para resolver problemas en condiciones ambiguas. Pioneros de esta aproximación fueron Barlett [45] y Piaget [76] .

El ***Enfoque Histórico-cultural*** ha evolucionado sobre la base de la obra de L. S. Vigostky y su teoría del desarrollo histórico cultural de las funciones psicológicas. En ella es necesario destacar el desarrollo alcanzado a partir de la experiencia individual y la "zona de desarrollo potencial" engendrada. El individuo concurre a un proceso educativo de carácter mediatizado en el cual, más allá de responder a los estímulos y adaptarse pasivamente a las condiciones del medio, actúa sobre dichos estímulos y modifica activamente su entorno.

Algunas ideas fundamentales para la práctica pedagógica que se han desarrollado tomando como punto de partida el Enfoque Histórico-cultural, son las siguientes:

- El análisis del conocimiento como un proceso dinámico e interactivo, que además de involucrar a aprendices y profesores, también afecta la relación padre-aprendiz.
- El carácter interactivo del proceso propicia que lo interpsicológico se convierta en intrapsicológico, momento en el que aparece el fenómeno metacognitivo.
- La cultura y el contexto histórico concreto constituyen determinantes del aprendizaje e influyen en su carácter positivo o negativo.
- El lenguaje como mediador entre lo intra y lo interpsíquico, conjugado adecuadamente con herramientas didácticas, condiciona y facilita el proceso de aprendizaje.
- El proceso educativo dirige y a su vez propicia un mayor desarrollo de los procesos psicológicos y, por tanto, de la personalidad de los implicados en dicho proceso.

Actualmente algunas de estas teorías están más ampliamente aceptadas en ciertos círculos que otras. Ninguna de ellas se puede considerar como universalmente válida frente a las demás, ya que existen aspectos ambientales, socioculturales y técnicos que pueden influir a la hora de tomar como base alguna de ellas para el diseño de sistemas educacionales. Y esta elección tendrá, lógicamente, consecuencias en la concepción y el proceso de desarrollo de estos sistemas.

1.1.2 Clasificación pedagógica del software educativo.

En dependencia de la estructura y realización, que obedece al objetivo que se persiga en la elaboración del software ya sea educativo o no, y las teorías del aprendizaje que les resultan afines, se pueden distinguir diversas modalidades de software educativo, a saber:

- Tutoriales
- Hipermedia
- Ejercitadores
- Simulaciones
- Juegos
- Herramientas
- Evaluaciones
- Aprendizaje soportado en la Web

❖ **Tutoriales**

Una gran parte del software educativo existente hoy en día corresponde a esta categoría. Intentan reproducir una forma de enseñanza conocida como "diálogo socrático", pues le presentan información al aprendiz y lo van guiando en las etapas iniciales de adquisición del conocimiento, a través del planteamiento de preguntas y el análisis de sus respuestas, con el fin de provocar la reflexión del estudiante y promover el aprendizaje de los conceptos objeto de estudio. En el diseño de estos programas se confiere una atención especial a los mecanismos de diagnóstico y corrección de errores, a fin de impedir la acumulación de los mismos [108] .

La actividad del aprendiz es esencialmente dirigida por el programa. La base psicopedagógica fundamental de este tipo de software educativo es la teoría del conductismo [99] , aunque también se aprecia una tendencia a la incorporación de ideas provenientes de la teoría del cognitivismo [13] . En este sentido cabe señalar aquellos materiales de software educativo que utilizan técnicas de Inteligencia Artificial, los cuales frecuentemente adoptan la forma de tutoriales [31] en los que se intenta modelar alguna de las características cognitivas del aprendiz y permitirle en cierta medida desplegar su iniciativa. Existen marcadas diferencias en la forma en que se concibe el diseño de un tutorial convencional y de uno inteligente [33] .

❖ **Hipermedia**

Esta categoría de software educativo presenta un gran auge en los últimos años. Un *hipertexto* se puede definir como un grafo entre cuyos nodos se establecen vínculos, que

permiten la organización “no lineal” de la información. En una estructura hipermedia [69] los nodos pueden contener información expresada en diferentes formatos (texto, gráfico, sonido, animación, video) y además pueden combinarse nodos pasivos y activos. Estos últimos pueden incluir módulos ejecutables que se activan en tiempo real.

En su utilización como recurso educacional, la hipermedia resulta potencialmente menos estructurada que el tutorial, por el hecho de que permite a los usuarios establecer trayectorias propias para el acceso a la información, las cuales probablemente diferirán entre sí por diversas razones. Un problema específico que puede surgir es que el usuario “se pierda” en el hiperespacio, sin saber claramente en qué dirección debe proseguir su trayectoria. Este problema puede ser aliviado mediante la inclusión de recursos de ayuda a la *navegación,* que suministren orientaciones adecuadas [72] .

Los fundamentos psicopedagógicos de la hipermedia educativa pueden relacionarse tanto con la teoría del conductismo como con la del constructivismo, en dependencia del grado de iniciativa que esté a disposición del estudiante, ya sea para la utilización o para la elaboración [55] de software educativo hipermedia.

- **Ejercitadores**

Estos programas permiten la aplicación práctica de los conocimientos. El propósito fundamental de los ejercitadores no es enseñar nuevos contenidos, sino repetir el material a ser aprendido hasta que el usuario demuestre que lo domina adecuadamente. Normalmente la utilización de un ejercitador debe ir precedida de otras metodologías de instrucción, a través de las cuales el aprendiz pueda tener acceso a la información didáctica pertinente. Frecuentemente se combinan las modalidades de ejercitación y juego, con el fin de promover una mayor motivación en el estudiante. Los ejercitadores, al igual que los tutoriales convencionales, incorporan como base teórica los principios del conductismo.

- **Simulaciones**

Esta es una categoría compleja de software educativo, que instrumenta un modelo de algún fenómeno o actividad acerca del cual se pretende que los estudiantes aprendan, a través de la interacción con el programa. En principio una simulación puede utilizarse para presentar

información y guiar al aprendiz, para guiarlo y que practique, para hacer las tres cosas o también para evaluar su desempeño. Las simulaciones pueden ser aplicadas siguiendo un enfoque objetivista, o un enfoque constructivista. En muchos casos el software de simulación permite a los estudiantes operar libremente, dentro de los límites de un determinado entorno o *micromundo* [17] . Las simulaciones pueden ser también combinadas con la modalidad de juego didáctico [32] con el fin de propiciar el aprendizaje por descubrimiento [26] .

- **Juegos**

Esta modalidad de software educativo presenta dos conjuntos principales de usuarios: comunmente son utilizados con niños pequeños, en escuelas de nivel elemental y medio y también frecuentemente se aplican con estudiantes de nivel superior o de formación profesional, en cursos sobre administración y negocios. Los juegos didácticos, como ya se ha mencionado previamente, pueden combinarse con ejercitaciones o con simulaciones. Los juegos normalmente permiten practicar con los conocimientos de manera no repetitiva, en lo cual se diferencian de las ejercitaciones; pueden servir como entorno de aprendizaje por descubrimiento, o también pueden utilizarse para integrar aprendizajes sobre un conjunto de materias, como se hace a menudo con juegos del género de aventuras [8] . Con menor frecuencia los juegos se utilizan para suministrarle orientación o evaluar el desempeño del aprendiz, cuando se combinan con la modalidad de simulación.

- **Herramientas.**

Consisten en programas de computador que los estudiantes pueden utilizar, conjuntamente con otros medios o actividades, para alcanzar alguna meta educacional. Son esencialmente abiertos y flexibles.

Por ejemplo, una herramienta de diseño gráfico puede apoyar el aprendizaje del dibujo artístico, o también la elaboración de gráficos de funciones matemáticas. Diferentes asistentes de cálculo pueden ser aplicados en la enseñanza de cursos de administración y negocios. A través de un micromundo se pueden desarrollar las capacidades de formulación de conjeturas y pruebas en geometría euclideana [58] .

Las herramientas pueden ser incorporadas en cualquiera de las fases del modelo de instrucción y su utilización puede venir dictada tanto desde una perspectiva tanto objetivista

como constructivista. Algunas actividades didácticas que típicamente se benefician con la aplicación de un software tipo herramienta son: escribir, calcular, dibujar, planificar, componer y comunicar.

❖ **Evaluaciones**

Las pruebas o exámenes son las que detrminan el grado de aconocimientos adquiridos en la etapa de aprendizaje. A partir de las muchas situaciones posibles a considerar en relación con las necesidades y modalidades de evaluación del aprendizaje, el software puede ser empleado para facilitar la misma en dos sentidos principales: 1) la *elaboración* de exámenes (los cuales eventualmente pudieran ser aplicados de manera convencional) y 2) la *administración* de pruebas y exámenes. Por una parte el software permite generar, imprimir y calificar pruebas que los examinandos pueden leer y responder sobre el papel. Por otra parte, gracias a la creciente expansión de las redes de computadores, la aplicación de pruebas directamente a los estudiantes, ubicados frente a terminales interactivas, se convierte cada día en un hecho más factible y generalizado [29] .

❖ **Aprendizaje soportado en la Web**

World Wide Web (WWW) es el nombre de un servicio hipermedia distribuido que tuvo su origen en el Consejo Europeo para Investigaciones Nucleares [7] . El mismo está disponible como parte de Internet y permite navegar con facilidad a través de un vasto - y continuamente creciente - volumen de información La Web por tanto es esencialmente un medio para distribución y acceso a la información, que en principio permite soportar cualquiera de las modalidades de software educativo antes mencionadas. Hoy en día se utiliza mayormente en combinación con la modalidad de hipermedia educativa.

Es importante recalcar que la clasificación anteriormente presentada se basa en criterios pedagógicos y no tanto tecnológicos, aunque la separación entre categorías no siempre se puede establecer de forma totalmente nítida, e incluso a veces se dificulta por la coincidencia de términos utilizados. Por ejemplo, la hipermedia, entendida como *tecnología* (y no como enfoque pedagógico), puede servir tanto para implementar un tutorial como un entorno de aprendizaje menos estructurado, o incluso evaluaciones. Así mismo, la tecnología para

implementar simulaciones y juegos es a menudo prácticamente la misma, ya que, desde este punto de vista, muchos juegos se pueden ver como simulaciones. En la mayoría de las ocasiones estos software están estrechamente relacionados y dependen unos de otros, aunque esto no tiene que ser necesariamente así.

En los apartados precedentes se consideran las teorías psicopedagógicas más importantes que influyen sobre la concepción y función del software como sistema educacional. A continuación enfocamos la atención sobre el proceso de desarrollo del software educativo.

La *ingeniería del courseware*, como disciplina centrada en el desarrollo sistemático del software educativo [46] , establece las tareas genéricas y comunes a cualquier proceso de desarrollo de software con algunas otras particularidades. Este se inscribe a su vez dentro del proceso más amplio y general del Diseño Instruccional, que se puede definir como la producción de cualquier tipo de material o actividad orientada a promover la enseñanza de un contenido [14] [36] .

1.2 El proceso de diseño instruccional

El *diseño instruccional* es un enfoque sistemático para lograr unos resultados específicos de aprendizaje por medio del diseño de la metodología de enseñanza y los materiales más adecuados de forma que promuevan los cambios deseados en conocimientos, habilidades y actitudes con respecto a un determinado contenido, en una determinada población [81] . Debe tenerse en cuenta que el diseño instruccional es independiente del empleo del computador para la enseñanza, siendo éste, en cualquier caso, un medio más, utilizable como soporte de material educativo, al igual que podrían serlo el papel, equipos de audio o vídeo, maquetas, fotografías, etc...

Evidentemente, el diseño instruccional se sustenta, en último término, en una teoría del aprendizaje, que orienta las decisiones que se adoptan durante el proceso de desarrollo de los materiales y técnicas de enseñanza. Pero independientemente de ésto, como cualquier proceso de desarrollo, consta de una serie de fases en las que se realizan ciertas tareas bastante bien diferenciadas, hasta llegar al producto final.

El proceso de desarrollo que se lleva a cabo durante el diseño instruccional de una actividad formativa, se describe en base a las siguientes fases [36] [38] [59]

I. Análisis de requisitos: el objetivo de esta fase es diagnosticar las necesidades de aprendizaje y producir una especificación de requisitos más o menos precisa. En esta fase, se debe:

- Identificar los tipos de resultados de aprendizaje que se desea lograr; traducir objetivos más o menos abstractos a elementos específicos que puedan ser medidos y establecer una jerarquía de resultados adecuada.
- Identificar qué procesos o condiciones internas deben estar presentes en el aprendiz para poder alcanzar estos resultados.
- Identificar qué condiciones externas o instrucciones deben de proporcionarse para que se produzcan las condiciones internas.
- Determinar el costo del proyecto y realizar una adecuada planificación del mismo, para ajustarse a las restricciones de tiempo y recursos.

II. Diseño y selección de medios de comunicación: las tareas que se realizan típicamente en esta fase se centran en planear una estrategia de desarrollo de la instrucción, a partir de los resultados obtenidos en la fase anterior. Ellas son:

- Esclarecer el contexto del aprendizaje y las características relevantes de los receptores potenciales de la acción formativa.
- Seleccionar la combinación de medios de comunicación más adecuada, que será utilizada para apoyar las diferentes etapas de instrucción: a través de libros, enseñanza asistida por computador, videos u otros.
- Planificar las etapas instruccionales, a fin de apoyar las actividades del aprendizaje. Para cada uno de los resultados incluidos en la jerarquía de aprendizaje, se diseñan las etapas instruccionales según el tipo de resultado de aprendizaje requerido, en el orden de los prerrequisitos de la jerarquía del aprendizaje y considerando los medios y materiales apropiados y la eventual intervención del instructor.

III. Desarrollo e implementación: en esta fase se generan los planes de lecciones o unidades didácticas y el correspondiente material asociado a cada una. Esto puede suponer escribir textos, grabar vídeos, desarrollar software... Todo ello debe ser luego integrado adecuadamente en el sistema educacional final.

IV. Revisión: cuando el desarrollo del material didáctico está avanzado y el diseño instruccional obtenido aparentemente está listo para su aplicación, se realiza una evaluación formativa. Si el balance de ésta es suficientemente positivo, se puede aplicar una experiencia piloto, con vista a garantizar la comprensión del material por parte del estudiante y la adquisición de conocimientos planeada. Una vez que la instrucción ha sido llevada a la práctica, una prueba de campo permite a los desarrolladores juzgar acerca de su adecuación y efectividad.

V. Evaluación: Realmente la evaluación no es una fase como tal. Debe ser una tarea sistemática, que se realiza durante todo el proceso de diseño instruccional. Ésta puede ser formativa o sumativa. La primera se efectúa en cada fase y entre fases del proceso, para tratar de controlar la calidad y mejorar la instrucción antes de llegar a la versión final. La segunda se lleva a cabo con el producto final del proceso, para medir su efectividad didáctica y tomar decisiones sobre la acción instruccional, definir nuevos requisitos, corregir deficiencias.

Resumiendo, el Diseño Instruccional comienza con un análisis del aprendizaje que se debe alcanzar, seguido del diseño y selección o desarrollo de medios y materiales adecuados y finalmente todo ello se traduce en un plan de etapas instruccionales que estimulen y apoyen los procesos cognitivos del aprendiz. Estos materiales serán finalmente verificados, utilizados y evaluados prácticamente para comprobar su efectividad.

1.2.1 Modelos de diseño instruccional

En el apartado anterior se detallan las fases básicas de que consta todo proceso de diseño instruccional. Pero a la hora de llevar a cabo un proceso concreto, estas fases pueden estructurarse de variadas formas e incluir diferentes tareas. Esto da lugar a la aparición de una diversidad de modelos de diseño instruccional.

Los modelos instruccionales son guías o conjuntos de estrategias en base a los cuales se realiza la enseñanza. Los diferentes modelos están basados en las teorías del aprendizaje descritas en el apartado 0 y definen una determinada estructura y significado para un proceso de diseño, desde una perspectiva particular. Los modelos más relevantes con respecto al soporte al diseño instruccional como proceso de desarrollo, son los arquitectónicos, que se distinguen por presentar el proceso desde la perspectiva de fases relacionadas entre sí.

Los modelos de diseño instruccional son importantes porque proporcionan un marco de actuación para la producción sistemática de materiales y recursos de instrucción, e incorporan las fases fundamentales del proceso de diseño instruccional. Pueden ser utilizados en diferentes contextos y combinados para adaptarse a situaciones y requisitos particulares.

Están disponibles muchas propuestas de modelos de diseño instruccional. Por ejemplo, Andrews y Goodson [3] revisaron 40 modelos publicados y Branch [16] recoge la práctica actual en las escuelas secundarias en los Estados Unidos. Pero todos estos modelos pueden catalogarse con referencia a un conjunto reducido de estructuras arquitectónicas o modelos más generales. Estos modelos no son exclusivos del diseño instruccional, sino que aparecen en muchas otras ingenierías (entre ellas la ingeniería del software), ya que describen estructuras generales de procesos de desarrollo que pueden utilizarse en muy diversos ámbitos.

- **Modelo de cascada:**

El modelo de proceso de desarrollo quizá más clásico es el “modelo de cascada”, debido a que propone una sucesión o cascada de etapas, cada una de las cuales no se inicia hasta que las anteriores hayan terminado. En la práctica, las etapas se solapan y se intercambia información entre unas y otras.

Como un ejemplo de modelo de diseño instruccional cuya estructura se ajusta al modelo de cascada, se puede mencionar el debido a Dick y Carey [30] que es uno de los más conocidos y representativos. Describe todas las fases de un proceso iterativo que comienza con la identificación de las metas instruccionales y finaliza con una evaluación sumativa. Este modelo es aplicable a un amplio espectro de áreas de contenido y categorías de aprendices. Otro ejemplo en la categoría de cascada es la Metodología ISE propuesta por Galvis [40] , la cual abarca mecanismos de análisis, diseño educativo y comunicacional, prueba piloto y de campo, los que se fundamentan en principios educativos, comunicacionales y de tecnología educativa de validez comprobada (Ver Anexo 1).

- **Modelo de prototipado:**

El enfoque de desarrollo de un sistema mediante la construcción de prototipos evolutivos parte de una especificación abstracta del sistema deseado, a partir de la cual se crea un

prototipo que implementa los requisitos más importantes del cliente. Dicho prototipo se modifica según se van descubriendo nuevos requisitos, debido a los ensayos que se efectúan con el prototipo tanto por el cliente como por el equipo de desarrolladores. Eventualmente, el prototipo evoluciona hasta convertirse en el sistema deseado.

Un modelo de diseño instruccional basado en el prototipado rápido es el debido a Tripp y Bichelmeyer [30] . Este modelo está pensado más bien para la creación de lecciones independientes, no de cursos completos. Las etapas incluyen un análisis de necesidades, la elaboración de un prototipo, la utilización de dicho prototipo para efectuar investigación y eventualmente llegar a la instalación de la versión final del sistema. Para su instrumentación, este modelo requiere que sea aplicado por diseñadores instruccionales expertos, que utilicen heurísticas así como su intuición y experiencia acumulada para conducir a buen término el proceso de diseño.

- **Modelo espiral:**

Este modelo fue originalmente propuesto por Boehm [11] . Pretende reflejar la dialéctica asociada al proceso de desarrollo a partir de un ciclo interno donde se establece la factibilidad del sistema, seguido de un ciclo de definición de requisitos, el cual a su vez continúa con el diseño del sistema, su verificación, la planeación de una nueva fase y así sucesivamente. Cada ciclo consta de cuatro fases que se van repitiendo, pero cada vez con un bagaje de experiencia acumulada mayor. Un aspecto distintivo de este modelo es la consideración explícita de los riesgos asociados al proyecto, su identificación y resolución.

El modelo de diseño instruccional propuesto por Jerrold Kemp [30] se asemeja en su estructura al modelo espiral y se fundamenta en un enfoque holístico. Toma en consideración prácticamente todos los factores asociados al entorno de aprendizaje, pues incluye análisis del contenido, de las características de los aprendices, los objetivos de aprendizaje, las actividades de enseñanza, los recursos didácticos (libros, instrumentos, software, etc), los servicios de apoyo y la evaluación pedagógica. El proceso es iterativo y el diseño se somete a una constante revisión y perfeccionamiento.

.

1.2.2 Análisis de requisitos del software

En la ingeniería del software se presta mucha atención a la especificación de los requisitos, puesto que es una tarea fundamental para el éxito del proceso [21] [102] . Existen diversos enfoques o teorías subyacentes para la obtención de requisitos del software. Se pueden citar los siguientes:

- Obtención orientada a ***puntos de vista*****.** Se basa en recoger y estructurar los requisitos derivados de los diferentes puntos de vista que diferentes usuarios pueden tener acerca de un sistema. Un ejemplo es el método VORD [56]
- Obtención basada en ***escenarios***. Un escenario es un ejemplo de una sesión de interacción con el sistema, que se plantea a un usuario y que permite capturar requisitos en base a lo que el usuario describe que debería ocurrir en tal escenario [18] . En los últimos años este enfoque ha cobrado gran relevancia, gracias a la difusión de la técnica de *casos de uso* [87] , incluida en el Proceso Unificado de Desarrollo [51] .
- Obtención a partir de la ***etnografía*****.** Se apoya en la observación del contexto real donde funcionará el sistema para entender los requisitos sociales y organizacionales [2] . La etnografía es útil para capturar requisitos que se derivan de la forma real en que las personas trabajan, más que de los procedimientos formales definidos de cómo deberían hacerlo y que se obtienen gracias a la cooperación y el conocimiento de las actividades de la gente.

En cualquier caso, a nivel del análisis de requisitos del software educativo, ninguno de estos tres enfoques han sido particularmente considerados en la literatura sobre el tema.

Esto puede explicarse por el hecho de que, a menudo en el desarrollo de software educativo no existe una participación directa del usuario final (el estudiante), o ésta es muy limitada. Por tanto quienes ejercen de fuentes de información e interlocutores del analista de requisitos son principalmente los miembros del equipo encargado del diseño instruccional de courseware, así como la documentación y resultados provenientes del proceso de diseño instruccional.

Por otra parte, conviene señalar que en ciertos casos la funcionalidad del software educativo es sensiblemente menos compleja que la de otros tipos de software, lo cual simplifica su análisis. Por ejemplo, los tutoriales o las aplicaciones hipermedia suelen centrarse más en

presentar información que en proporcionar funciones o servicios complejos. Una excepción puede ser el software con alto enfoque constructivista, o el más orientado a enseñanza superior y de contenido más científico-tecnológico. Pero incluso en estos casos, la funcionalidad a implementar suele quedar bien definida con anterioridad al análisis de requisitos del software.

Todo ello descarta en muchos casos la utilidad de un enfoque de análisis basado en puntos de vista. También hace que normalmente no existan excesivos problemas en la determinación de la funcionalidad esperada del software en diferentes escenarios, por lo que aunque un enfoque basado por ejemplo, en casos de uso [87] , puede ser útil para documentar los requisitos, a menudo no aportará más información que la ya suministrada por el diseño instruccional.

En consecuencia, el enfoque clásicamente más extendido para completar el análisis de requisitos del software a construir suele ser el de basarse en la construcción de prototipos [1] [14] , que se expone en el siguiente apartado.

En cualquier caso, normalmente el resultado del análisis de requisitos se recoge en un documento de Especificación de Requisitos del Software.

1.2.2.1 Prototipado

Dada la importancia de un correcto análisis de requisitos para el desarrollo de software, esta técnica ha mostrado ser una valiosa alternativa para reducir los riesgos asociados a errores en las especificaciones, e incluso para reducir los costos totales de desarrollo [10] . Por lo general, el desarrollo de prototipos incrementa el costo en las etapas iniciales del proceso, pero éste se reduce en las etapas subsiguientes.

Hoy en día, debido a la naturaleza dinámica de las interfaces de usuario que se requieren comúnmente en las aplicaciones educativas, la construcción de prototipos es una parte esencial del proceso de desarrollo de software educativo [94]

Es importante distinguir entre la construcción de prototipos exploratorios y desechables, pues se trata de paradigmas evolutivos diferentes, e incluso contrapuestos en ciertos aspectos:

Bajo un paradigma **exploratorio**, se pretende construir un prototipo que vaya evolucionando hasta convertirse en el sistema final que se entregue. Por ello, se comienza incluyendo en el prototipo los requisitos más claros y evidentes, para luego ir refinándolo, conjuntamente con

el cliente, al tomar en cuenta otros requisitos que inicialmente no estaban claros o no se habían identificado. En consecuencia, la construcción del prototipo debe realizarse con atención a normas de calidad y sin descuidar ningún aspecto de su diseño e implementación, ya que la mayoría de los elementos del prototipo inicial formarán parte del sistema final.

Bajo un paradigma de prototipado **desechable**, la función primordial del prototipo es ayudar a la especificación y validación de requisitos. Por ello, la construcción del prototipo se centra en tratar de implementar los requisitos menos entendidos, más oscuros, para poder contrastarlos con el cliente y así lograr una especificación más fiable. No se pretende en ningún caso que el prototipo sea la base del sistema final, e incluso se considera generalmente que eso es algo a evitar a toda costa. Suele ser un factor clave el poder construir un prototipo según esta modalidad de forma rápida y a bajo costo. Y dado que será desechado, sin necesidad de prestar atención a los aspectos que no sean relevantes para su misión primordial. Por ejemplo, la fiabilidad, robustez, rendimiento y calidad del diseño, no serán normalmente tomados en cuenta (a menos que se trate precisamente de prototipar alguno de dichos aspectos). Por todo ello, normalmente es fundamental evitar que se tome el prototipo como base del sistema final.

En la actualidad ha cobrado un mayor auge el prototipo evolutivo, a ello han contribuido especialmente la gran cantidad de desarrollos de sistemas de tamaño pequeño y medio, altamente interactivos, basados en computador personal, que en su gran mayoría siguen una aproximación evolutiva. También se sigue esta aproximación, generalmente, en el desarrollo de sitios Web. Además, la aparición y rápida difusión de herramientas RAD (como Visual Basic [109] , Delphi [27] , entornos Java [47] , PowerBuilder, etc...) ha proporcionado un soporte instrumental muy importante a este paradigma [67] [104] .

Esta consideración es importante, ya que uno de los enfoques para llevar a cabo un desarrollo evolutivo en la construcción de software es el basado en componentes reutilizables.

Es importante tener en cuenta que en un modelo de proceso evolutivo no existe una clara separación entre análisis, diseño e implementación, sino que estas fases se llevan a cabo de forma iterativa para generar pequeños incrementos de funcionalidad en el software. Por tanto, no se dispone de un documento de análisis o especificación funcional completa.

1.3 Diseño de software educativo

Un diseño de software es una descripción de la estructura del software a implementar, los datos que son parte del sistema, las interfaces entre los componentes del sistema, y a veces los algoritmos utilizados [103] . Dada la amplia literatura existente sobre este tema no se describe aquí el proceso de obtención de un diseño detallado. Basta mencionar que generalmente implica desarrollar varios modelos del sistema con diferentes niveles de abstracción. Las actividades del diseño se entrelazan, existe mucha retroalimentación entre ellas, ya que al descomponer un diseño se descubren los errores y omisiones introducidas en etapas previas.

Los métodos de diseño más usuales se apoyan en algunos de los siguientes modelos de un sistema:

- Modelo de flujo de datos, en el que el sistema se representa sobre la base de las transformaciones que van sufriendo los datos cuando se procesan.
- Modelo entidad-relación, que describe las entidades fundamentales en el diseño y las relaciones entre ellas.
- Modelo estructural, que describe los componentes del sistema y sus interacciones.
- Métodos orientados a objetos, que incluyen un modelo de herencia del sistema, modelos de relaciones estáticas y dinámicas entre objetos y modelo de interacción de objetos cuando el sistema se ejecuta.

En el software educativo, varios de estos modelos pueden ser utilizados provechosamente, en dependencia de la orientación del software. Por ejemplo, los modelos orientados a objetos y los de flujo de datos soportan especialmente bien el diseño de simulaciones, entornos de modelado, micromundos, etc... Los modelos entidad-relación, en cambio, se adecuan más al diseño de aplicaciones que utilicen bases de datos, o presenten información estructurada, como en el caso de muchos sitios Web.

El resultado final del proceso de diseño son unas especificaciones precisas de los algoritmos y estructuras de datos a implementar. El documento final de diseño puede recoger especificaciones de diferentes tipos, como la de la arquitectura del sistema, su funcionalidad, la interfaz y restricciones de cada subsistema, la especificación de la estructura de datos y de los algoritmos.

Conclusiones parciales del Capítulo.

El proceso de desarrollo del software educativo depende en gran medida de las consideraciones pedagógicas y metodológicas asociadas al diseño instruccional.

Existen diferentes modelos para el proceso de desarrollo del software; entre ellos cabe mencionar el de cascada, el de prototipado y el espiral, todos los cuales pueden ser aplicados al caso del software educativo.

El análisis de requisitos es una fase decisiva dentro del proceso de desarrollo del software, pues a partir de la misma se define la funcionalidad del sistema. El prototipado puede apoyar en gran medida el levantamiento de los requisitos de un software educativo.

Entre los métodos de diseño de software educativo más usualmente utilizados se encuentran el orientado al flujo de datos, el de entidad-relación, el enfoque estructurado y el orientado a objetos. Este último resulta particularmente interesante con relación al presente trabajo.

Capítulo 2

2. Principios, métodos y herramientas para el desarrollo del software orientado a objetos

Objetivos específicos.

Analizar las características del paradigma orientado a objetos, así como revisar diferentes metodologías para el análisis y diseño de software de acuerdo a este enfoque. Describir las características de la notación UML y la herramienta Rational Rose, considerados como recursos para el desarrollo del software orientado a objetos.

2.1 Paradigma Orientado a Objeto.

En la actualidad se manifiesta un cambio en la naturaleza de la programación. Aparece una nueva cultura en la cual el énfasis no se pone sobre los proyectos sino en las componentes. La necesidad de este cambio fue expresada en la década del 60 por Doug McIlroy [61] cuando expresó su tesis de que la industria de software esta débilmente fundamentada, en parte por la ausencia de una subindustria de componentes de software, la cual podría ser muy exitosa.

El paradigma de la programación orientada a objetos (POO) es una respuesta a esta necesidad. Ha alcanzado una amplia aceptación como una tecnología básica dentro de la industria de software. La nueva técnica cambia la naturaleza del diseño de software, que de una tarea artesanal pasa a ser un proceso industrial. La POO facilita desarrollar el producto de software de una manera similar a los productos de hardware; la programación cambia de una tarea en la cual lo fundamental es escribir instrucciones en un lenguaje, a una tarea donde el centro es interconectar componentes de software reusables.

El término “reusable” denota que los diseños pueden ser construidos a partir de tipos predefinidos, componiendo nuevos objetos a partir de otros y redefiniendo tipos mediante la herencia. La reutilización a gran escala permite que una clase sea usada no sólo por sus creadores originales, sino por otros grupos de programadores.

La cuestión principal radica en que este paradigma ofrece el objeto como un elemento que es

común a todas las fases del ciclo de vida del software (análisis, diseño e implementación). Esta uniformidad permite una transición suave de una fase a otra. Otro aspecto que caracteriza al enfoque OO para el desarrollo del software es que la reusabilidad de código existente pasa a un primer plano. Sin embargo, la utilización de un lenguaje orientado a objetos (LOO) para implementar un sistema no implica necesariamente que el programador tenga que aplicar un enfoque de desarrollo del software orientado a objetos.

La practica ha demostrado que si el programador no utiliza una metodología de diseño acorde con el paradigma OO los primeros resultados que se alcanzan al emplear un LOO muestran una codificación en un estilo procedimental que no explota las ventajas de la POO. Los que se inician en este paradigma frecuentemente presentan dificultades para expresar su problema en términos de objetos. Aunque dividir un problema en objetos y definir las acciones que son naturales para dichos objetos simplifica los programas; algunos programadores han dicho que dividir un problema en objetos es justamente un proceso de poner las cosas donde ellas deben de estar.

La transición hacia el empleo de LOOs tiene un impacto mucho más dramático sobre los programadores que, por ejemplo, el cambio de FORTRAN a C, porque los LOOs requieren que los programadores piensen en el problema que están tratando de resolver de una manera radicalmente diferente. El enfoque orientado a objetos significa abandonar una visión en la cual la interacción humano-computador es la principal, en favor de un paradigma centrado en el producto, dirigido por la relación productor-consumidor.

Para que un programador experimentado aprenda la POO se requieren varias etapas:

- Liberarse de los hábitos de programación procedimental y analizar los conceptos básicos de la POO a partir de un ejemplo.

Estudiar los conceptos de la POO y como se materializan en un LOO, preferiblemente un LOO puro (como Smalltalk o Actor).

- Resolver ejercicios simples.
- Estudiar las bibliotecas de las clases del LOO escogido.
- Desarrollar una aplicación real.

Cuando los programadores ganan experiencia en el empleo de las técnicas de POO ellos aprenden a crear clases que agrupan un comportamiento común, en una manera que se maximiza la reusabilidad del código sin añadir complejidad.

2.2 Análisis y Diseño Orientado a Objetos

La necesidad para desarrollar y mantener sistemas de software grandes y complejos en un ambiente dinámico y competitivo ha motivado el interés por nuevos enfoques para el diseño y desarrollo de software.

En el ciclo de vida del desarrollo de software aparecen tres fases esenciales: análisis, diseño e implementación. La fase de análisis abarca desde el inicio del proyecto, pasa al análisis de los usuarios y el estudio de la factibilidad; la fase de diseño comprende el diseño global, lógico, detallado, el diseño de programas y el diseño físico; luego de la etapa de diseño el programa es codificado y probado durante la fase de implementación. En el ciclo de vida del software orientado a objetos se identifican estas tres actividades pero se eliminan las fronteras entre ellas.

La razón primordial para que estas fronteras sean difusas es que los elementos de interés en cada una de las fases son los mismos: los objetos. Los objetos y las relaciones entre objetos son identificadas en las fases de análisis y diseño. Los objetos y las relaciones identificadas y documentadas en la fase de análisis sirven no sólo como entrada a la fase de diseño, sino también como una capa inicial en el diseño.

La segunda razón para que las fronteras sean difusas es que el proceso de desarrollo del software orientado a objetos es iterativo.

2.2.1 Análisis

En el A.O.O se pretende construir un modelo que represente de la mejor manera posible la situación del mundo real que se quiere sistematizar. Esta etapa se compone de un modelo estático, llamado diagrama de clases, un modelo dinámico y un modelo funcional.

El modelo estático o diagrama de clases contiene todos los elementos que juegan un papel importante y merecen ser representados como clases en el sistema. Se compone de la definición de las clases, sus atributos, métodos y las relaciones entre ellas. Para ello se pueden utilizar diversas notaciones, entre ellas el Lenguaje Unificado de Modelación (UML).

El modelo dinámico muestra la relación existente entre los eventos (sucesos que son generados por el usuario o creados por un objeto o por el sistema; por ejemplo, eventos de reloj, click con el mouse) y los estados de los objetos (situación en un instante determinado del objeto, valor de sus atributos y reacción ante un evento). Para esto se elabora un diagrama de estados y eventos, donde a través de flechas se representan los eventos y entre óvalos se escribe el estado del objeto. El fin de este diagrama es poder apreciar cuando cambian de estado los objetos; es decir, ante cuáles eventos reaccionan; tratando de anticiparse a lo que normalmente sucederá en tiempo de ejecución.

El modelo funcional especifica qué acciones (métodos o procesos) realiza el objeto cuando tiene lugar algún evento. Para ello se utiliza un diagrama de flujo de datos, donde se representan las relaciones funcionales, los datos que circulan y la interacción con elementos de almacenamiento de datos en el sistema.

Frecuentemente ocurre que al pasar de la situación del mundo real al modelo de clases estático se pasan por alto características importantes y funcionalidades necesarias, debido principalmente a la falta de práctica y por que se pasa muy rápido del modelo de clases al modelo dinámico y al funcional, sin comprender a fondo cómo se va a comportar el modelo de clases durante la ejecución.

Se necesita una ayuda que haga más real lo abstracto y contribuya además a disminuir los errores provocados por un incorrecto modelo de clases inicial. Es común observar que hacen falta clases, métodos o atributos, que cumplen un papel crucial en el funcionamiento del sistema. Igualmente la poca comprensión del modelo inicial asegura un modelo dinámico pobre y un modelo funcional incompleto. Por lo que se debe prestar un especial interés a estos aspectos.

2.2.2 Diseño

El diseño orientado a objetos (DOO) es el último paso dado en una progresión que ha conducido desde un enfoque puramente procedimental, a un enfoque basado en objetos y ahora al enfoque orientado a objetos. El paradigma de diseño procedimental utiliza la descomposición funcional para especificar las tareas a resolver. El enfoque basado en objetos presta más atención a la especificación de los datos que el enfoque procedimental, pero aún

utiliza la descomposición funcional para desarrollar la arquitectura de un sistema. El enfoque orientado a objetos va más lejos que el basado en objetos en cuanto al énfasis dado a los datos, utilizando las relaciones entre los objetos como una parte fundamental de la arquitectura del sistema.

El objetivo de diseñar componentes individuales de software es poder representar un concepto lo más cercano posible a su forma ejecutable. En el enfoque basado en objetos se usa con este propósito el tipo de dato abstracto. Mientras que en el enfoque orientado a objeto este papel lo juega la clase.

En resumen, si un sistema se desarrolla usando un análisis y diseño orientado a objetos y es implementado en un lenguaje de programación orientado a objetos, entonces los objetos y las clases identificadas durante el análisis son preservadas y enriquecidas durante el diseño y son implementadas directamente.

Los conceptos que caracterizan la POO mejoran el ciclo de vida y el mantenimiento de sistemas orientado a objetos. La herencia minimiza la cantidad de nuevo código que se necesita para añadir rasgos adicionales, y ella, junto con el polimorfismo y el enlace dinámico, minimizan la cantidad de código existente que tiene que ser cambiado cuando se requiere extender un sistema.

2.3 Metodología para el análisis y diseño de sistemas con POO

Los métodos de análisis y diseño orientados a objetos comparten los siguientes pasos básicos, aunque los detalles y el ordenamiento de los pasos pueden variar un poco:

- Encontrar los modos en los que el sistema interactúa con el entorno
- Identificar los objetos y los nombres de sus métodos y atributos.
- Establecer las relaciones entre objetos.
- Establecer las interfaces de cada objeto y el manejo de excepciones.
- Implementar y probar los objetos.
- Ensamblar y probar los sistemas.

Seguidamente se presenta la secuencia de pasos a seguir para desarrollar un sistema usando un enfoque orientado a objetos de acuerdo a la metodología descrita por Brian Henderson-

Sellers [49]

Paso 1: Acometer la especificación de los requisitos del sistema.

En esta etapa se realiza un análisis de alto nivel del sistema en términos de los objetos y los servicios que estos deben prestar.

Paso 2: Identificar los objetos.

Definir que entidades serán consideradas objetos, los atributos de estos y los servicios que ellos proporcionan.

Paso 3: Establecer las interacciones entre los objetos.

Se deben definir las relaciones existentes entre los objetos en términos de los servicios requeridos y los servicios proporcionados.

Paso 4: Unir el análisis con la etapa de diseño.

Detallar más la estructura de los objetos. Identificar componentes reusables existentes en diseños previos. La unidad de reutilización es la clase.

Algunas recomendaciones para hacer más fácil la reusabilidad de los componentes son:

- Realizar un agrupamiento conceptual de las clases para facilitar la localización de la clase apropiada cuando se necesite representar un concepto.
- Eliminar el acceso directo por un objeto a la implementación de otro.
- Reducir el número de dependencias entre las clases, incrementando la flexibilidad del diseño.
- Separar en los equipos de diseño la responsabilidad de construir componentes de propósito general de la responsabilidad de reutilizar clases.
- La reusabilidad de una clase no se alcanza desde un inicio, sino que se necesita un proceso iterativo de prueba y mejora.

Paso 5: Precisar la estructura interna de los objetos.

Usar diagramas de entidad-relación para describir más detalladamente la estructura interna de los objetos.

Paso 6: Introducir las relaciones de herencia jerárquica requeridas.

Paso 7: Agregar y generalizar clases.

Aunque algunos otros autores han definido otro tipo de metodología, como la descrita por Guerrero [42] en la cual la habilidad más importante en el análisis y diseño orientado a

objetos radica en asignar eficientemente las responsabilidades a los componentes de software y en segundo lugar aparece la determinación de las clases de objetos.

Algunas de las tareas a realizar en la etapa de **análisis** orientado a objetos son las siguientes:

1. Definir los requisitos.
2. Definir los casos de uso esenciales.
3. Crear y perfeccionar los diagramas de casos de uso.
4. Crear y perfeccionar el modelo conceptual.
5. Crear y perfeccionar el glosario.
6. Definir los diagramas de secuencia de los sistemas.
7. Definir los contratos de operaciones.

Algunas de las tareas a realizarse en la etapa de **diseño** son las siguientes:

1. Definir los casos reales de uso.
2. Definir los reportes, la interfaz de usuario y la secuencia de las pantallas.
3. Perfeccionar la arquitectura del sistema.
4. Definir los diagramas de interacción.
5. Definir los diagramas de diseño de clases.
6. Definir el esquema de la base de datos.

Durante el análisis y el diseño orientado a objetos, se procura identificar, describir y definir los objetos que finalmente serán implementados en un lenguaje de programación orientado a objetos.

La mayoría de los proyectos de software son complejos y la estrategia primaria para dominar la complejidad, es la descomposición (dividir para vencer). La estrategia es dividir el problema en unidades más pequeñas que sean manejables. Un enfoque tradicional para realizar esto es el análisis y diseño estructurado, donde se trata de descomponer el problema en funciones o procesos. Este método origina una división jerárquica de procesos constituidos por subprocesos Otra forma de realizar la descomposición, es usando un esquema de análisis y diseño orientado a objetos. En este esquema, se busca descomponer el problema en objetos y no en funciones.

2.4 Ventajas del análisis, diseño e implementación OO.

Se localizan los cambios requeridos y las interacciones inesperadas con otros módulos de programa son improbables.

La herencia y el polimorfismo hacen a los sistemas orientados a objetos más ampliables, contribuyendo esto a un desarrollo más rápido.

El diseño orientado a objetos es adecuado para la implementación distribuida, paralela o secuencial.

Los objetos corresponden más estrechamente con las entidades en los mundos conceptuales del diseñador y del usuario.

Se encapsulan las áreas de datos compartidas, lo cual reduce la posibilidad de modificaciones inesperadas u otras anomalías de actualización.

Las ventajas de usar el enfoque OO se traducen en un mejoramientos de la calidad a lo largo del ciclo de vida de una aplicación, facilitando además el mantenimiento y la creación de nuevas versiones que extiendan el programa.

Al disminuir las barreras entre las etapas de análisis, diseño y desarrollo, se garantiza que se está hablando de las mismas cosas y en los mismos términos desde el comienzo del análisis hasta el final de la etapa de implementación. Esto evita inconsistencias y permite verificar que las cosas están claramente definidas y satisfacen todos los requisitos, incluso antes de escribir alguna línea de código del programa. Las características de encapsulamiento, herencia y reutilización permiten crear un software mucho más robusto.

El hecho de modelar el mundo y no únicamente los datos necesarios para determinada aplicación, permiten crear diversas aplicaciones sobre la misma información sin necesidad de repetir las tareas de análisis. Esto permite concentrarse en cumplir los requisitos de la aplicación basándose en las facilidades que ofrecen los objetos del mundo ya modelados.

Se pueden enunciar otros beneficios del enfoque orientado a objetos:

- Reutilización de software: permite describir clases y objetos que podrán ser usados en otras aplicaciones.
- Estabilidad: el diseñador piensa en términos de comportamiento de objetos, no en detalles de bajo nivel.
- Diseño rápido y de alta calidad: puesto que se concentra en satisfacer los requisitos y no en detalles técnicos.

- Integridad: facilidad de programación al usar efectivamente toda la información de la fase de diseño, poniéndola en términos de un lenguaje específico.
- Facilidad de mantenimiento: dado que al tener el modelo del mundo, es fácil realizar mantenimiento en términos de objetos, atributos y métodos de los mismos.
- Independencia en el diseño: el diseño de un software se puede hacer independientemente de plataformas, software y hardware.

Lo que es verdaderamente revolucionario de la POO es que esta ayuda a los programadores a usar código existente, tal como los chips de silicio permiten a los constructores de circuitos reutilizar el trabajo de los diseñadores de tales pastillas. Esta habilidad para definir nuevas facilidades especializadas a partir de facilidades genéricas disponibles en una biblioteca es una de las cosas claves que hacen a la POO tan productiva y reduce el volumen de código necesario. Es posible para los programadores escribir clases que pueden ser consideradas como “cajas negras” y por lo tanto puedan ser incluidas en los programas con poca separación del código no necesario. Si ellas están bien diseñadas (como componentes no complejas, débilmente acopladas y altamente coherentes), los programadores pueden verlas como partes de software estándar a ser incluídas en futuras aplicaciones. El acople entre la herencia y el enlace dinámico facilitan el reuso de código. También se incrementan la modularidad, flexibilidad y confiabilidad, ayuda a separar los procedimientos y sus especificaciones de representación.

En resumen, la POO está llamada a reducir tanto el tiempo de desarrollo como el costo de mantenimiento, a simplificar la creación de nuevos sistemas y de nuevas versiones de sistemas ya existentes, pero la POO no sería del todo eficiente sin disponer de un adecuado análisis y diseño; para lograr estas ventajas es necesario lograr un verdadero análisis y diseño orientado a objetos, que permita producir componentes reusables, o sea, es necesario planificar la reusabilidad de las componentes.

2.4.1 Reusabilidad

La aparición del paradigma y de los lenguajes orientados a objetos responde a otro problema que la programación estructurada no había podido resolver satisfactoriamente: la reutilización. La reutilización es un pilar básico de la ingeniería del software para lograr

abaratar los costes de desarrollo y mantenimiento de software, y por tanto aunque ya fue mencionada en apartados anteriores, conviene profundizar un poco más en este aspecto.

A pesar de que varios lenguajes ofrecían abstracciones modulares y que ya en 1969 McIlroy [61] proponía la construcción de software en base a componentes reutilizables como solución a la crisis del software, la tasa de reutilización durante el desarrollo de nuevo software era muy baja. La abstracción del objeto como tipo abstracto de datos que aúna en una misma entidad encapsulada datos y funcionalidad, presentando una determinada interfaz al cliente, prometía ofrecer esta reusabilidad. Meyer [65] propone el "diseño por contrato", para promover el diseño de objetos como cajas negras, cuya implementación no tiene por qué ser conocida por el cliente, sino que simplemente respondan a un contrato o interfaz requerido, soportando así mayor adaptabilidad, al poder modificarse la implementación de un objeto sin afectar a otras partes del sistema, siempre que se mantenga la misma interfaz.

Idealmente, las clases o tipos abstractos de datos podrían ser el producto de la reutilización de piezas de software desarrolladas para una determinada aplicación, cuya funcionalidad es lo suficientemente genérica como para resultar útil en otro contexto diferente para el que originalmente se desarrolló. Sin embargo y frente a lo esperado, generalmente la reusabilidad del software orientado a objetos es baja. El uso de métodos y lenguajes orientados a objetos no lleva por sí solo a la reutilización [92] [90] . Los métodos de análisis y diseño orientados a objetos normalmente se basan en el dominio del problema, y por ello producen normalmente diseños basados en objetos específicos del problema. Al no hacer una clara separación entre funcionalidad y composición, oscurecen la arquitectura de objetos del software, dificultando tanto su mantenibilidad y adaptabilidad como la reutilización de las piezas existentes. Además, muchos de estos métodos parten de la suposición de que el software se diseñará por completo desde cero, por lo que no se incorporan adecuadamente ni la reutilización de componentes existentes ni el desarrollo orientado a la reusabilidad en el ciclo de desarrollo del sofware [92] [78] [90] [106] . A pesar de estos problemas, hay pocas dudas de que el paradigma de la orientación a objetos es una herramienta de gran valor en el desarrollo de software, pero también está claro que se necesita algo más para lograr el grado de reutilización necesario que pueda solventar la crisis del software.

En este sentido, el desarrollo de software basado en componentes se perfila como el paradigma más prometedor para conseguir la construcción de software en base a

componentes reutilizables [9] [63] [71] [93] [106] . Frente a las clases o prototipos de objetos, los componentes han de ofrecer ciertos requisitos como la posibilidad de su uso independiente de otras piezas, no persistencia y compatibilidad binaria [106] . Estas condiciones pocas veces son reunidas por las clases definidas en el contexto de una aplicación, de forma que la programación de un componente ha de ser normalmente deliberada. Es de señalar que la programación orientada a objetos se ha mostrado especialmente adecuada tanto para la programación de componentes como para la programación basada en componentes.

El desarrollo de software basado en componentes se centra sobre todo en resolver problemas de composición [92] , en determinar cómo construir aplicaciones en base a componentes reutilizables, [93] y en investigar las arquitecturas de software y cómo éstas influyen en la reusabilidad de los componentes [28] [41] . Pero evidentemente, las bases de la programación orientada a componentes son la propia existencia de componentes reutilizables, su interoperabilidad y su adaptabilidad [28] [92] . Dentro de este paradigma, generalmente es más conveniente tratar a los componentes como cajas negras, es decir, como una entidad encapsulada con una interfaz de servicios bien definida y una implementación oculta a la que los clientes de estos servicios no tienen acceso [9] [63] [92] , ya que se considera generalmente que la reutilización de código es menos ventajosa que la reutilización de componentes [9] [63] [106] . Actualmente la tecnología orientada a componentes dispone de gran cantidad de recursos, desde entornos de desarrollo visual como por ejemplo Delphi [14] o Visual Basic [66] , hasta arquitecturas distribuidas como DCOM [83] , JavaBeans [105] o CORBA [73] .

Aunque todavía quedan por resolver numerosos problemas en el camino hacia el completo desarrollo del paradigma del software basado en componentes, puede decirse que mucho del progreso de desarrollo del software moderno está basado en el concepto de reusabilidad. Al principio, la reusabilidad significó el uso de subprogramas, pero más recientemente paradigmas más poderosos de programación orientada a objetos y arquitectura de componentes, han sido estrategias particularmente eficaces para producir software fiables y poderosos que empiezan donde el software anterior acabó.

2.5 Recomendaciones para la definición de los objetos

Seguidamente se expone la secuencia de pasos a desarrollar para reconocer y definir los objetos, propuesta por Elizabeth Gibson [44] y a la que se le conoce como metodología OBA (*Object Behavior Analysis*). Esta metodología responde a las interrogantes siguientes:

¿Cómo reconocer los objetos inicialmente?

¿Cuáles son sus características?

¿Cuáles son las relaciones entre los objetos?

¿Cómo ellos interactúan?

La metodología OBA parte del criterio de que la implementación esta basada en el diseño y el diseño se basa en el análisis; luego el análisis debe soportar las nociones básicas del paradigma de programación seleccionado; en este caso la POO. Los pasos a ejecutar tienen como propósito comprender la aplicación e identificar los comportamientos iniciales, definir los objetos que exhiben esos comportamientos, clasificar los objetos, identificar las relaciones entre ellos y modelar su ciclo de vida.

Paso 1: Identificar los comportamientos.

Entrevistar a los usuarios de la futura aplicación y observar como ellos actúan para determinar qué hacen, con quien y para qué interactúan, en que orden y qué produce cada acción. Lo principal en este paso es definir una lista de deseos y necesidades que debe satisfacer el comportamiento del sistema. Hay algunos principios que deben tenerse en cuenta cuando se realiza una entrevista:

- No permitir que las suposiciones o conocimientos propios sobre un área de aplicación le perjudique escuchar lo que el entrevistado está diciendo.
- Mantener una actitud abierta, no dirigir preguntas como "Dígame ¿qué usted hace primero cuando aborda esta parte de su trabajo?", ni preguntas cerradas como "yo apuesto a que la primera cosa que usted hace es chequear esto, ¿correcto?".
- No usar preguntas de selección múltiple, usar preguntas simples.
- Usar técnicas de audición activa.

Para definir el comportamiento del sistema y sus componentes trate de encontrar una respuesta a "¿qué tiene que hacer el sistema?". Es útil pensar en términos del papel que debe desempeñar el sistema y en sus responsabilidades. El comportamiento del sistema se describe

en términos de sus responsabilidades, quién o qué es responsable por cada comportamiento particular y quién o qué interactúa con el responsable.

Otra fuente de información útil es cualquier especificación de los requisitos funcionales que existan acerca del futuro sistema.

Paso 2: Definir los objetos.

Para definir los objetos es necesario determinar quién o qué es responsable de los comportamientos particulares. Los objetos de la aplicación son los objetos concretos, conceptos, procesos y eventos que exhiben un comportamiento.

La salida de este paso es una lista de los objetos, los grupos de objetos que se relacionan por su comportamiento y las propiedades visibles de éstos.

Paso 3: Clasificar los objetos.

Significa agrupar los objetos de acuerdo a alguna similaridad de su comportamiento y estado (función y forma). Se puede crear un objeto abstracto a partir de los elementos comunes de dos o más objetos concretos.

La salida de este paso es un diagrama de relaciones jerárquicas entre objetos creadas en base a los objetos abstractos formados usando un criterio de comportamiento similar.

Paso 4: Identificar las relaciones.

En este paso se trata de extraer una descripción preliminar de las relaciones entre objetos. Usando este esquema se puede crear una tabla que aclare las relaciones que cada objeto tiene con los restantes objetos. Estas relaciones pueden ser de subordinación, de dependencia (el comportamiento de un objeto dirige automáticamente el comportamiento de otro objeto) o de comunicación (un objeto requiere información de otro objeto, enviarle información a éste o conocer su estado).

Paso 5: Modelar los procesos.

Finalmente, se tiene que determinar cuáles objetos inician las actividades e identificar la secuencia de las actividades. Al determinar los objetos y especificar sus interfaces se está definiendo un lenguaje de programación

2.5 Recomendaciones para el diseño de la jerarquía de clases

El resultado de un diseño orientado a objetos (DOO) es una lista de definiciones de clases. Un diseño esta completo cuando todos los objetos y todas las operaciones están definidos. En el DOO se usan los nombres de los objetos como un punto inicial para determinar las clases de objetos a ser diseñadas. Una respuesta positiva a cualquiera de las preguntas siguientes indica que se debe crear una nueva clase.

- Cumplir una responsabilidad ¿tiene su propio significado?
- Una responsabilidad ¿es cumplida por varias entidades?
- Una responsabilidad ¿es compleja?
- El cumplimiento de la responsabilidad ¿le interesa relativamente a pocos usuarios?
- Para cumplir la responsabilidad ¿se usa poco la información de la entidad?

A continuación aparecen algunas recomendaciones presentadas por Ralph E. Johnson [53] para diseñar la jerarquía de clases.

Sobre los **métodos:**

Reducir el número de argumentos.

Reducir el tamaño de los métodos.

Minimizar el acceso a las variables.

Sobre las **clases**:

Si una subclase implementa un método de un modo y otras lo implementan de otro modo entonces la implementación de ese método es independiente de la superclase.

Una instancia de una clase A no debe enviar un mensaje directamente a una componente de la clase B.

Un operador debe ser público, si y sólo si, él debe estar disponible a los usuarios de las instancias de la clase.

Cada operador que pertenece a una clase accede o modifica algunos de los datos de la clase.

Una clase debe ser dependiente de la menor cantidad de otras clases como sea posible.

La interacción entre dos clases debe ser explícita (reducir la información global).

Sobre cuándo crear **clases abstractas**:

La jerarquía de clases debe ser profunda y estrecha.

El tope de la jerarquía de clases debe ser una clase abstracta.

Las clases grandes deben ser divididas en clases más pequeñas, de modo que las subclases sean especializaciones (cada subclase debe ser desarrollada como una especialización).

Dividir las clases grandes (una clase con más de 50 métodos se debe dividir).

Una clase debe ser dividida cuando la mitad de sus métodos accedan a la mitad de sus variables de instancias y la otra mitad de los métodos accede al resto de las variables.

2.7 El Lenguaje Unificado de Modelado (UML)

Cualquier rama de la ingeniería o arquitectura ha encontrado útil desde hace mucho tiempo la representación de los diseños de forma gráfica. Desde los inicios de la informática se han estado utilizando distintas formas de representar los diseños de una forma más bien personal o con algún modelo gráfico. La falta de estandarización sobre la manera de representar gráficamente un modelo impedía que los diseños gráficos realizados se pudieran compartir fácilmente entre distintos diseñadores.

Se necesitaba por tanto un lenguaje no sólo para comunicar las ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo se creó el Lenguaje Unificado de Modelado (UML: *Unified Modeling Language*). UML se ha convertido en ese estándar tan ansiado para representar y modelar la información con la que se trabaja en las fases de análisis y especialmente, de diseño.

El lenguaje UML posee una notación gráfica muy expresiva, que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue.

2.7.1 Historia del UML

El lenguaje UML comenzó a gestarse en octubre de 1994 [12] , cuando Rumbaugh se unió a la compañía Rational fundada por Booch (dos reputados investigadores en el área de metodología del software). El objetivo de ambos era unificar dos métodos que habían

desarrollado: el método Booch y el OMT (*Object Modelling Tool*). El primer borrador apareció en octubre de 1995. En esa misma época otro reputado investigador, Jacobson, se unió a Rational y se incluyeron ideas suyas. Estas tres personas son conocidas como los “tres amigos”. Además, este lenguaje se abrió a la colaboración de otras empresas para que aportaran sus ideas. Todas estas colaboraciones condujeron a la definición de la versión inicial de UML.

Esta primera versión se presentó al Grupo de Gestión de Objetos (OMG: *Object Management Group*) que en 1997 lo reconoció como un estándar. Este grupo, que gestiona estándares relacionados con la tecnología orientada a objetos (metodologías, bases de datos OO, CORBA, etc.), propuso una serie de modificaciones y una nueva versión de UML (la 1.1), que fue adoptada por el OMG como estándar en noviembre de 1997. Desde aquella versión se han hecho varias revisiones, gestionadas por la OMG *Revision Task Force*.

2.7.2 ¿Qué es UML?

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para establecer una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema.

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

• Mayor rigor en la especificación.

• Permite realizar una verificación y validación del modelo realizado.

• Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos). Esto permite que el modelo y el código estén actualizados, con lo que siempre se puede mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones principales:

• Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otra persona lo pueda entender.

• Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.

• Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.

• Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado y pueden utilizarse para su futura revisión.

Aunque UML está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, tales como flujos de trabajo (*workflows*) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

Un modelo UML esta compuesto por tres clases de bloques constructivos:

• Elementos: son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)

• Relaciones: relacionan los elementos entre sí.

• Diagramas: son colecciones de elementos con sus relaciones.

2.7.3 Diagramas UML

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones. En concreto, un diagrama ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas. UML incluye los siguientes diagramas:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de objetos.
- Diagrama de secuencia.
- Diagrama de colaboración.
- Diagrama de estados.
- Diagrama de actividades.
- Diagrama de componentes.
- Diagrama de despliegue.

Los diagramas más interesantes (y los más usados) son los de casos de uso, clases y secuencia, por lo que nos centraremos en ellos. Pare ello, utilizaremos ejemplos de un sistema de venta de entradas de cine por Internet.

El diagrama de casos de uso representa gráficamente las funcionalidades asociadas a un sistema. Se define un caso de uso como cada interacción supuesta con el sistema a desarrollar, donde se representan los requisitos funcionales. Es decir, se está diciendo lo que tiene que hacer un sistema y cómo. En la figura (Ver Anexo 2) se muestra un ejemplo de este tipo de diagrama, donde aparecen tres actores (los clientes, los taquilleros y los jefes de taquilla) y las operaciones que pueden realizar (sus roles).

El diagrama de clases muestra un conjunto de clases, interfaces y sus relaciones. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos. En la figura (Ver Anexo 3) se muestran las clases globales, sus atributos y las relaciones de una posible solución al problema de la venta de entradas.

En el diagrama de secuencia se muestra la interacción de los objetos que componen un sistema de forma temporal. Siguiendo el ejemplo de venta de entradas, la figura (Ver Anexo 4) muestra la interacción de crear una nueva sala para un espectáculo.

El resto de diagramas muestran distintos aspectos del sistema a modelar. Para modelar el comportamiento dinámico del sistema están los diagramas de interacción, colaboración, estados y actividades. Los diagramas de componentes y despliegue están enfocados a la implementación del sistema.

2.7.4 Proceso de desarrollo

Aunque UML es bastante independiente del proceso de desarrollo que se siga, sus mismos creadores han propuesto su propia metodología de desarrollo, denominada el Proceso Unificado de Desarrollo [52] .

El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidos. Además, el Proceso Unificado utiliza el UML para expresar gráficamente todos los esquemas de un sistema software. Pero realmente, los aspectos que definen este Proceso Unificado son tres: es iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura [50] :

• Dirigido por casos de uso: Basándose en los casos de uso, los desarrolladores crean un conjunto de modelos de diseño e implementación que los llevan a cabo. Además, estos

modelos se validan para que sean conformes a los casos de uso. Los casos de uso también sirven para realizar pruebas sobre los componentes desarrollados.

• Centrado en la arquitectura: Antes de construir un edificio éste se enfoca desde varios puntos de vista: estructura, conducciones eléctricas, fontanería, etc. Cada uno de estos aspectos está representado por un gráfico con su notación correspondiente. Siguiendo este ejemplo, el concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema.

• Iterativo e incremental: Todo sistema informático complejo supone un gran esfuerzo que puede durar desde varios meses hasta años. Por lo tanto, lo más práctico es dividir un proyecto en varias fases. Actualmente se suele hablar de ciclos de vida en los que se realizan varios recorridos por todas las fases. Cada recorrido por las fases se denomina iteración en el proyecto en la que se realizan varios tipos de trabajo (denominados flujos). Además, cada iteración parte de la anterior incrementado o revisando la funcionalidad implementada, (Ver Anexo 5).

Resumiendo, el Proceso Unificado es un modelo complejo con mucha terminología propia, pensado principalmente para el desarrollo de grandes proyectos. Es un proceso que puede adaptarse y extenderse en función de las necesidades de cada empresa.

2.8 Análisis y diseño orientado a objetos con el Lenguaje Unificado de Modelado (UML).

UML promueve la reusabilidad, con la inserción en el diseño de componentes reutilizables. Permite presentar detalles esenciales de un problema complejo y filtrar los detalles que no lo son. Proporciona además un mecanismo para ver el sistema a desarrollar desde diferentes perspectivas.

Diseñar un modelo para sistemas de software es tan esencial como tener un plano para ejecutar una construcción grande. Los buenos modelos:

- Identifican requisitos y comunican información

- Se enfocan sobre cómo interactúan los componentes del sistema, sin caer en los detalles específicos
- Permiten ver las relaciones entre los componentes del diseño
- Mejoran la comunicación dentro del equipo por el uso de un idioma gráfico común

2.8.1 Elementos de los modelos de la notación UML.

Casos de uso: secuencias de transacciones realizadas con el sistema en respuesta a eventos "disparadores", iniciados por un actor.

Actores: estereotipos de una clase que interactúa con el software en desarrollo. Típicamente son personas y otros sistemas de software.

Clases: descripciones de la estructura y comportamiento comunes a un conjunto de objetos. Las clases proveen el comportamiento estático del sistema; los diagramas de estado se usan para describir el comportamiento dinámico.

Paquetes de clases: agrupaciones (*clusters*) de clases lógicamente relacionadas. En UML, tienen un módulo de especificación y uno de implementación (llamada comúnmente "cuerpo del paquete".).

Objetos: entidades dotadas de estado, comportamiento e identidad.

Operaciones: servicios provistos por una clase o por los objetos de una clase. Tienen asociado un tipo de retorno.

Componentes: módulos de software definidos durante el diseño. Incluyen programas principales, subprogramas, paquetes y tareas.

Paquetes de componentes: agrupaciones de componentes lógicamente relacionados. Poseen también un módulo de especificación y uno de implementación.

Nodos de procesamiento: componentes hardware capaces de ejecutar programas.

Nodos de dispositivos: componentes hardware que no poseen capacidad de cómputo (módems, terminales.)

2.8.2 Formas de utilizar el UML.

Mostrar las fronteras de un sistema y sus funciones principales, usando los casos de uso y actores.

Ilustrar los casos de uso mediante diagramas de relaciones.

Representar la estructura estática de un sistema usando el diagrama de clases.

Modelar el comportamiento de los objetos con los diagramas de transición de estados.

Revelar la arquitectura física de implementación con los diagramas de componentes y de despliegue.

Extender la funcionalidad con estereotipos.

2.8.3 Vistas del sistema en UML.

Vista de casos de uso: perspectiva del sistema para los usuarios (incluyendo a los sistemas externos.)

Vista lógica: información sobre las clases y las entidades que ellas representan.

Vista de componentes: muestra cómo la solución se corresponde con los módulos de software.

Vista de despliegue: muestra la correspondencia entre los procesos y el hardware.

2.8.4 Procedimiento para pasar al diseño de clases.

Revisar las anotaciones de la entrevista con los clientes y usuarios potenciales. Resaltar los nombres.

Tomar cada nombre utilizado en el ámbito del software como base para una clase de objetos. Fijarse especialmente en los nombres de personas (actores.)

Identificar todo lo que un objeto miembro de una clase deba recordar y hacerlo una variable de instancia. Si hay algo que varios objetos deban recordar, hacerlo una variable de la clase.

Fijarse en los verbos usados e identificar a qué nombres (o clases) se aplican. Tomarlos como base para las operaciones. Si la clase en conjunto debe responder a la petición, hacerlo una operación de la clase.

Buscar los lugares en que se pueda decir que una clase es una especie de otra clase (relación *is-a*) Disponer estas clases como una jerarquía de generalización /especialización.
Buscar lugares donde un objeto de una clase es un miembro componente de otra clase. Esta es la relación de agregación (*is a part of*.)
Asegurarse de que para cada clase existen formas de crear los objetos y de destruirlos.
Asegurarse de que cada atributo sea inicializado, situado y leído.

2.8.5 El RUP antes y durante el análisis y diseño.

El ciclo de vida del RUP es un conjunto de ciclos sucesivos que abarca varias fases, cada una de las cuales tiene varias iteraciones (mini proyectos). Seguidamente se muestra el procedimiento seguido antes y durante el análisis y diseño.
Captación de requisitos: seleccionar/definir los casos de uso a ser implementados en esta iteración. Actualizar el modelo del objeto para reflejar las clases del dominio y las asociaciones adicionales descubiertas. Desarrollar un plan de prueba para la iteración.
Análisis y diseño: determinar las clases a desarrollar o actualizar en esta iteración. Actualizar el modelo del objeto para reflejar las clases del dominio y las asociaciones adicionales descubiertas. Actualizar el documento de arquitectura si se necesita. Comenzar el desarrollo de procedimientos de prueba.

2.9 La herramienta Rational Rose.

Rational Rose es una herramienta CASE comercial que apoya el trabajo con UML. Comienza por defecto con dos presentaciones, una para los diagramas de casos de uso y la otra para los diagramas de clases. Pero ésta no es la separación más sensible: se puede colocar cualquier tipo de diagrama en cualquier presentación y mezclar clases, casos de uso y actores en un diagrama como se juzgue apropiado. Los casos de uso pueden arrastrarse dentro de los diagramas de clases y viceversa. Se pueden mezclar en un diagrama o no, según se prefiera.
Los diagramas de clases y los diagramas de transición de estados están siempre separados. Además, no se permite más de un diagrama de transición de estados por clase.

Rose se integra bien con otras herramientas de Rational para manejo de la configuración, prueba, documentación requerida y otros.

2.9.1 Ventajas de Rational Rose.

El desarrollo guiado por el modelo resulta en un incremento de la productividad del desarrollador.

El desarrollo de casos de uso enfocado sobre la empresa tiene como consecuencia un mejoramiento en la calidad del software.

El uso de un lenguaje estándar común (UML) mejora la comunicación del equipo

Las capacidades para ingeniería inversa permiten la integración con sistemas orientados a objeto heredados (*legacy systems*).

Los modelos y la codificación permanecen sincronizados durante el ciclo de desarrollo.

Conclusiones parciales del capítulo.

La programación orientada a objetos proporciona ventajas en el análisis, diseño, producción y mantenimiento del software, menores tiempos de desarrollo, un alto grado de compartición del código y flexibilidad. Estas ventajas hacen de este enfoque de la programación un recurso importante para construir complejos sistemas de software.

Para los que desarrollan software, la programación orientada a objetos ofrece la oportunidad de producir aplicaciones basadas en la reutilización del trabajo de otros desarrolladores. Esto constituye un reto para aquellos que han trabajado siempre solos, usando sus propias herramientas y estilos.

La utilización de UML permite lograr un mayor rigor en la especificación de requisitos, realizar una verificación y validación del modelo y representar gráficamente un sistema de forma que otra persona lo pueda entender con facilidad. Por esta vía se pueden especificar las características de un sistema antes de su construcción. La herramienta Rational Rose, que está basada en UML, aporta significativas ventajas en cuanto a la calidad y productividad del proceso de desarrollo del software.

Capítulo 3

3. Incorporación del enfoque orientado a objeto al proceso de desarrollo del software educativo

Objetivos específicos.

Proporcionar una visión integradora de las propiedades y métodos del diseño instruccional y de la ingeniería del software orientado a objeto con el fin de facilitar y elevar la calidad del proceso de desarrollo del software educativo y de los productos obtenidos a partir del mismo.

3.1 Software educativo de modelación y simulación OO.

La orientación a objetos es un paradigma que se ha aplicado a la construcción de software en múltiples dominios, incluyendo el modelado y la simulación basados en computador, tanto a nivel científico como educativo. Al mismo tiempo, se está tendiendo a crear software más abierto, fácilmente ampliable, reutilizable y que permita su integración con otras herramientas. Por todo lo cual es de gran utilidad trabajar con la perspectiva de los objetos de aprendizaje reutilizables que se abordan un poco más adelante en el capítulo.

Tanto el análisis como el diseño y la programación orientados a objetos se vienen utilizando de forma creciente en el desarrollo de una gran variedad de software en las últimas décadas. El software orientado al modelado y simulación de sistemas no es una excepción. Por ejemplo, podemos encontrar aplicaciones de modelado que permiten añadir algoritmos o componentes de forma sencilla [24] , y bibliotecas de objetos reutilizables para crear simulaciones [57] [68] . Ya existen trabajos en este sentido, no sólo en el ámbito del software educativo, sino también, por ejemplo, en el ámbito de las aplicaciones gráficas[19] , del courseware [48] , o del modelado y la simulación [74] .

Las capacidades de los lenguajes orientados a objetos de abstracción, encapsulación, polimorfismo y herencia, unidas a las facilidades de integración en Internet [22] han hecho que, por ejemplo, el lenguaje Java [88] [91] sea considerada una herramienta atractiva para desarrollar y distribuir software educativo [79] , de lo que ya existen algunos ejemplos [21] .

Dentro del software educativo, las aplicaciones de modelado y simulación ocupan un lugar importante, especialmente en la educación universitaria. Es fácil establecer similitudes entre los entornos de modelado y simulación en los ámbitos científico y educativo. Tanto el hecho de que las técnicas de ingeniería de software utilizadas no diferirán sustancialmente como que, especialmente en el ámbito universitario, no exista, en muchos casos, una excesiva distancia entre ambos tipos de entornos, pueden ser apuntados como motivos de estas similitudes.

Seguidamente se presentan algunas consideraciones sobre la creación e integración de software educativo de modelado y simulación usando programación orientada a objetos, con perspectivas de creación de objetos de aprendizaje para la integración de aplicaciones como un medio de posibilitar la ampliación de entornos de aprendizaje basados en software, y su relación con la programación orientada a objetos. Todo ello surge como consecuencia del desarrollo de varias aplicaciones de modelado y simulación construidas para servir como base de distintos entornos de aprendizaje utilizados en la Universidad de Vigo. Estas aplicaciones son Model-Lab [85] , un software de modelado y simulación especialmente orientado a la dinámica de ecosistemas, Neuro-Lab, un entorno de diseño y simulación de redes de neuronas artificiales, y Expert-Lab, un simulador de sistemas expertos basados en reglas.

3.1.1 Estructura de clases, ampliabilidad y reutilización en software de modelado y simulación

El paradigma de la orientación a objetos se adapta de forma especialmente óptima a la construcción de simulaciones de sistemas basadas en computador, tanto a la hora de modelar las entidades que forman parte del dominio como en la implementación misma de software de simulación, ya que la programación orientada a objetos se basa en la idea de la interacción entre representaciones abstractas de objetos reales. Esto supone que en muchos casos la estructura del software tenga una gran similitud con la del sistema simulado [60] [99] .

Usando este paradigma, se desarrolló el software Neuro-Lab de diseño y simulación de redes de neuronas artificiales, destinado a utilizarse como apoyo en la docencia para los alumnos de Ingeniería Informática de la Universidad de Vigo. Existen múltiples entornos de

simulación de redes de neuronas, algunos de los cuales han sido utilizados para propósitos 99educativos [37] . Se pueden encontrar, en la literatura didáctica sobre el tema, propuestas para crear implementaciones en software de redes de neuronas basándose en matrices y lenguajes no orientados a objetos [36] . Más recientemente se ha desarrollado software de simulación en este ámbito basándose en lenguajes orientados a objetos [21] . También la aplicación de la orientación a objetos ha permitido la creación de bibliotecas de objetos que implementan redes de neuronas, que pueden ofrecer persistencia incluida en las clases [57] . Otro enfoque utilizado, tanto en el campo de la simulación de redes de neuronas como en otro tipo de simulaciones, ha sido usar una aproximación más jerárquica a la hora de definir el número y tipo de clases a crear [62] [24] . Neuro-Lab se ha desarrollado siguiendo esta línea, implementando cada componente de una red de neuronas como una clase de objeto, lo que aumenta en gran medida la flexibilidad del software resultante, al permitir por ejemplo añadir nuevos tipos de neuronas o definir nuevos tipos de redes simplemente heredando y refinando una clase genérica. Los métodos de entrenamiento se han separado de la implementación de la red, posibilitando así la adición de nuevos métodos de forma sencilla. De forma similar, las funciones de transferencia también se han implementado separadamente, permitiendo así tanto la adición de nuevas funciones como la creación de redes con neuronas que utilicen diferentes funciones de transferencia.

A la hora de implementar la parte de presentación de Neuro-Lab, donde debía mostrarse al usuario la red de forma gráfica y ofrecerle una interfaz intuitiva para su manejo, se optó por utilizar un conjunto de clases de objetos visuales que se correspondieran con las clases de objetos definidos para componer una red susceptibles de ser mostrados gráficamente. En tiempo de ejecución, se establece una ligadura lógica entre cada par de objetos no visual-visual, donde el primero implementa el funcionamiento de un componente de la red, por ejemplo una neurona, de forma independiente del interfaz, mientras que el segundo maneja la presentación gráfica y responde a las posibles acciones del usuario. Existe pues total independencia entre la implementación de la red y su presentación visual. Esto aporta gran flexibilidad a la hora de modificar la presentación, y por otro lado, no obliga a la creación de elementos visuales para poder construir y simular una red, permitiendo la creación casi directa de bibliotecas de clases reutilizables a partir del código existente. Por el contrario, la separación lógica entre los objetos y sus correspondientes representaciones visuales implica

mayor complejidad en el desarrollo, ya que han de implementarse los mecanismos de conexión entre ambos para sincronizar las respuestas a los cambios que se produzcan en cualquiera de ellos. Consideramos que la estructura basada en pares de clases visual-no visual facilita esta tarea, ya que existe una simetría evidente entre las clases y por tanto en las funciones a desarrollar en cada una de ellas para posibilitar su interrelación.

Un enfoque distinto se usó en el desarrollo del software de modelado matemático Model-Lab [85] . Se siguió una metodología de descomposición jerárquica similar para estructurar las clases, de forma que un modelo es una clase de objeto compuesta por objetos que representan los componentes del modelo, y éstos a su vez contienen objetos que representan los términos (ecuaciones o constantes) que definen el funcionamiento de cada componente.

Cada componente se encarga tanto de la lógica de funcionamiento interno como la del interfaz de usuario y visualización. Esto simplifica la programación al no existir necesidad de comunicar objetos separados, como ocurría en el caso de Neuro-Lab. Por otro lado, se genera una dependencia de la presentación de los objetos, lo que dificultaría una hipotética integración con otras herramientas donde sólo se quisiera disponer de la lógica de funcionamiento de los objetos, pero no de su interfaz de usuario o visualización. La elección de esta solución, a pesar de sus posibles inconvenientes, parece aceptable cuando se trate de crear un software poco susceptible de ser reutilizado como parte de un entorno mayor, o de ser exportadas sus clases como bibliotecas. Model-Lab cumple estos requisitos, ya que es más susceptible de integrar otros elementos dentro de su entorno que de formar parte de otro.

Por otro lado, en Model-Lab se crearon clases dedicadas para el manejo e interpretación de expresiones y funciones matemáticas, lo que permite su reutilización de forma sencilla. Además se posibilita la ampliación de la biblioteca de funciones disponibles para crear modelos sin tener que modificar el código de la aplicación, pudiendo incluso integrar en Model-Lab bibliotecas de funciones creadas y compiladas en otros lenguajes. Aunque esto supone un importante aumento de la complejidad a la hora de crear la clase que implementa esta funcionalidad, este esfuerzo se ve compensado por una mayor potencia y flexibilidad y la disminución de las necesidades de mantenimiento.

Expert-Lab es un ejemplo más de software educativo orientado a objetos: mantiene una total independencia, al igual que Neuro-Lab, de la interfaz de entrada/salida con el usuario. El objetivo de Expert-Lab es presentar al alumno universitario un sistema experto basado en

lógica proposicional, capaz de explicar su funcionamiento. Cada regla y cada hecho dentro de Expert-Lab está representado por un objeto, que es capaz de autodescribirse, por ejemplo, si el objeto es una regla, es capaz de devolver una cadena que representa de forma adecuada sus antecedentes y sus consecuentes; de esta forma, la representación en la interfaz es unívoca uno a uno, si bien existe una separación física entre el objeto representado en la interfaz y el objeto real de datos dentro del programa.

3.1.1.1 Integración de entornos de modelado y simulación

Se ha apuntado que la integración con otras herramientas software puede ser una característica importante de un entorno de modelado y simulación [107] . La interdisciplinaridad de determinados métodos o herramientas a menudo plantea la conveniencia de integrar entornos inicialmente creados separadamente. Es el caso, por ejemplo, de las redes de neuronas artificiales, que se han ido aplicando de forma creciente a ámbitos muy diversos, que incluyen la economía, la biología o la medicina. Esto ha llevado a considerar la posibilidad de añadir funciones basadas en redes de neuronas artificiales en el entorno de modelado orientado a la ecología Model-Lab; y así mismo, elementos basados en lógica proposicional a resolver por Expert-Lab. Al disponer de la herramienta Neuro-Lab, y Expert-Lab, se decidió integrarlas en el entorno, de forma que las redes de neuronas o los sistemas lógicos diseñados en este último fueran utilizables desde Model-Lab. Al haber diseñado Neuro-Lab y Expert-Lab usando una estructura jerárquica de clases, la creación de una biblioteca de funciones es casi inmediata, ya que basta incluir en ella las clases deseadas, en este caso la clase que implementa una red de neuronas, que al ofrecer además persistencia, cubre las funcionalidades deseadas de poder utilizar diseños previamente realizados con las herramientas Neuro-Lab/Expert-Lab. La adición de las funciones pertinentes a Model-Lab, como se ha descrito anteriormente, se realiza sin necesidad de modificar el código, sino a través de una aplicación de mantenimiento de la biblioteca de funciones.

Resumiendo podemos decir que el paradigma de la orientación a objetos se adapta bien a la construcción de software de modelado y simulación. La definición de una correcta jerarquía de clases permite obtener software más fácilmente ampliable y reutilizable [77] . En el ámbito de la reutilización, cobra importancia la forma en que se construye la interfaz visual de usuario. La separación de la lógica interna y de la presentación puede ofrecer,

conjuntamente con una jerarquización de clases de granularidad fina, ventajas a la hora de crear posteriormente bibliotecas de clases reutilizables basadas en software de simulación o modelado. Esta separación también permite ofrecer, de forma sencilla, distintas representaciones de los mismos objetos en un entorno de simulación.

Los entornos de modelado y simulación son elementos susceptibles de ser ampliados, ya sea permitiendo la inclusión de nuevas funciones en una biblioteca [85] , o de bloques constitutivos completos [74] . La utilización de técnicas de desarrollo orientado a objetos, creando clases separadas para el manejo de bibliotecas de funciones o algoritmos, y la definición de interfaces abiertas que permitan añadir de forma directa, sin necesidad de recompilación, este tipo de elementos, aun suponiendo un esfuerzo inicial de desarrollo mayor, se ve compensado por las ventajas que aporta de extensibilidad sin necesidad de mantenimiento.

La integración de entornos aporta múltiples ventajas, ya que se reutiliza software disponible, con un esfuerzo mucho menor que el de la creación de nuevo software, y permite mantener entornos individuales que pueden ser más especializados. La utilización de la programación orientada a objetos puede facilitar esta integración, al permitir exportar clases con funcionalidad completa de un entorno a otro.

Actualmente se hace necesario un cambio en la manera de producción del courseware. Pasar desde las estructuras monolíticas e inflexibles hacia las bibliotecas de objetos de aprendizaje modulares y reutilizables, creadas como resultado de aplicar una estrategia con enfoque más ingenieril, que permita diseñar objetos que puedan ser fácilmente reutilizados o repropositados y que al combinarse permitan generar estructuras complejas, adecuadas a las necesidades específicas de entrenamiento identificadas.

3.2 Objetos de aprendizaje

Los objetos de aprendizaje son un nuevo tipo de estructuras de información basada en computadoras que tienen por finalidad apoyar el proceso de enseñanza-aprendizaje. Son pequeños componentes informativos que pueden ser reutilizados y ensamblados en diferentes contextos [111] , los cuales se conciben de forma análoga a la noción de objeto utilizada en la ingeniería de software [20] [86] [98] . En general se considera a los objetos de aprendizaje

como a entidades digitales distribuidas a través de Internet. La idea de crear pequeños bloques reutilizables de material informativo, sobre la que se apoyan los objetos de aprendizaje, parte del proceso que siguen los profesores cuando dividen los contenidos programáticos en partes, de tal manera que cada bloque temático apoye los objetivos de formación deseados [82] . Para describir a este tipo de componentes, el *Learning Technology Standards Committee* (LTSC) del IEEE acordó que el término "objetos de aprendizaje" era el más adecuado [110] aunque en la literatura aparecen diversos adjetivos asociados al concepto "objeto". Así, hay "objetos instruccionales", "objetos educacionales", "objetos de conocimiento" "objetos de educación", "documentos pedagógicos", entre otros. Como puede apreciarse, independientemente de los nombres que se le han dado a estos componentes, la mayoría están asociados a la educación. Esta asociación se debe a que el contenido de los "objetos" promueve procesos cognitivos si se toman en consideración ciertas consideraciones pedagógicas en cuanto a su desarrollo y evolución [14]. Es decir, un "objeto de aprendizaje" es una forma precisa de describir el contenido de una materia que debe ser aprendida.

Los objetos de aprendizaje pueden considerarse como los ladrillos con los que una institución construye un programa de acuerdo con su propia arquitectura predilecta. Usando ladrillos estándar, cada institución se puede ahorrar el costo de hacer sus propios ladrillos y --lo que resulta igualmente importante-- dispondría de materia prima con un estándar común. Esto permite, siguiendo con la analogía, que una organización pequeña construya su oferta de cursos módulo a módulo, en vez de agregar aposentos a una casa o tomar toda la casa y fundirla con otra.

Si un curso se diseña como un conjunto de unidades (objetos) autosuficientes, independientes del contexto y que se corresponden con habilidades o conocimientos, se pueden reaprovechar estas mismas unidades en otros cursos. La reutilización incrementa el valor y la calidad didáctica, además de la rentabilidad. Otra de las propiedades de los Objetos de Aprendizaje es la posibilidad de asociarles etiquetas y descriptores (Metadatos) lo que permite tratamientos informáticos tales como clasificación, seguimiento, búsquedas, secuenciación.

La potencialidad de los objetos de aprendizaje ha dado lugar a que sean foco de atención tanto de empresas, sectores gubernamentales e instituciones educativas, por lo que se están convirtiendo en un factor importante para la instrucción asistida por computadora [43] . Entre sus propiedades más interesantes se encuentran las siguientes:

1. **Reutilización:** los componentes informativos pueden utilizarse las veces que se requiera, en diferentes contextos y de manera simultánea.

2. **Interoperabilidad**: flexibilidad para utilizar los componentes desarrollados en un lugar con herramientas o plataformas localizadas en otro lugar y con herramientas y plataformas diferentes.

3. **Durabilidad:** resistencia a los cambios, sin necesidad de rediseñar.

4. **Accesibilidad:** acceso a los componentes informativos desde cualquier lugar y distribuirlo a otros lugares.

5. **Colaboración:** quienes incorporan objetos de aprendizaje pueden colaborar y beneficiarse inmediatamente con nuevas versiones.

6**. Personalización**: los componentes se enfocan según un modelo de competencias más que un modelo de curso.

Los aspectos antes señalados muestran claramente las diferencias entre los objetos de aprendizaje y los recursos multimedia instruccionales que sólo existen en un lugar y en un tiempo específico. Además, los objetos de aprendizaje no sólo son herramientas de apoyo para *e-learning*, sino también para la educación y la formación presénciales. Las experiencias de las instituciones educativas que ya incorporan objetos de aprendizaje a su practica docente cotidiana ha dado muy buenos resultados. Resumiendo, los beneficios que se obtienen al utilizar los objetos de aprendizaje son múltiples:

1. Tener material de buena calidad, informativo y didáctico al mismo tiempo, diseñado por expertos.

2. Promover el trabajo colaborativo entre profesores e instituciones educativas, tanto a nivel nacional como internacional.

3. Contar con objetos de aprendizaje adecuados para competencias de cursos comunes y eliminar duplicidad en este tipo de trabajo.

4. Mejorar la eficiencia docente, dedicando a otras actividades del proceso enseñanza-aprendizaje, el tiempo de preparación y búsqueda de material informativo y didáctico.

5. Incrementar rapidez y eficiencia en la preparación y la actualización de nuevos cursos, ya que el sistema puede crearlos.

6. Usar el material las veces que sean necesarias y en el tiempo que se desee.

7. Presentar los contenidos programáticos en componentes digitales diseñados bajo criterios que promuevan el interés tanto en docentes como en aprehendientes.

3.2.1 Estándares

Obviamente, la falta de una adecuada interoperabilidad entre todos los implicados (proveedores de contenidos, fabricantes de soluciones, plataformas y utilidades, organizaciones, usuarios), ha generado una gran confusión en las organizaciones, frenando el natural desarrollo y consolidación del mercado del *e-learning*, por lo que se han redoblado los esfuerzos e iniciativas tendientes a estandarizar todos los aspectos de estos sistemas vinculados a su interoperabilidad: definición conceptual y alcance de sus componentes, normas y mecanismos de interconexión, formatos de intercambio y otros.

El reto de los estándares tiene que ver con el desarrollo de interfaces y lenguajes comunes mediante los cuales puedan comunicarse e interactuar todas las piezas que componen un sistema formativo con soporte digital.

Los estándares del *e-learning* son normas comunes, acuerdos consensuados, interfaces o protocolos, que el mercado va adoptando progresivamente. Se trata de una cuestión que viene despertando mucho interés entre los usuarios y proveedores de *e-learning*. Los estándares deben aplicarse en la educación virtual por el mismo motivo que se utiliza la corriente eléctrica de 110 o de 220V en las diferentes redes eléctricas a nivel mundial.

Las ventajas de tener un sistema de enseñanza y aprendizaje integrado y basado en estándares son muy importantes ya que permiten, por ejemplo:

- Dar una respuesta inmediata para que los alumnos e instructores puedan tomar las medidas necesarias.
- Tener una biblioteca completa de material de formación en un sólo depósito.
- Combinar contenido listo para usarse con contenido personalizado
- Proporcionar informes que permiten tener mejores medidas del uso y los resultados.
- Reducir al mínimo los costos asociados con la implementación de sistemas múltiples.
- Ofrecer soporte para una selección amplia de herramientas de autoría que producen datos que se pueden registrar.

Existen cuatro organizaciones que han creado estándares en la industria del *e-learning*:

- ADL (*Advanced Distributed Learning*, o Aprendizaje Distribuido Avanzado).
- AICC (*Aviation Industry CBT Committee*, o Comité de Adiestramiento Basado en Computadoras de la Industria de Aviación).
- IEEE (*Institute for Electrical and Electronic Engineers*, o Instituto de Ingenieros Eléctricos y Electrónicos).
- IMS (*Instructional Management Systems*).

ADL es una organización apoyada por el gobierno estadounidense que investiga y desarrolla especificaciones para motivar la adopción y el avance del *e-learning*. Su propósito es intentar favorecer que los materiales de aprendizaje se adapten a las necesidades de adiestramiento y que estén disponibles de manera general. El estándar propuesto por ADL, actualmente muy utilizado, se llama SCORM (*Shareable Content Object Reference Model*, o Modelo de Referencia de Objetos de Contenido Compartido). SCORM combina varias especificaciones en un documento que puede ser fácilmente implementado.

El Modelo de Referencia de Objetos de Contenido Compartido consiste en un conjunto de especificaciones que permiten desarrollar, empaquetar y distribuir materiales educativos de alta calidad en el lugar y momento necesarios. Los materiales se elaboran asegurándose del cumplimiento de cuatro principios:

- Reutilización.
- Accesibilidad.
- Interoperabilidad.
- Duración.

Las especificaciones de SCORM, distribuidas por ADL, detallan cómo deben de publicarse los contenidos y usarse los metadatos; también, incluyen las especificaciones para representar la estructura de los cursos por medio de XML y el uso de API (*Application Programmer Interface).*

Este modelo está basado en el trabajo de AICC, IMS IEE, ARIADNE y otros grupos para crear un único modelo unificado de referencia de especificaciones técnicas interrelacionadas y directrices para identificar los requisitos de alto nivel para los contenidos del aprendizaje basado en la Web. SCORM incluye aspectos que afectan a los LMS (*Learning Management*

Systems) y a los proveedores de herramientas de creación de contenidos, diseñadores instruccionales y desarrolladores de contenidos, proveedores de formación y otros.

Se puede decir que SCORM abarca tres componentes:

Empaquetamiento de contenidos; se refiere a la manera en que se guardan los contenidos de un curso, el modo en que están ligados entre sí y la forma en la que se entrega la información al usuario. Todos estos datos se concentran en un archivo llamado *Manifest.xml*

Ejecución de comunicaciones; detalla el entorno para manejar la información y consta de dos partes: las directivas de ejecución y los metadatos del estudiante.

Metadatos del curso; son de dos tipos: los que incluyen la información del curso en sí, y los que contienen el material del estudiante.

Actualmente se cuenta con la versión 1.3 de SCORM, la cual se puede obtener en la siguiente dirección http://www.adlnet.org/.

Asimismo, existe un curso en línea que, además, es un buen ejemplo del uso de las especificaciones de SCORM, en www.scorm.tamucc.edu/.

AICC es una de las organizaciones más antiguas en cuanto a la proposición de estándares de *e-Learning*. Desde 1988 plantearon lineamientos para el desarrollo de sistemas de entrenamiento para la industria de la aviación, que ha sido un área pionera en el entrenamiento basado en simuladores y en el uso de herramientas de entrenamiento mediado por tecnología. AICC plantea un entrenamiento efectivo en costos, eficiente y sostenible, y para esto publican un conjunto de recomendaciones que incluyen configuraciones de hardware y software. El estándar CMI (*Computer Managed Instruction*, o Instrucción Administrada por Computadora) plantea lineamientos para el desarrollo de contenido que podría comunicarse con la mayoría de los sistemas de administración del aprendizaje (*Learning Management Systems)*

El comité de estándares de tecnologías del aprendizaje (**LTSC**) de la organización IEEE también ha desarrollado recomendaciones y estándares técnicos que enfatizan las mejores prácticas, que pueden ser evaluados. El estándar de uso más extendido de esta organización es LOM (*Learning Object Metadata*, o Metadatos de Objetos de Aprendizaje), que define las características de los elementos y estructuras de aprendizaje. Los metadatos no son más que

etiquetas descriptivas usadas para catalogar materiales educativos con el fin de facilitar su localización y uso, debido a que incorporan los requisitos de los materiales y la descripción de la forma en que pueden ser implementados.

El consorcio IMS reúne a vendedores e implementadores que se centran sobre el desarrollo de especificaciones basadas en XML, y describen las características clave de cursos, lecciones, evaluaciones, aprendices y grupos. Los estándares más importantes de IMS son: *IMS Metadata*, que plantea recursos para el procesamiento de datos de aprendizaje; *IMS Content Packaging*, que describe la manera de empaquetar contenidos, e *IMS QTI*, que describe formas de intercambio de preguntas y evaluaciones (cuestionarios).

Se puede concluir que la esencia de los estándares se enfoca sobre las posibilidades de comunicación de contenidos, los metadatos y el empaquetamiento, y quienes estén trabajando en algún proyecto de implementación o adopción de soluciones de *e-Learning* deberán consultar los documentos fuente de estos estándares para familiarizarse con sus contenidos.

3.3 Metodología del software educativo O.O. para micromundos interactivos.

La propuesta que se desarrolla en este documento busca unir lo expuesto en los anteriores capítulos, metodología del software educativo con paradigma O.O, con miras a crear entornos de aprendizaje basados en micromundos interactivos. El gran reto es diseñar e implementar micromundos altamente interactivos que tomen muy en cuenta el potencial tecnológico y los recursos disponibles actualmente, sobre una sólida base educativa y comunicacional.

El enfoque base para la conceptualización y diseño de micromundos está desarrollado en el libro de Galvis [40] , y las adiciones propuestas provienen de mecanismos de Ingeniería de Software usados actualmente para el análisis y diseño de Materiales Educativos Computarizados (MEC). Para establecer la estructura genérica sobre la cual se puedan "montar" micromundos lúdicos se va a tener en cuenta el conjunto de elementos mencionados en Galvis [40] y se usa el enfoque O.O. para definir el modelo de datos. La notación usada en este modelaje es la de UML.

Siguiendo el ciclo de vida de un MEC, la siguiente descripción permite entender cada una de sus etapas, enriquecidas con el enfoque OO mencionado.

3.3.1 Análisis

El objetivo de esta etapa es determinar el contexto en el cual se va a crear la aplicación y derivar de allí los requisitos que deberá atender la solución interactiva, como complemento a otras soluciones basadas en uso de otros medios (personales, impresos, audio-visuales, experienciales), teniendo claro el rol de cada uno de los medios educativos seleccionados y la viabilidad de su utilización.

De acuerdo con Galvis [40] en esta etapa se establece como mínimo la siguiente información:

- *Características de la población objetivo*: edad (física y mental), sexo, características físicas, y mentales (si son relevantes), experiencias previas, expectativas, actitudes, aptitudes, intereses o motivadores por aprender.
- *Conducta de entrada y campo vital*: nivel escolar, desarrollo mental, físico o psicológico, entorno familiar y escolar, etc.
- *Problema o necesidad a atender*. Para establecer la necesidad se puede recurrir a los mecanismos de análisis de necesidades educativas presentados en [40] , Cap. 5. Estos mecanismos usan entrevistas, análisis de resultados académicos, etc. para detectar los problemas o posibles necesidades que deben ser atendidas. El problema o necesidad no tiene que estar obligatoriamente relacionado con el sistema educativo formal, pueden ser necesidades sentidas, económicas, sociales, normativas, etc.
- Una vez identificado el problema, se deben establecer las bases para resolverlo. *Principios pedagógicos y didácticos aplicables* [40] , Cap. 4. En esta fase se debe analizar cómo se ha llevado a cabo el proceso de enseñanza-aprendizaje para establecer cómo debe enfocarse el entorno, qué factores se deben tomar en cuenta, qué objetivos debe cumplir.
- *Justificación de uso de los medios interactivos* como alternativa de solución. Para cada problema o necesidad identificada se debe plantear una estrategia de solución contemplando diferentes posibilidades. El apoyo informático debe ser tomado en cuenta siempre y cuando no exista un mecanismo mejor para resolver el problema:

soluciones administrativas, ver si el problema se soluciona al tomar decisiones de tipo administrativo; soluciones académicas, cambios en metodologías de clase; mejoras a los medios y materiales de enseñanza contemplando el uso de medios informáticos. Una vez que se han analizado todas las alternativas, se puede decir por qué el uso de medios informáticos es una buena solución. La justificación se puede basar en la no existencia de otro medio mejor y en la relación costo-beneficio para la institución pues puede ser que exista una mejor solución, pero que demande mayor tiempo y esfuerzo o un mayor costo económico, etc.

3.3.2 Especificación de requisitos

Como síntesis de la etapa de análisis se deben formular los requisitos que deberá atender el material interactivo que se desea obtener.

La especificación de requisitos debe abarcar lo siguiente:

- *Descripción de la Aplicación*: Contiene las características particulares de la aplicación dentro de determinado dominio: área de contenido, restricciones etc. Se hace una descripción de lo que hará la aplicación.
- Además se deben dejar claras las restricciones que se aplican y una descripción de los posibles escenarios de interacción que tendrá el usuario.
- Las restricciones están relacionadas con aspectos tales como:
    1. Población Objetivo y sus características (información recopilada en la fase de análisis).
    2. Áreas de contenido y sus características.
    3. Principios pedagógicos aplicables
    4. Modos de uso de la aplicación: individual, grupal, con apoyo de instructor, etc.
    5. Conducta de entrada. Todo aquello con lo que el usuario cuenta antes de usar la aplicación: experiencias, conocimiento, habilidades, etc.
- Los escenarios de interacción corresponden a los momentos de interacción que tendrá el usuario en cada uno de los ambientes del mundo. Por ejemplo, el registro de datos al iniciar la aplicación, la selección de herramientas, etc.

- *Diagramas de Interacción*: Permiten ver secuencias de interacción entre el usuario y la aplicación, representando lo que se espera del diálogo y dando más detalle a la descripción textual de la aplicación. Los diagramas de interacción son un formalismo que permite revelar la secuencia de acciones entre diferentes partes de la aplicación involucrada al llevarse a cabo determinada actividad. Es importante ver la secuencia de acciones para cada escenario de interacción. Con base en estos diagramas se pueden ver cuáles pueden ser las necesidades de información en cada escenario de interacción y se puede ir pensando en cuáles pueden ser los algoritmos que serán usados.

Las operaciones que aparecen en el diagrama son requisitos de información que se comparten entre cada uno de los diferentes objetos. Con base en estas operaciones se puede especificar la secuencia para llevar a cabo la acción objetivo del diagrama. Se debe tener un diagrama por cada escenario de interacción de la aplicación.

3.3.3 Diseño

El diseño del Micromundo Interactivo se realiza a tres niveles diferentes: educativo, comunicacional y computacional. La metodología de Ingeniería de Software Educativo (ISE) original es fuerte en cuanto al diseño educativo y diseño comunicacional de MECs. En esta propuesta ISE-OO se van a tomar en cuenta estas fortalezas y se van a usar de manera que sean reflejadas en el diseño computacional de la aplicación y en la implementación de la misma.

Al diseñar el ambiente en el que se desarrollará la acción se deben definir claramente los elementos que se determinaron como necesarios en todo micromundo interactivo y aquellos deseables que convenga para el caso. La identificación de estos elementos en esta etapa permite una mayor vinculación con la etapa de desarrollo. Muchas de las decisiones importantes acerca del micromundo y su comportamiento se toman aquí.

Se va a realizar el diseño usando el enfoque O.O., formalizando muchos de los aspectos relacionados con la aplicación, definiendo desde esta etapa los objetos, su comportamiento, el propósito de la aplicación, las restricciones existentes y los escenarios de interacción.

Como complemento al diseño educativo de ISE, se plantea el uso de una metodología que acerque mucho más los resultados y formulaciones hechas en dicha etapa hacia la

implementación computacional de la aplicación. Con ello se garantiza un diseño computacional y posterior implementación con una alta calidad. Cualquier ajuste se puede hacer en la etapa de diseño, reduciendo costos innecesarios en la etapa de implementación.
En este trabajo se toma como base una propuesta planteada por Figueroa [34] , la cual sirve como soporte al diseño O.O. y posterior diseño de datos e implementación de la aplicación. Se utiliza UML para la notación del modelo. Se desea obtener una arquitectura genérica para micromundos interactivos, que pueda ser extendida para satisfacer necesidades de un problema en particular. Junto con la arquitectura se especifica la funcionalidad que el usuario tendrá sobre el modelo, para saber qué cosas puede hacer sobre él. A continuación se definen en detalle cada una de las etapas del diseño: educativo, comunicacional y computacional.

3.3.3.1 Diseño Educativo

Tomando como punto de partida la necesidad o problema a atender, así como la conducta de entrada y el campo vital de la población objeto, se debe establecer lo que hay que enseñar o reforzar para subsanar con apoyo del MEC las necesidades encontradas. Como resultado de la fase de diseño educativo se debe tener lo siguiente: contenido y su estructura; micromundo; sistema de motivación; sistema de evaluación. De acuerdo con Galvis [40] , Cap. 6, el diseño educativo debe resolver los siguientes interrogantes: ¿Qué aprender con el MEC? ¿En qué micromundo aprenderlo? ¿Cómo motivar y mantener motivados a los aprendices? ¿Cómo saber que el aprendizaje se está logrando?

¿Qué aprender con el MEC?

Debe cuidarse la manera como se presentan los contenidos en el MEC. Las relaciones de dependencia entre los diferentes temas deben tomarse en cuenta para no forzar el paso de un tema a otro y mantener coherencia a lo largo del material.
Se debe tener clara la diferencia entre lo que se sabe antes de usar el MEC y lo que se espera que se sepa al finalizar el trabajo con éste: Objetivos, contenidos y sus interrelaciones. Siguiendo la idea de Galvis [40] , Cáp. 6 se debe establecer esto en términos operacionales, determinando los contenidos a tratar y el objetivo terminal del MEC [40] , Cáp.13 y luego descomponiendo éste en objetivos específicos y secuenciándolos.

¿En qué ambiente o micromundo aprenderlo?

Un MEC se compone de varios ambientes o micromundos, cada uno relacionado con un objetivo en particular. Para cada micromundo se debe establecer: Argumento, Mundo, Escenarios, Retos, Personajes y Herramientas, Objetos. Siguiendo el modelo O.O., se deben definir las *clases* que identifican cada uno de estos elementos. Algunas de estas clases serán la base sobre la cual se puede extender el micromundo. Al realizar el modelaje del mundo se deben definir las relaciones existentes entre estas clases.

Al refinar la definición de los elementos y al establecer las relaciones se puede saber cuáles de estos elementos serán clases que formarán parte del modelo estático del mundo y cuáles son simplemente atributos complejos de alguna clase de dicho modelo.

Además se debe definir qué cosas puede hacer el usuario en el mundo. En términos de UML se refiere a los *casos de uso* en el mundo. Los casos de uso se identifican al establecer los requisitos de información que debe satisfacer la aplicación. Los casos de uso pueden extenderse de acuerdo con las necesidades del problema. Cada caso de uso se especifica meditante diagramas de interacción que permitan ver los objetos que están involucrados así como la secuencia de mensajes entre ellos.

¿Cómo motivar y mantener motivados a los usuarios?

Según Mockus Seymour Papert [75] una de las contribuciones principales de Piaget, más allá del concepto de estadios de desarrollo, es mostrar que la gente posee diferentes teorías acerca del mundo. De acuerdo con esto, los niños aprenden mejor cuando son alentados a apoyarse sobre su propia intuición y a emplear lo que ya saben para desarrollar nuevas ideas.

En esta etapa del proceso de diseño se definen las metáforas usadas, así como cada personaje que aparece, dejando claro cuál es el rol que el usuario juega., las herramientas de interacción que podrá usar y cuál es el reto que debe resolver.

En el caso de los micromundos interactivos resulta vital despertar la motivación intrínseca proponiendo ambientes o situaciones que sean interesantes. Se debe tratar de llegar a lo que Piaget llama “intento de asimilar experiencia en las estructuras existentes en su mente”, con mínimas necesidades de acomodarlas a las demandas de una realidad externa [76]

La especificación unida a los resultados del diseño educativo puede servir como información de base para la utilización de herramientas como las encontradas en Rational Rose para

elaborar el diseño O.O. de los datos del mundo de la aplicación. Hay que reflejar la motivación en el modelo. Esto se nota adicionando eventos al modelo así como estableciendo relaciones para que las clases del modelo reaccionen de acuerdo con todo lo que pudiera suceder en el modelo.

¿Cómo saber que el aprendizaje se está logrando?

Las situaciones de evaluación (retos, etc.) deben estar relacionadas con los contenidos. La relevancia y pertinencia de determinado reto o prueba se debe sustentar con base en los contenidos que se han presentado y con la manera como han sido tratados.

Manejo de retroinformación, refuerzo y niveles de logro

Dependen mucho del enfoque del micromundo, según sea para aprendizaje por descubrimiento (enfoque heurístico) o por transmisión (enfoque algorítmico). En el caso de ambientes heurísticos como es el caso de la mayoría de los micromundos interactivos, la retroinformación se traduce en mostrar en el micromundo el efecto de lo que hizo el usuario, independientemente de si es correcto o no, para que éste sea quien analice lo que ha pasado y tome decisiones al respecto.

3.3.3.2 Diseño Comunicacional

En esta fase del proceso de diseño se define la interfaz (zona de comunicación usuario-programa) de la aplicación. En este momento se debe complementar ese bosquejo definiendo formalmente los objetos que posee cada pantalla y cuáles elementos del mundo son usados/afectados. Se toma como base la descripción macro dada en especificación. Es importante conseguir que la interfaz sea: amigable, flexible y agradable de usar; también debe ser consistente, es decir, cuidando que los mensajes y la distribución en pantalla, el juego de colores, etc. sigan un mismo patrón. También es necesario que sea altamente interactiva, lo cual conlleva incluir mecanismos de comunicación entre el usuario y la aplicación.

Al definir la interfaz se debe tener en cuenta: ¿cuáles dispositivos de entrada-salida conviene poner a disposición del usuario para trabajar con el Micromundo?, ¿qué zonas de comunicación entre usuario y programa debe tener el Micromundo?, ¿cuáles son las

características de dichas zonas de comunicación?, ¿cómo verificar que la interfaz satisface los requisitos mínimos deseados? Para cada pantalla de la interfaz se deben definir las zonas de comunicación así como la distribución de las mismas. Para hacer esto se deben seguir indicaciones de diseño de interfaces. En Galvis [40] , Cap.7, se hace una revisión de aspectos a tomar en cuenta.

Al diseñar una interfaz también se deben tomar en cuenta restricciones tecnológicas, características de la población y aspectos psicológicos de la percepción [40] , Cap.7.

Así como se estableció un modelo para el mundo, se debe establecer un modelo para la interfaz que esté atento a todo lo que ocurre en el mundo pero que sea independiente de él.

El modelo computacional de la interfaz consta de:

- Definición formal de cada pantalla
- Objetivo
- Eventos del modelo del mundo que está en capacidad de detectar
- Diagrama de la pantalla, indicando cuáles objetos tiene y dónde están ubicados.
- Listado de las características tanto de la pantalla como de cada objeto (colores, tamaño de fuentes, resolución de imágenes, etc.)
- Enlaces con otros elementos de la interfaz. En caso de que algún objeto (p. ej. botones) permitan "navegar" a otras pantallas.
- Notas adicionales. En caso de que se requiera realizar operaciones especiales en la interfaz. Por ejemplo indicar si hay animación cuando se activa o desactiva la pantalla, si hay música de fondo, etc.
- Diagrama de flujo de información en la Interfaz. Indica la relación entre las diferentes pantalla de la interfaz. Con este diagrama se puede establecer cual es la secuencia que se seguirá en la aplicación.

3.3.3.3 Diseño Computacional

Al final de esta etapa se obtiene como resultado, claramente definidas, cada una de las diferentes clases de objetos, incluyendo sus atributos (indicando si serán públicos -visibles a todo el mundo- o privados), el conjunto de métodos y el invariante de cada clase que corresponde al conjunto de restricciones o de requisitos que debe siempre cumplir una determinada clase. Por ejemplo, se puede tener definida una clase "reloj" que tiene como

atributo un intervalo de tiempo. El invariante de esta clase puede ser tan sencillo como "el intervalo debe ser siempre mayor o igual a cero".

Durante las fases de diseño educativo y comunicacional se han definido los diferentes objetos tanto del mundo como de la interfaz. Esta información se refina en esta fase, adecuándola a las posibilidades de la herramienta de desarrollo que se vaya a utilizar. Algunas clases necesitarán extenderse para ser usadas en el modelo.

Además se puede dar el caso de agregar nuevas clases y relaciones para dar mayor funcionalidad al modelo, acorde con los requisitos propios de la aplicación. La herramienta de desarrollo puede ofrecer mecanismos que faciliten la implementación de la interfaz. En caso de no ser así, el modelo del mundo se extiende de tal manera que pueda comunicarse efectivamente con el modelo de interfaz que deberá ser desarrollado.

Unido al conjunto de clases, llamado también modelo estático del mundo, se debe ilustrar la lógica acerca de cómo se desarrollan cada una de las actividades en el modelo. Para ello se deben refinar los casos de uso (algunos de los cuales ya se han obtenido en fases anteriores, ilustrando para cada uno de ellos el proceso que se sigue). Para hacer esto se pueden usar diagramas de interacción que pueden ser de dos tipos: diagramas de secuencia (similares a los usados en la fase de especificación) o diagramas de colaboración. En estos diagramas ya se puede ver la secuencia de mensajes entre los diferentes objetos involucrados en cada caso de uso y se pueden modelar todas las alternativas que puedan presentarse en cada caso.

Esta información puede ayudar a redefinir el modelo antes de iniciar la fase de desarrollo. Además permite validar si el modelo es completo y permite satisfacer todos los requisitos de la aplicación.

La figura (Ver Anexo 6) muestra los casos de uso generales de una aplicación que atiende la funcionalidad de micromundos interactivos. Estos casos de uso corresponden a aquellos que son satisfechos en el modelo genérico del mundo (ver Jacobson *et al* [51]). Básicamente el usuario puede recorrer todos los escenarios del mundo y en cada uno de ellos resolver retos. Puede interactuar con personajes y así obtener pistas para resolver determinado reto. Además puede recoger objetos que encuentra a su paso e incluso usar herramientas para modificar el escenario.

La figura (Ver Anexo 7) muestra el modelo de clases de mundo para un micromundo interactivo. Este modelo puede considerarse como la base sobre la cual se pueden montar

todos los elementos presentes en la aplicación. Este modelo usa notación UML. En dicho modelo se tiene el mundo y su conjunto de ambientes. Cada ambiente o escenario incluye un conjunto de objetos, herramientas, retos y personajes. El usuario puede navegar por el mundo libremente, cambiando de escenarios, resolviendo retos e interactuando con personajes.

Hay que estar atento a cuanto sucede en el modelo del micromundo. Para esto deben extenderse todos los elementos del mundo para que reaccionen ante determinados eventos. Estos eventos deben modelarse especificando qué eventos genera cada elemento del modelo (mundo, escenario, etc.). Además se debe especificar para cada elemento del modelo ante cuáles eventos está en capacidad de reaccionar.

Para ello se puede definir una *clase Evento*, a partir de la cual se pueden establecer todos los eventos del sistema. Esta clase está relacionada con todos los elementos del modelo que precisen generar un tipo de evento que identifique acciones hechas por él.

El desarrollo de micromundos interactivos es una necesidad actual que debe ser atendida por los desarrolladores de software educativo. El avance tecnológico, unido a una cultura informática cada vez mayor a nivel de estudiantes y profesores, permite pensar en tener materiales educativos computarizados cada vez más sofisticados que exploten todo el potencial tecnológico en pro de apoyar efectivamente el proceso de enseñanza-aprendizaje.

La inclusión del modelo O.O. articulado al ciclo de ISE permite aprovechar todo el potencial de las metodologías de ISE y de la moderna IS-OO. Esto es importante a la hora de desarrollar software de calidad. Esta integración de enfoques facilita el mantenimiento computacional del mundo en el que se desarrolla la acción, así como la expansión de éste a medida que se requiera, garantizándose así integridad con cada cambio que se realice en el modelo del mundo.

El esquema de interacción entre la interfaz y el modelo del mundo permite trabajar en paralelo cada uno de ellos y permite realizar cambios sin afectar el proceso de desarrollo. Por otra parte, al trabajar con enfoque O.O. se facilita la reutilización de código así como la portabilidad del mismo en el caso de usar lenguajes de programación O.O. como JAVA.

La metodología que se propone permite montar un micromundo interactivo sobre una estructura genérica ya desarrollada, extendiéndola para satisfacer requisitos específicos.

Conclusiones parciales del capítulo.

La utilización de una metodología basada en el enfoque orientado a objetos para la producción de software educativo de modelación y simulación proporciona ventajas en relación a la reutilización, la interoperabilidad y la ampliabilidad de los elementos componentes del sistema, como resultado del aumento de granularidad y modularidad que son inherentes al enfoque orientado a objetos.

La elaboración de "objetos de aprendizaje", con el fin de establecer bibliotecas o repositorios de objetos estandarizados que estén disponibles para los desarrolladores de software educativo, resulta actualmente de vital importancia y puede redundar en un aumento de la eficiencia en tiempo y calidad. Esto no sería posible sin la existencia de los estándares para la estructura de los objetos, que son los encargados de regular y permitir el uso y ampliación de las funciones de dichos objetos.

Una combinación armoniosa del enfoque orientado a objetos, el UML y la herramienta Rational Rose resulta muy ventajosa para la producción de software educativo que soporte micromundos interactivos. Esto permite que el usuario pueda navegar por el micromundo libremente, cambiando de escenarios, resolviendo retos e interactuando con personajes. Se facilita además la mantenibilidad computacional y la expansión del micromundo.

Conclusiones

Conclusiones

Aunque se han propuesto diversas teorías psicológicas para intentar explicar el proceso del aprendizaje humano, hasta el presente ninguna puede reclamar una primacía absoluta en cuanto a servir de base para el desarrollo de software educativo. Esto puede atribuirse, entre otras razones, a que la efectividad de cada enfoque teórico depende, entre otros factores, del tipo de software que se desee desarrollar. Específicamente en este trabajo se enfatiza sobre la concepción constructivista, pues en nuestra opinión y la de otros autores, es la que más se adapta a la producción de software educativo de modelación y simulación.

El paradigma de la programación orientada a objetos proporciona considerables ventajas para la producción de software educativo, destacándose entre ellas la posibilidad de lograr la reutilización de objetos de aprendizaje. Esto trae aparejada un estilo más eficiente de trabajo, con un significativo ahorro de tiempo, dinero y esfuerzo.

Recomendaciones

Recomendaciones

Aplicar consecuentemente los avances actualmente disponibles en cuanto a metodologías de producción de software educativo orientadas a objetos.

Llevar a cabo una experiencia piloto para generar materiales demostrativos que permitan mostrar el valor práctico de la metodología y la herramienta propuesta.

Valorar la conveniencia de aplicar un enfoque orientado a componentes para el desarrollo del software educativo, superando las limitaciones propias del enfoque orientado a objetos.

Divulgar entre los especialistas de diferentes centros de nuestro país que desarrollan software educativo, tales como los vinculados a los Ministerios de Educación, de Educación Superior, de Salud Pública y de la Informática y Comunicaciones, los principios y métodos del enfoque orientado a objetos, del UML y de Rational Rose que han constituido el objeto de estudio fundamental en el presente trabajo.

Bibliografía

Bibliografía

[1] Alessi, S. M., Trollip S.R. (2000). *Multimedia for learning*. Third edition. Needham Heights, MA: Allyn & Bacon.

[2] Anderson, R. (1994). Representations and requirements: The value of ethnography in system design. In *Human-Computer Interaction*, 9:151-182.

[3] Andrews, D.H., Goodson L.A. (1980). A comparative analysis of models of instructional design. *Journal of Instructional Development* 3 (4) 2-16.

[4] Armstrong T., Loane R. (1994). Educational Software: A developer's perspective. *TechTrends*, 39:20-22.

[5] Ausubel, D. P. (1968). *Educational Psychology: A Cognitive View*. New York: Holt, Rinehart and Winston.

[6] Banan-Ritland, Brenda et al. (2000). Learning object system as constructivist learning environments: Related assumptions, theories, and applications. En The Instructional Use of Learning Objects, Association for Instructional Technology. Disponible en http://www.ait.net. Consultado: marzo 2004.

[7] Berners-Lee, T.J., Cailliau, R., Groff, J.-F., Pollermann, B. (1992). World-Wide Web: An Information Infrastructure for High-Energy Physics. *Proceedings of Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, La Londe-les-Maures, Francia, enero 1992. World Scientific, Singapore, (ed.) D Perret-Gallix.

[8] Betz, J.A. (1995). Computer games: increase learning in an interactive multidisciplinary environment. *Journal of Educational Technology Systems*, 24(2), 195-205.

[9] Biggerstaff T.J., Perlis A.J. (1989). *Software Reusability*. ACM Press, 1989.

[10] Boehm, B.W., Gray, T.E. et al. (1984). Prototyping versus specifying: A multi-project experiment. *IEEE Transactions on Software Engineering*, SE-10(3), 290-303. (Cap. 8).

[11] Boehm, B.W. (1988). A spiral model of software development and enhancement. *IEEE Computer*, 21(5), 61-72.

[12] Booch, G., J. Rumbaugh, I. Jacobson (1999). *El Lenguaje Unificado de Modelado*, Addison Wesley.

[13] Bork, A. (1986). *El Ordenador en la Enseñanza*. Barcelona: Edit. Gustavo Gili.

[14] Borland International (1995). Borland Delphi Users Manual.

[15] Bostock, S. (1998) Courseware Engineering - an overview of the courseware development process. Keele University, UK. Disponible en: http://www.keele.ac.uk/depts/cs/Stephen_Bostock/docs/atceng.htm

[16] Branch, R. C. (1994). Common Instructional practices employed by secondary school teachers. *Educational Technology*, March, pp. 25-34.

[17] Brehmer, B. y Dorner, D. (1993). Experiments with computer-simulated microworlds: Escaping both the narrow straits of the laboratory and the deep blue sea of the field study. *Computers in Human Behavior*, 9 (2/3), 171-184.

[18] Carroll, J.M. (Ed.) (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development.* New York: John Wiley & Sons.

[19] Celes W., Corson-Rikert J. (1997). An easy-to-use and dynamically extensible 3D graphics library. Paper presented at 10th Brazilian Symposium of Computer Graphic and Image Processing, Campos do Jordao, Brasil , 13-16.

[20] Coleman et al. (1994). *Object-Oriented Development. The Fusion Method*. Englewood Cliffs, NJ: Prentice Hall.

[21] Corbett F.D., Card, H.C. (1998). Java tools for research and education in artificial neural networks. *Proceedings of the 1998 11th Canadian Conference on Electrical and Computer Engineering*, CCECE (IEEE Comp. Soc.) 1: 417-420

[22] Chandy, K. Mani; Kinity, Joseph; Rifkin, Adam; Zimmerman, Daniel. (1998) Framework for structured distributed object computing. *Paralell Computing*, 24: 1901-1922.

[23] Davis, A. M. (1993). *Software requirements: Objects, functions, and states*. Englewood Cliffs, New Jersey: Prentice Hall.

[24] Davis A. (1996). Object-oriented approach to circuit simulation. Paper presented at 1996 IEEE 39th Midwest Symposium on Circuits & Systems, Ames, USA, 18-21 Agosto.

[25] de Jong, T., van Andel, J., Leiblum, M., Mirande, M. (1992). Computer assisted learning in higher education in the Netherlands, a review of findings. *Computers & Education*, 19:381-386.

[26] de Jong, T., van Jooligen W. (1996). Discovery Learning with Computer: Simulations of Conceptual Domains. *Proceedings of the First Simposium on Research and Development of Educational Software*. Convento dos Capuchos, Costa da Caparica, Portugal, 7-9 Octubre.

[27] Delphi 6.0 (2001). Prog. computacional. Borland Inc. http://www.borland.com

[28] Deri L. (1997) A Component-based Architecture for Open, Independently Extensible Distributed Systems. PhD. Thesis, University of Bern, Switzerland.

[29] Drasgow, F. y Olson-Buchanan, J.B. (Eds) (1999). *Innovations in computerized assessment.* Mahwah, NJ: Lawrence Erlbaum.

[30] Edmonds, G. S., Branch, R. C., y Mukherjee, P. (1994). A Conceptual Framework for Comparing Instructional Design Models. *Educational Research and Technology*, 42(2), pp. 55-72.

[31] Elorriaga, J. A., Fernández de Castro, I., Gutiérrez, J. (1995). Sistemas tutores inteligentes y aprendizaje automático. *Informática y Automática*, Vol. 28, No. 4, diciembre.

[32] Faria, A.J. (1998). Business simulation games: Current usage levels – An update. *Simulation & Gaming*, 29 (3), 295-308.

[33] Fernández de Castro, I., Díaz Harraza, A., Verdejo, F.(1993). Architectural and planning issues in intelligent tutoring systems. *Journal of Artificial Intelligence and Education*, Vol.4, No. 4.

[34] Figueroa, P. (1997). Metodología de desarrollo de software Orientado por Objetos. Universidad de los Andes, Bogotá, Colombia. Disponible en:
http://www.cs.ualberta.ca/~pfiguero/soo/metod/

[35] Fox G., Furmanski W. (1997). Java for parallel computing and as a general language for scientific and engineering simulation and modeling. *Concurrency Practice and Experience*, 9: 415-425.

[36] Freedman J.A., Skapura D.M. (1992). *Neural networks: algoritms, applications and programming techniques*. Addison-Wesley, New York. 400 pp.

[37] Fulcher J. (1998). Laboratory support for the teaching of neural networks. *International Journal of Electrical Engineering Education*, 35: 29-36.

[38] Gagné, R.M. (1985) *Conditions of Learning and Theory of Instruction*. Austin, Texas: Holt, Rinehart and Winston.

[39] Gagné, R.M., Briggs, L., Wager, W.(1992). *Principles of Instructional Design*, 4th edition. New York: Harcourt, Brace, Jovanovich.

[40] Galvis Panqueva, A. (1992) *Ingeniería de Software Educativo*. Ediciones Uniandes, Santafé de Bogotá, Colombia.

[41] Garlan D., Allen R., Ockerbloom J. (1995) Architectural Mismatch or, Why it's hard to build systems out of existing parts. In *Proceedings of the 17th International Conference on Software Engineering* (ICSE-17), April 1995.

[42] Guerrero Luis A. (2004) Análisis y diseño orientado a objetos. Universidad de Chile. . Disponible en www.dcc.uchile.cl/~luguerre/cc51h/ Consultado: mayo 2004.

[43] Gibbons, Andrew S. et al. (2000). The nature and origin of instructional objects. En The Instructional Use of Learning Objects, Association for Instructional Technology. Disponible en http://www.ait.net. Consultado: mayo 2004.

[44] Gibson, E. (1990). Objects- Born and Bread, *BYTE*, vol. 15, no. 10..

[45] Good, T. L., Brophy, J. E. (1990). *Educational psychology: A realistic approach*. (4th ed.).White Plains, NY: Longman.

[46] Goodyear, P., (1995). Infrastructure for courseware engineering. In: R.D. Tennyson, A.E. Barron (Eds). *Automating instructional design: computer-based development and delivery tools*. Berlin: Springer-Verlag, pp. 11-31.

[47] Gosling, J. (1996) *The Java language specifications*. Reading, MA: Addison-Wesley.

[48] Harding R.D and others (1996). A consortium approach to courseware design in mathematics. *Computers & Education*, 26: 171-178.

[49] Henderson-Seller, B. y J.M. Edwards (1990). The OO systems life cycle. *Comm. of ACM*, vol. 33, no. 9..

[50] Hernández, E., J. Hernández, C. Lizandra (2001). *C++ Estandar*. ITP Paraninfo..

[51] Jacobson, I., G. Booch, J. Rumbaugh (1998*). The Unified Software Development Process*. Reading, MA: Addison-Wesley Longman Inc.

[52] Jacobson, I., G. Booch, J.Rumbaugh (2000) *El Proceso Unificado de Desarrollo*. Addision Wesley.

[53] Johnson, R.E. (1988) Designs reusable classes, *JOOP*, vol. 1, no. 2.

[54] Jonassen, D.H., T.M.R. McAleese, (Sin fecha). A Manifesto for a constructivist approach to technology in higher education. Disponible en: http://led.gcal.ac.uk/clti/papers/TMPaper11.html. .

[55] Jonassen, D.H. (2000). *Computers as mindtools for schools: Engaging critical thinking*. Columbus, OH: Prentice Hall.

[56] Kotonya G., Sommerville, I. (1998). *Requirements engineering: processes and Techniques.* Chichester, UK: John Wiley and Sons. Caps. 5 y 6.

[57] Kunze M., Steffens J. (1996). Neural network objects. Paper presented at *5th International Workshop on Software Engineering, Neural Nets, Genetic Algorithms, Expert Systems, Symbolic Algebra and Automatic Calculations in Physics Research,* AIHENP'96, Lausanne, Suiza , 2-6.

[58] Laborde, J.M., Strasser, R. (1990). Cabri-Géomètre: a microworld of geometry for guided discovery learning, *ZDM*, 90:5, 171-177.

[59] Main, R.G., (1997). Integrating Motivation into the Design Process. *Educational Technology*, (33)12, 37-41.

[60] Martin P.(1997) Development of an object-oriented, discrete-event simulation language using Java. *Proceedings of the Asia-Pacific Software Engineering Conference and International Computer Science Conference,* APSEC and ICSC (IEEE Comp. Soc., USA), 123-130.

[61] McIlroy M.D. (1969) *Mass Produced Software Components. In Software Engineering*, P. Naur and B. Randell, editors,. NATO Science Committee, January 1969.

[62] Mehrotra P., Venkatesan R., Quaicoe J. E.(1997). Development of a flexible object-oriented artificial neural network simulator. *Proceedings of the 1997 Canadian Conference on Electrical and Computer Engineering,* CCECE'97 (IEEE Comp. Soc.), 1:318-321.

[63] Meijler T.D., Nierstrasz O. (1997) Beyond Objects: Components. In *Cooperative Information Systems:Current Trends and Directions* , M.P. Papazoglou, G. Schlageter (Eds), Academic Press, Nov.1997, pp 49-78.

[64] Merrill, M. D. (1991). Constructivism and instructional design. *Educational Technology*, May, 45-53.

[65] Meyer B. (1996) *Object oriented software construction*, 2º ed. Prentice-Hall.

[66] Microsoft Corporation (1997). Visual Basic Programming.

[67] Millington D., Stapleton J. (1995). Special Report: developing a RAD standard. *IEEE Software,* 12 (5), 54-56. Cap. 8.

[68] Neilson I., Thomas R. (1996). Designing educational software as a re-usable resource. *Journal of Computer Assisted Learning*, 12:114-126.

[69] Nielsen, J.(1990). *Hipertext & Hipermedia*. Academic Press.

[70] Nierstrasz O., Meijler T.D. (1995) Research directions in software composition. *ACM Computing Surveys*, 27(2):262–264.

[71] Nierstrasz O., Gibbs S., Tsichritzis D. (1992) Component-Oriented software Development. *Communications of the ACM*, 35:160-165.

[72] Norman, K. (1994). Navigating the educational space with HyperCourseware. *Hypermedia*, Vol. 6, enero.

[73] Object Management Group (1996). The Common Object Request Broker: Architecture and Specification, Julio.

[74] Odum H.T., Peterson N. (1996). Simulation and evaluation with energy systems blocks. *Ecological Modelling*, 93: 155-173.

[75] Papert, S. (1980). *Mindstorms*. New York, NY: Basic Books. Harper Colophon Books.

[76] Piaget, J. (1952). *The Origins of intelligence in children*. New York: International Universiy Press.

[77] Praehfofer, Herbert. (1998). Object Oriented, modular hierarchical simulation modelling: towards reuse. *Simulation Practice and Theory*, 4: 120-124.

[78] Pressman, R.S. (1997). *Software Engineering: A Practitioner's Approach*. Fourth Edition. New York: McGraw-Hill.

[79] Reed J.A., Afjeh A.(1997) Using Java to develop educational engineering software. Paper presented at ASEE Annual Conference, Milwaukee, WI, USA, 15-18 Junio.

[80] Reenskaug T. (1996) *Working with Objects: the OOram Software Engineering Method*. Manning Publications, 1996.

[81] Reigeluth, C.M. (1983). Instructional Design: What Is It and Why Is It? En Reigeluth, C.M. (Ed.). *Instructional-Design Theories and Models: An Overview of their Current Status*, p.7. Hillsdale, NJ: Lawrence Erlbaum Associates.

[82] Reigeluth, C. M. & Nelson, L. M. (1997). A new paradigm of ISD? En R. C. Branch & B. B. Minor (Eds.), *Educational media and technology yearbook* (Vol. 22, pp. 24-35). Englewood, CO: Libraries Unlimited.

[83] Rogerson, D. (1997). *Inside COM: Microsoft's Component Object Model*. Microsoft Press.

[84] Roselló E.G., Ayude J., García Pérez-Schofield B., Pérez Cota M. (2001) *Towards orthogonal concurrency principles in object-oriented systems design.* Journal of Object-Oriented Programming.

[85] Roselló E. G., Fernández R.B., Suárez E. F.(1998). Desarrollo de una herramienta software de modelado matemático. Paper presented at 3º Simposio de Investigación y Desarrollo de Software Educativo. Evora , Portugal, 3-5 Sep.

[86] Rumbaugh, J. et al. (1991). *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice Hall.

[87] Rumbaugh, J. (1994). *Getting Started: Using Use Cases to Capture Requirements*. Journal of Object-Oriented Programming 7(5), 8-23.

[88] Sabharwal C. L. (1998). Java, Java, Java. *IEEE Potentials*, 17: 33-37.

[89] Saettler, P. (1990). *The evolution of american educational technology* . Englewood, CO: Libraries Unlimited, Inc.

[90] Sametinger J. (1997) *Software Engineering with Reusable Components*, Springer, New York.

[91] Schawbe D., G. Rossi (1998). Object Oriented approach to web-based applications design. *Theory and Practice of Object Systems*, 4 : 207-225.

[92] Schneider J.G. (1999). Components, Scripts and Glue: A conceptual framework for software composition. PhD Dissertation, University of Bern, Switzerland.

[93] Schneider J.G., Nierstrasz O. (1999). Components, Scripts and Glue. En *Software Architectures. Advances and Applications*. L. Barroca, J.Hall, and P. Hall, (eds.) pp 13–25. Springer Verlag.

[94] Schneiderman, B. (1998). *Designing the user interface*. Third edition. Cap. 13. Reading, MA: Addison-Wesley.

[95] Schuman, L. (1996). Perspectives on instruction. Información disponible en: http://edweb.sdsu.edu/courses/edtec540/Perspectives/Perspectives.html

[96] Schwetman H. (1995). Object-oriented simulation modeling with C plus plus /CSIM17. Paper presented at 1995 Winter Simulation Conference, WSC'95, Arlington, USA.

[97] Senbetta, G. (1991). An inquiry of time and cost estimating for computer-based training courseware design and development as determined by modified delphi method. Ph.D. thesis disertation, Purdue University.

[98] Shlaer, S. & Mellor, S. (1988). *Object-Oriented Systems Analysis. Modeling the World in Data*. Yourdon Press Computing Series. Englewood Cliffs, NJ: Prentice Hall.

[99] Silvert W. (1993). Object-oriented ecosystem modelling. *Ecological Modelling* 68:91-118.

[100] Skinner, B. F.(1954). The science of learning and the art of teaching. *Harvard Educational Review*, 24(2), 86-97.

[101] Skinner, Thorndike, Watson. http://userwww.sfsu.edu/~rsauzier/Thorndike.html

[102] Sommerville, I., P.Sawyer (1997). *Requirements Engineering: A Good Practice Guide*. Wiley, 1997.

[103] Sommerville , I. (2002). *Software Engineering*. 6th Edition. Harlow, England: Pearson Education Ltd.

[104] Stapleton J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, UK: Addison Wesley Longman. Cap. 8.

[105] Sun MicroSystems (1999). Enterprise JavaBeans Specification. Agosto

[106] Szyperski, C. (1998). *Component Software - Beyond Object-Oriented Programming*. ACM Press/Addison-Wesley.

[107] Van Milgen J., Boston R., Kohn R., Gerguson J. (1996). Comparison of available software for dynamic modelling. *Annales de Zootechnie* (Paris), 45: 257-273.

[108] Vaquero, A. (1998). La Tecnología en la Educación. TIC para la enseñanza, la formación y el aprendizaje. *Actas del IV Congreso Iberoamericano de Informática Educativa*. Brasilia, Brasil.Octubre.

[109] Visual Basic 6.0 (2000). Prog. computacional. http://www.microsoft.com.

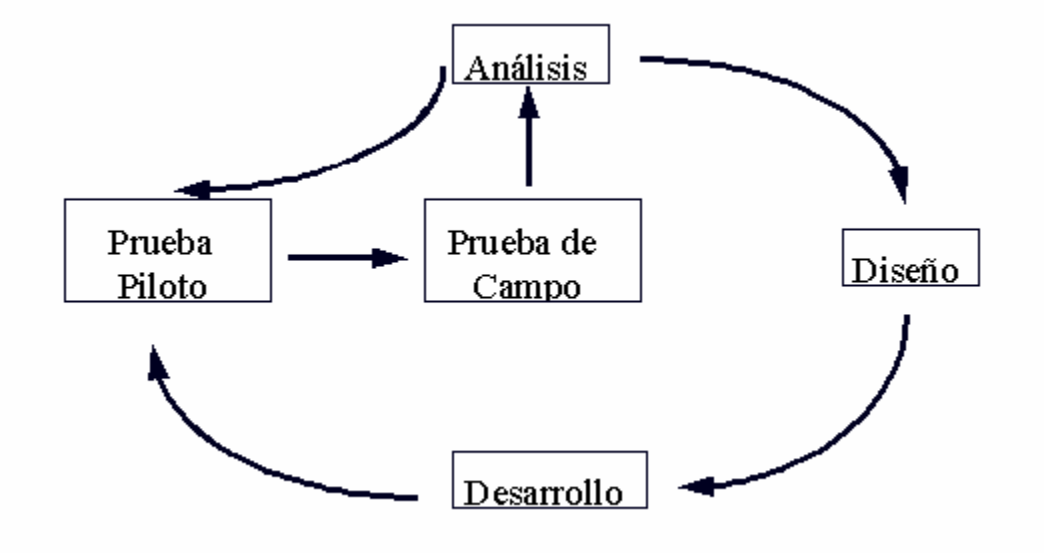
[110] Wiley, D. (2000). Learning object design and sequencing theory. A dissertation submitted to the faculty of Brigham Young University. Departament of Instructional Psychology and Technology. Disponible electrónicamente en http://davidwiley.com/papers/dissertation/dissertation.pdf.

[111] Wiley, D. (2001). The Instructional Use of Learning Objects, Association for Instructional Technology. Disponible http://www.ait.net. Consultado: mayo 2004.

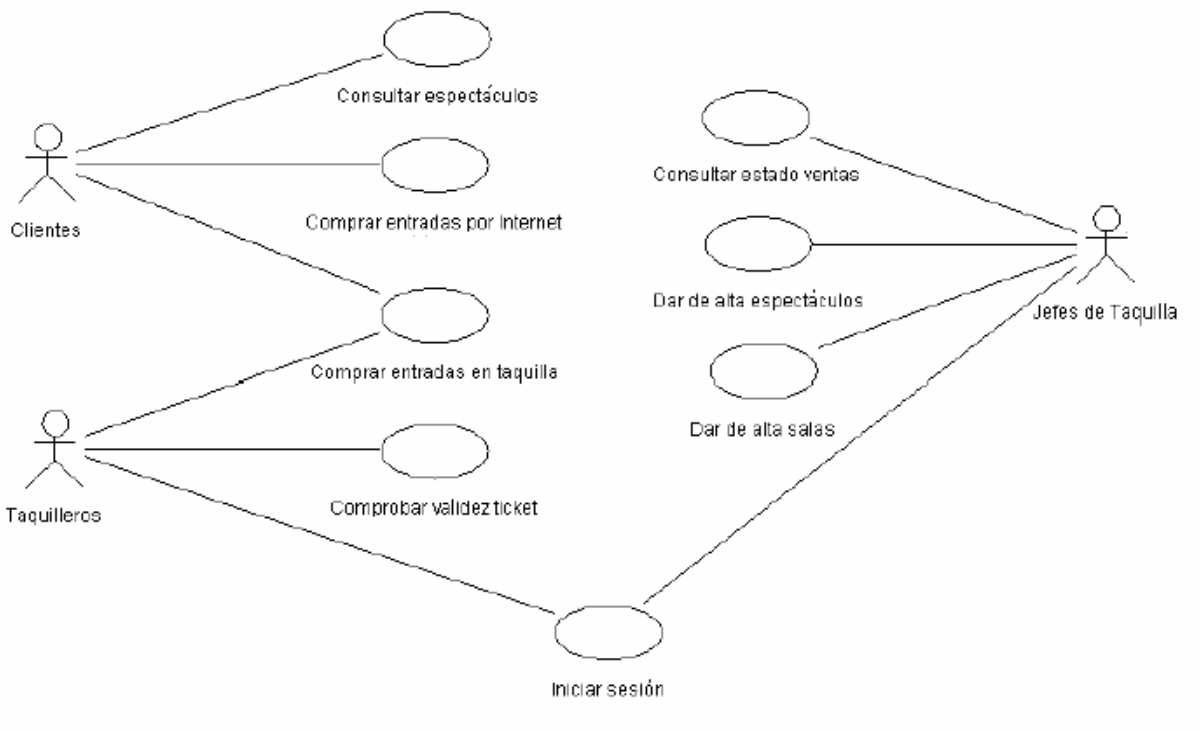
[112] Wong S. (1994). Quick prototyping of educational software: an object-oriented approach. *Journal of educational technology systems*, 22: 155-172.

Anexos

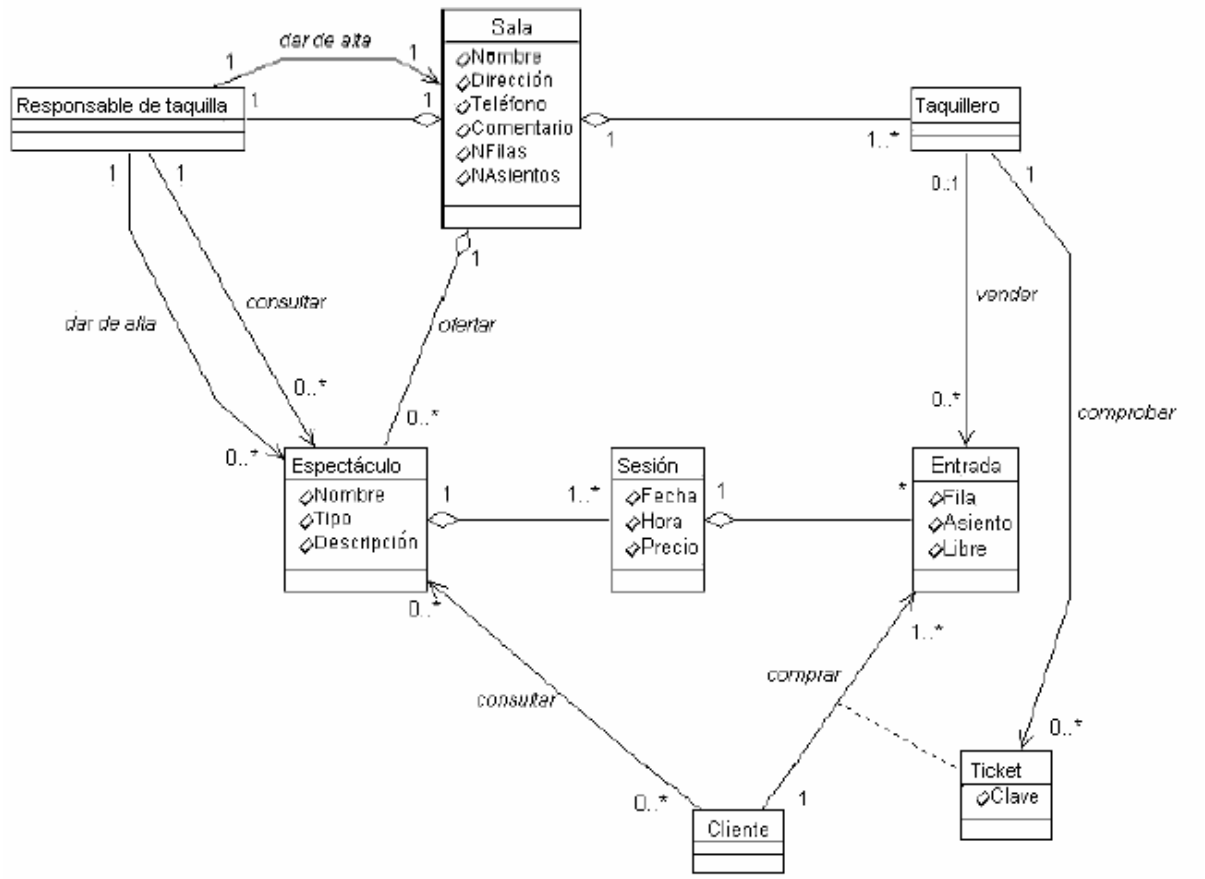
Anexo 1. Metodología ISE propuesta por Galvis [40] .



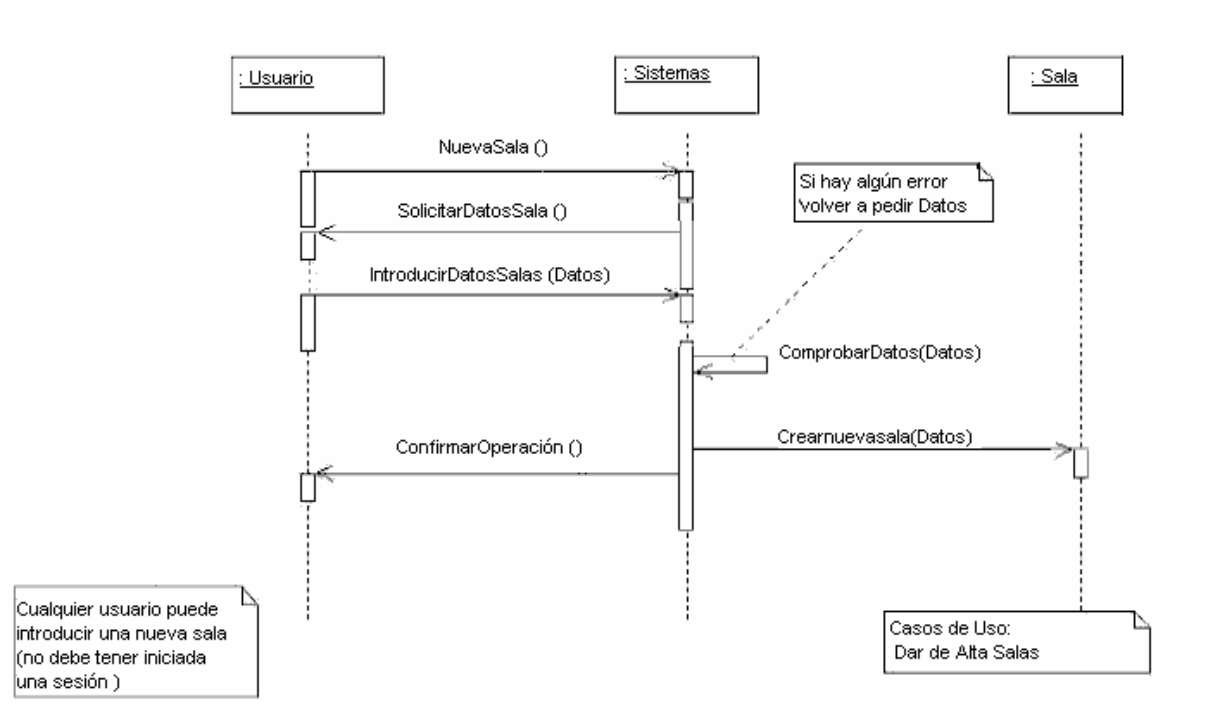
Anexo 2. Diagrama de casos de uso.



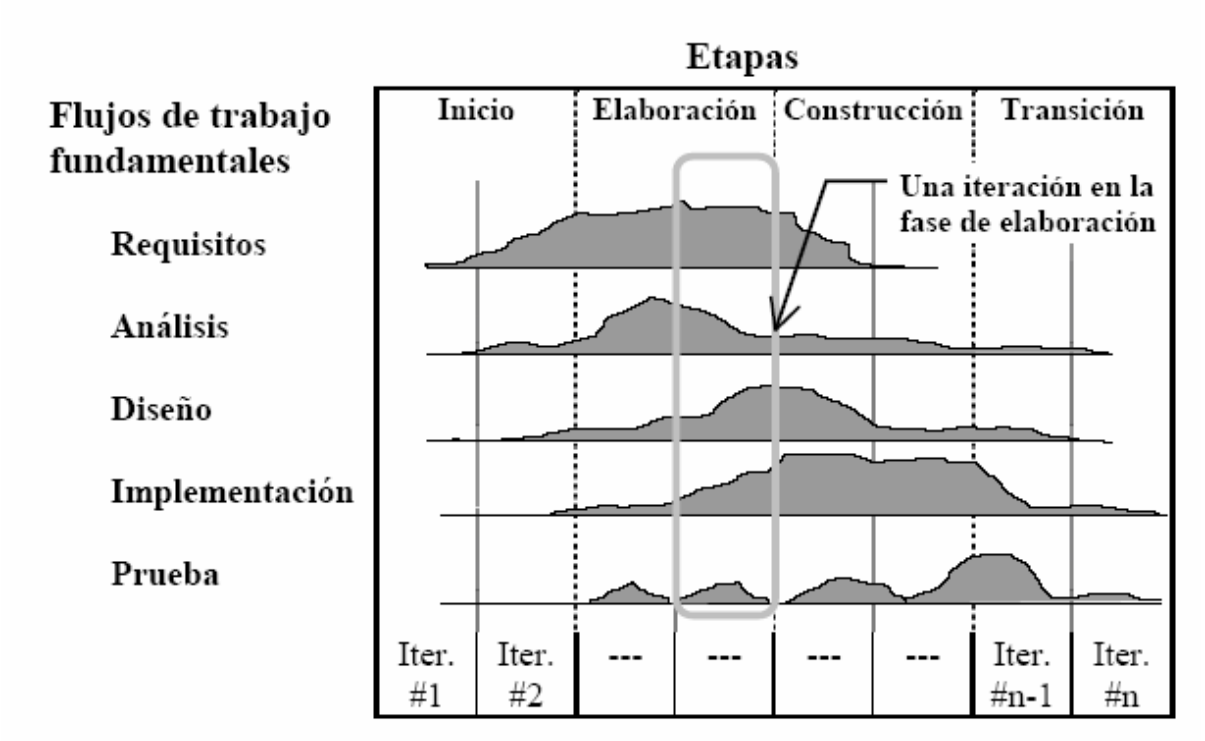
Anexo 3. Diagrama de clases.



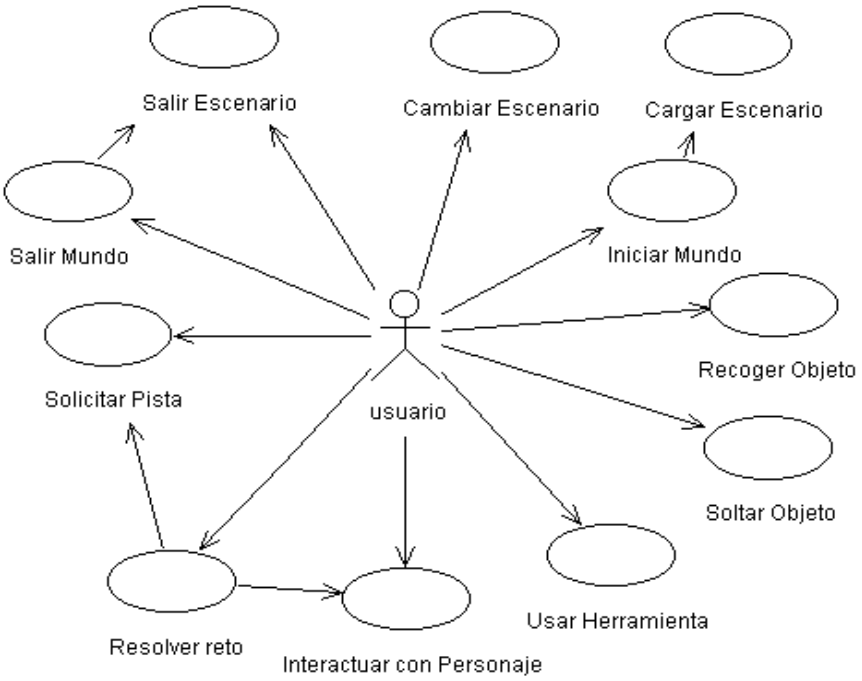
Anexo 4. Diagrama de secuencia.



Anexo 5. Proceso iterativo e incremental.



Anexo 6. Diagrama de casos de uso para un micromundo interactivo.



Anexo 7. Modelo UML del Mundo, para un micromundo interactivo.