Tema 5: Técnicas de Especificación y Modelado. Introducción a UML.

BLOQUE II:
ANÁLISIS Y DISEÑO DE LOS SISTEMAS SOFTWARE

Segundo curso de Grado en Ingeniería Informática
Curso 2014-2015

Javier Sánchez Monedero
jsanchezm@uco.es
http://www.uco.es/users/i02samoj

Índice

Índice

1.

2. Introducción

Introducción

Advertencia

Advertencia
Estos apuntes no contienen todo el material necesario para trabajar con UML. Intentan dar una visión de conjunto y proporcionar los conocimientos básicos para que podáis consultar la bibliografía y manuales de referencia, así como material complementario colgado en Moodle.

Objetivos

- Ampliar y formalizar **técnicas de especificación** que hemos visto en el Tema 4.
- Conocer UML (*Unified Modeling Language*) como lenguaje de **especificación** y **modelado** de sistemas software.

El lenguaje unificado de modelado

- El **lenguaje unificado de modelado** o **UML** (*Unified Modeling Language*) es un lenguaje de propósito general estandarizado pensado para visualizar el parte de la especificación y diseño de un sistema software [OMG, 2014a].
- Definimos ***modelado*** como la actividad de diseñar las aplicaciones software antes de la programación.
- UML está estandarizado por la organización ***Object Management Group (OMG)***[1].

[1] http://www.omg.org

¿Por qué usar UML?

En palabras literales de la OMG:

> "Using a model, those responsible for a software development project's success can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, ***before*** implementation in code renders changes difficult and expensive to make." [OMG, 2014a]

¿Qué nos permite?

UML **ayuda** especificar, visualizar y documentar modelos de sistemas software, incluyendo su estructura y diseño, de forma que estas cumplan todos los requisitos. UML permite modelar:

- Cualquier **actividad** (trabajos).
- **Componentes** individuales del sistema (y cómo interactúan con otros componentes).
- **Cómo se ejecutará** el sistema.
- Cómo **interaccionarán** las entidades con otras entidades (componentes o interfaces).
- **Interfaces** de usuario externas.

Figura 1: Fuente @Pontifex_es

Atención: ¡No hay que ser más papista que el Papa! La OMG utiliza con precisión la palabra *ayuda* a especificar y documentar. Los modelos visuales tendrán que ir acompañados de las aclaraciones textuales, con citas de código y demás recursos que hagan preciso y claro el documento.

El ejemplo más claro de esto lo podemos ver en los **Casos de Uso**. Los casos de uso se describen por completo textualmente, los diagramas de casos de uso nos ayudan a comprenderlos mejor proporcionando relaciones visuales entre ellos y construyendo una visión general de sistemas o subsistemas.

- También nos puede ayudar a **examinar un sistema existente** generando modelos a partir del código fuente (*técnicas de ingeniería inversa*).
- Existen varias **extensiones del estándar UML** para tipos de sistemas concretos (ver [OMG, 2014b]).
- Algunos autores consideran que puede utilizarse para programar, a través de extensiones como xtUML o utilizando metodologías como MDA (arquitectura dirigida por modelos o *Model-Driven, Architecture*).
- UML sirve para documentar **otros tipos de conocimiento** no software.

Objetivos de UML

Objetivos:

- Modelar sistemas, **desde los requisitos hasta los artefactos ejecutables**, utilizando técnicas de **orientación a objetos** (OO).
- Cubrir las cuestiones relacionadas con el tamaño propias de los sistemas complejos y críticos.
- Lenguaje utilizable por las personas y las máquinas.
- Encontrar equilibrio entre expresividad y simplicidad.
- UML es una notación, *no es un proceso* o metodología de desarrollo.
- Muchos procesos se basan en UML en mayor (RUP, UP, OpenUP) o menor medida (algunas metodologías ágiles y otros [Ambler, 2006]).
- La principal utilidad es permitir la comunicación.

Nota sobre orientación a objetos

- Es un conocimiento relativamente *extendido y erróneo* que *sólo se puede implementar la orientación a objetos con lenguajes que expresamente están diseñados bajo este paradigma*.
- Sin embargo hay técnicas para implementar orientación a objetos (¡incluyendo polimorfismo en tiempo de ejecución!) con lenguajes como C.

Ejemplos de libros e implementaciones que usamos a diario:

- *Object-Oriented Programming With ANSI-C* [Schreiner, 2011]
- GObject[2]
- GTK+
- ...

[2] http://en.wikipedia.org/wiki/GObject

¿Cómo empezar con UML según la OMG?

Según la OMG, para afrontar un proceso utilizando UML debemos (sin que el orden sea extricto) [OMG, 2014a]:

1. **Elegir una metodología** que se ajuste lo mejor posible al ámbito de nuestro sistema.
2. **Elegir una herramienta de desarrollo UML**, que puede estar condicionada por la metodología.
3. **Formase en UML**.

Por cuestión de imparcialidad, la OMG no recomienda ninguna herramienta o metodología concreta, se limita a ofrecer listados de éstas.

Historia y proceso de estandarización

En la década de los 90 existían muchos métodos de análisis y diseño, con objetivos y notaciones similares, pero no estandarizados, lo que llevaba a situaciones de confusión.

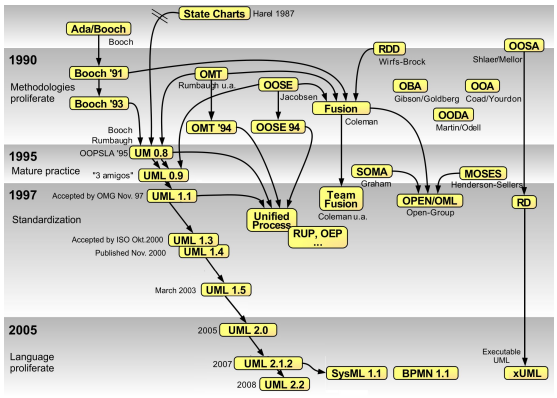

Figura 2: Historia del modelado de lenguajes orientados a objeto [Wikipedia, 2014b].

Historia y versiones [Wikipedia, 2014b]:

- En 1994 **Grady Booch**, **Ivar Jacobson** y **James Rumbaugh** desde la empresa Rational Software propusieron la primera versión.
- En 1997 la **OMG** adoptó UML como estándar y desde entonces lo gestiona.
- En 2000 UML fue aceptado por la organización internacional de estandarización (**ISO**, *International Organization for Standardization*) como *estándar ISO*.

- En 2005 apareció la versión actual, **UML 2.0**, cuya revisión más reciente es la versión 2.5, de 2012. Por ejemplo, se incluye el lenguaje descripción de restricciones OCL (*Object Constraint Language*).

Modelado

> "Un modelo es una simplificación de la realidad. [...]. Construimos modelos para comprender mejor el sistema que estamos modelando. [...] Construimos modelos de sistemas complejos porque no podemos comprender el sistema en su totalidad" [Booch et al., 2006] (Capítulo 1)

Modelado

A través del modelado conseguimos [Booch et al., 2006]:

1. Visualizar cómo es y cómo queremos que sea un sistema.
2. Especificar la estructura o comportamiento del sistema.
3. Proporcionar plantillas que guíen en la construcción.
4. Documentar las decisiones que hemos tomado.

Principios del modelado

Primero
La elección acerca de qué modelos crear tienen una profunda influencia sobre cómo se acomete un problema y cómo se da forma a una solución.

Segundo
Todo modelo puede ser expresado con diferentes niveles de precisión.

Tercero
Los mejores modelos están ligados a la realidad.

En el análisis estructurado el punto débil es el hecho de la desconexión básica entre el modelo de análisis y el modelo de diseño del sistema. En sistemas orientados a objetos es posible conectar todas las vistas casi independientes de un sistema en un todo semántico.

Cuarto
Un único modelo o vista no es suficiente. Cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes con múltiples puntos de vista.

3. Adopción y Críticas

Adopción y Críticas

Adopción

Ventajas y popularización

- UML se ha utilizado con éxito en multitud de contextos [Wikipedia, 2014c].
- Algunos autores no sólo valoran el proceso de estandarización por parte de la OMG, sino que hacen hincapié en la labor de *marketing* realizada por el consorcio que haya hecho que se haya popularizado tanto [Ambler, 2014].

Críticas a UML

Peligros en su uso
Algunas desventajas vienen más de un abuso de UML que de UML en si mismo [Bell, 2004]:

- A menudo se ha considerado como una "bala de plata" de diseño.
- Uso excesivo tratando de diseñar cualquier pequeña porción del código del sistema (lo cuaĺ es innecesario) y asumir que cualquier cosa se puede modelar con UML.
- Algunos críticos, incluidos Grady Booch o Ivar Jacobson, aseguran que hay un exceso de artefactos, lo que dificulta su aprendizaje (veremos que algunos diagramas se solapan en objetivos).

> Alot of talk in the blogosphere lately about the userfulness of UML. The general consensus is that - "if it can't generate code, it isn't worth using" [3]

En general, aunque UML sea enormemente popular, hay otras tecnologías que para escenarios concretos, como sistemas de información web, permiten la generación de código automática y plenamente funcional a partir de especificaciones de los modelos junto con porciones de código que implementan la funcionalidad del problema concreto (por ejemplo *Django* para generar portales web en *Python*).

[3] http://geekswithblogs.net/appsguild/archive/2005/08/29/51543.aspx

Conclusiones sobre UML

Conclusiones

- UML es el lenguaje estándar para crear modelos visuales de software
- UML es una tecnología que hay que conocer y dominar porque será esencial tanto para desarrollar software como para entender software realizado por terceros. Como toda tecnología, es importante saber para qué se diseñó, y cuáles son sus puntos fuertes y débiles, para sacarle el mayor provecho.

4. Modelo conceptual de UML

Modelo conceptual de UML

Visión general de UML

UML es un lenguaje para:

- Visualizar
- Especificar
- Construir
- Documentar

los artefactos de un sistema con gran cantidad de software (ver [Booch et al., 2006], Cap. 2).

El modelo conceptual de UML

Para comprender UML, se necesita adquirir un modelo conceptual del lenguaje. Esto requiere aprender a utilizar tres elementos principales:

- **Bloques** básicos de construcción de UML:
  - **Elementos** de construcción OO:
    - Estructurales
    - Comportamiento
    - Agrupación
    - Notación
  - **Relaciones**: ligan los diferentes elementos entre sí
  - **Diagramas**: representación gráfica de un conjunto de elementos y sus relaciones entre sí
- **Reglas** que dictan cómo se pueden combinar esos bloques.
- Otros **mecanismos comunes** que se aplican a través de UML.

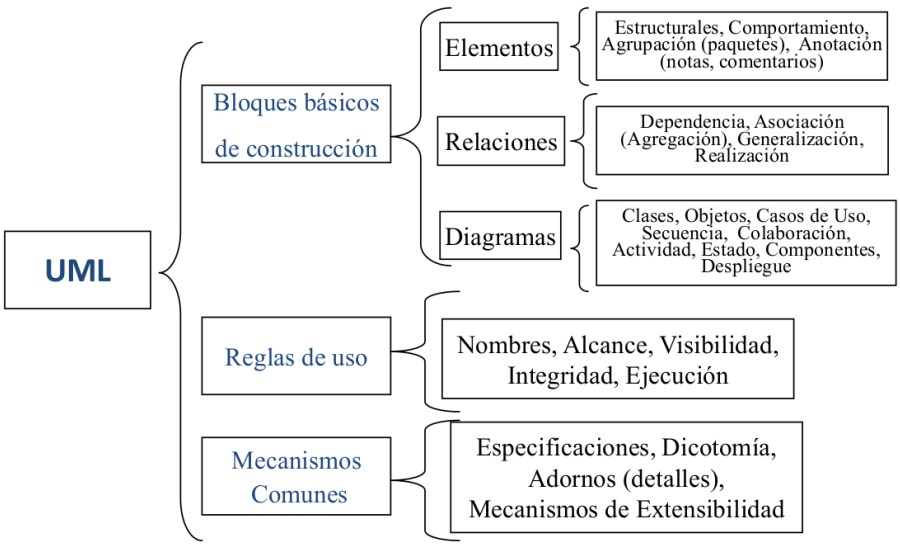

Figura 3: Esquema de conceptos de UML, fuente [Ruiz, 2013b]

5. Bloques UML

Bloques básicos UML

El vocabulario de UML incluye tres clases de bloques básicos:

1. Elementos.
2. Relaciones.
3. Diagramas.

5.1. Elementos UML

Elementos en UML

Hay cuatro tipos de elementos:

1. Elementos estructurales.
2. Elementos de comportamiento.
3. Elementos de agrupación.
4. Elementos de anotación.

Elementos estructurales

Elementos estructurales

Los **elementos estructurales**, también llamados *clasificadores*, son los nombres de los modelos UML.

- En su mayoría son las partes estáticas de un modelo.
- Representan conceptos o cosas materiales.
- Elementos que representan cosas conceptuales o lógicas: clase, interfaz, colaboración, caso de uso, clase activa y componente.
- Elementos que representan elementos físicos: artefactos y nodos.

Clases

Una *clase* es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Una clase implementa una o más interfaces. Gráficamente, una clase se representa como un rectángulo, que normalmente incluye su nombre, atributos y operaciones, como se muestra en la Figura 2.1.

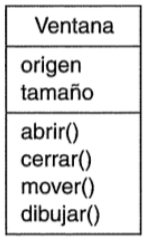

Figura 2.1: Clases.

Figura 4: Elemento estructural *clase*, fuente [Booch et al., 2006]

Interfaz I

Interfaz II

Colaboración

Caso de uso

Una *interfaz* es una colección de operaciones que especifican un servicio de una clase o componente. Por tanto, una interfaz describe el comportamiento visible externamente de ese elemento. Una interfaz puede representar el comportamiento completo de una clase o componente o sólo una parte de ese comportamiento. Una interfaz define un conjunto de especificaciones de operaciones (o sea, sus signaturas), pero nunca un conjunto de implementaciones de operaciones. La declaración de una interfaz se asemeja a la de una clase con la palabra «interface» sobre el nombre; los atributos no son relevantes, excepto algunas veces, para mostrar constantes. Una interfaz raramente se encuentra aislada. Cuando una clase proporciona una interfaz al mundo externo, ésta se representa como una pequeña circunferencia unida al rectángulo que representa a la clase por una línea. Una interfaz requerida por una clase, y que será proporcionada por otra clase, se representa como una pequeña semicircunferencia unida al rectángulo de la primera clase por una línea, como se muestra en la Figura 2.2.

Figura 5: Elemento estructural *interfaz*, fuente [Booch et al., 2006]

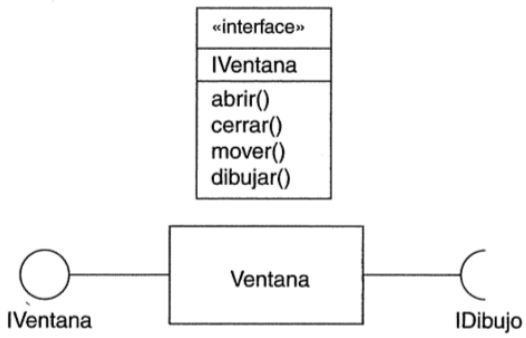

Figura 6: Elemento estructural *interfaz*, fuente [Booch et al., 2006]

Clase activa

Componente

Artefacto

Nodo

Elementos de comportamiento

Elementos de comportamiento

Una *colaboración* define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Las colaboraciones tienen tanto una dimensión estructural como una de comportamiento. Una clase dada o un objeto puede participar en varias colaboraciones. Estas colaboraciones representan la implementación de patrones que configuran un sistema. Gráficamente, una colaboración se representa como una elipse de borde discontinuo, que normalmente incluye sólo su nombre, como se muestra en la Figura 2.3.

Figura 2.3: Colaboraciones.

Figura 7: Elemento estructural *colaboración*, fuente [Booch et al., 2006]

Un *caso de uso* es una descripción de un conjunto de secuencias de acciones que ejecuta un sistema y que produce un resultado observable de interés para un actor particular. Un caso de uso se utiliza para estructurar los aspectos de comportamiento en un modelo. Un caso de uso es realizado por una colaboración. Gráficamente, un caso de uso se representa como una elipse de borde continuo, y normalmente incluye sólo su nombre, como se muestra en la Figura 2.4.

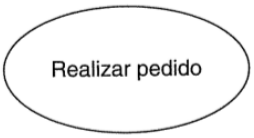

Figura 2.4: Casos de Uso.

Figura 8: Elemento estructural *caso de uso*, fuente [Booch et al., 2006]

Los **elementos de comportamiento** son las partes dinámicas de los modelos UML. Estos son los *verbos* de un modelo y representan el comportamiento en el tiempo y el espacio.

Semánticamente están conectados a elementos estructurales (normalmente a clases, colaboraciones y objetos).

Interacción

Máquina de estados

Una *clase activa* es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución y, por tanto, pueden dar origen a actividades de control. Una clase activa es igual que una clase, excepto en que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos. Gráficamente, una clase activa se representa como una clase, pero con líneas dobles a derecha e izquierda; normalmente incluye su nombre, atributos y operaciones, como se muestra en la Figura 2.5.

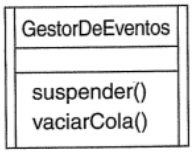

Figura 2.5: Clases Activas.

Figura 9: Elemento estructural *clase activa*, fuente [Booch et al., 2006]

Un *componente* es una parte modular del diseño del sistema que oculta su implementación tras un conjunto de interfaces externas. En un sistema, los componentes que comparten las mismas interfaces pueden sustituirse siempre y cuando conserven el mismo comportamiento lógico. La implementación de un componente puede expresarse conectando partes y conectores; las partes pueden incluir componentes más pequeños. Gráficamente, un componente se representa como una clase con un icono especial en la esquina superior derecha, como se muestra en la Figura 2.6.

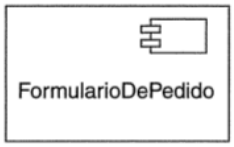

Figura 10: Elemento estructural *componente*, fuente [Booch et al., 2006]

Actividad

Elementos de agrupación y de anotación

Elementos de agrupación

Los **elementos de agrupación** son las partes organizativas de los modelos UML. Los **paquetes** son el tipo principal de elementos de agrupación. Dentro de UML hay varios tipos de paquetes

Un *artefacto* es una parte física y reemplazable de un sistema que contiene información física ("bits"). En un sistema hay diferentes tipos de artefactos de despliegue, como archivos de código fuente, ejecutables y scripts. Un artefacto representa típicamente el empaquetamiento físico de código fuente o información en tiempo de ejecución. Gráficamente, un artefacto se representa como un rectángulo con la palabra clave «artifact» sobre el nombre, como se muestra en la Figura 2.7.

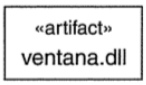

Figura 2.7: Artefactos.

Figura 11: Elemento estructural *artefacto*, fuente [Booch et al., 2006]

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que por lo general dispone de algo de memoria y, con frecuencia, capacidad de procesamiento. Un conjunto de artefactos puede residir en un nodo y puede también migrar de un nodo a otro. Gráficamente, un nodo se representa como un cubo, incluyendo normalmente sólo su nombre, como se muestra en la Figura 2.8.

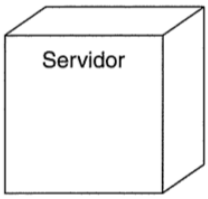

Figura 2.8: Nodos.

Figura 12: Elemento estructural *nodo*, fuente [Booch et al., 2006]

Paquete

Paquete

5.2. Relaciones en UML

Tipos de relaciones UML

1. Dependencia.

En primer lugar, una *interacción* es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular, para alcanzar un propósito específico. El comportamiento de una sociedad de objetos o una operación individual puede especificarse con una interacción. Una interacción involucra a muchos otros elementos, que incluye mensajes, acciones y enlaces (conexiones entre objetos). Gráficamente, un mensaje se muestra como una línea dirigida, incluyendo casi siempre el nombre de su operación, como se muestra en la Figura 2.9.

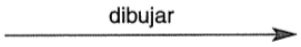

Figura 2.9: Mensajes.

Figura 13: Elemento de comportamiento *interaccion*, fuente [Booch et al., 2006]

En segundo lugar, una *máquina de estados* es un comportamiento que especifica las secuencias de estados por las que pasa un objeto o una interacción durante su vida en respuesta a eventos, junto con sus reacciones a estos eventos. El comportamiento de una clase individual o una colaboración de clases puede especificarse con una máquina de estados. Una máquina de estados involucra a otros elementos, incluyendo estados, transiciones (el flujo de un estado a otro), eventos (que disparan una transición) y actividades (la respuesta a una transición). Gráficamente, un estado se representa como un rectángulo de esquinas redondeadas, que incluye normalmente su nombre y sus subestados, si los tiene, como se muestra en la Figura 2.10.

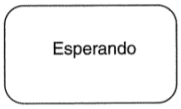

Figura 2.10: Estados.

Figura 14: Elemento de comportamiento *máquina de estados*, fuente [Booch et al., 2006]

2. Asociación.
3. Generalización.
4. Realización.

Dependencia

Asociación

Generalización

En tercer lugar, una *actividad* es un comportamiento que especifica la secuencia de pasos que ejecuta un proceso computacional. En una interacción, el énfasis se pone en el conjunto de objetos que interactúan. En una máquina de estados, el énfasis se pone en el ciclo de vida de un objeto cada vez. En una actividad, el énfasis se pone en los flujos entre los pasos, sin mirar qué objeto ejecuta cada paso. Un paso de una actividad se denomina una *acción*. Gráficamente, una acción se representa como un rectángulo con las esquinas redondeadas, con un nombre que indica su propósito. Los estados y las acciones se distinguen por sus diferentes contextos.

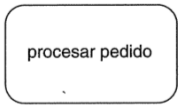

Figura 2.11. Acciones.

Figura 15: Elemento de comportamiento *actividad*, fuente [Booch et al., 2006]

Un *paquete* es un mecanismo de propósito general para organizar el propio diseño, en oposición a las clases, que organizan construcciones de implementación. Los elementos estructurales, los elementos de comportamiento e incluso otros elementos de agrupación pueden incluirse en un paquete. Al contrario que los componentes (que existen en tiempo de ejecución), un paquete es puramente conceptual (sólo existe en tiempo de desarrollo). Gráficamente, un paquete se visualiza como una carpeta, que normalmente sólo incluye su nombre y, a veces, su contenido, como se muestra en la Figura 2.12.

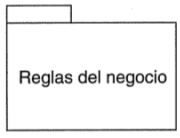

Figura 2.12. Paquetes.

Figura 16: Elemento de agrupación *paquete*, fuente [Booch et al., 2006]

Realización

5.3. Diagramas en UML

Tipos de diagramas

- En esta sección veremos por encima diagramas de ejemplo *sencillos*.
- La versión 2 de UML define **13 tipos de diagramas**.
- Vamos a explorar un ejemplo de cada tipo para tener una visión general y de utilidad de los diagramas.

Elementos de anotación. Los *elementos de anotación* son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo. Hay un tipo principal de elemento de anotación llamado nota. Una *nota* es simplemente un símbolo para mostrar restricciones y comentarios junto a un elemento o una colección de elementos. Gráficamente, una nota se representa como un rectángulo con una esquina doblada, junto con un comentario textual o gráfico, como se muestra en la Figura 2.13.

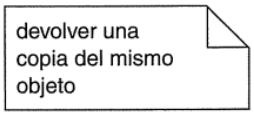

Figura 2.13: Notas.

Figura 17: Elemento de agrupación *nota*, fuente [Booch et al., 2006]

En primer lugar, una *dependencia* es una relación semántica entre dos elementos, en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (el elemento dependiente). Gráficamente, una dependencia se representa como una línea discontinua, posiblemente dirigida, que incluye a veces una etiqueta, como se muestra en la Figura 2.14.

Figura 2.14: Dependencias.

Figura 18: Relación de *dependencia*, fuente [Booch et al., 2006]

- Veremos como algunos se solapan hasta el punto de ser equivalentes (ejemplo los de secuencia y comunicación).

Diagrama de casos de uso

Diagrama de casos de uso (*use case diagram*): sirve para modelizar **cuáles son los casos de uso** del sistema y cuáles son los **actores** que están relacionados con ellos.

Diagrama de clases

Diagrama de clases (*class diagram*): Sirve para modelizar las clases de objetos y sus relaciones, haciendo hincapié en los aspectos estructurales más que en el comportamiento. Probablemente sea el más conocido.

En segundo lugar, una *asociación* es una relación estructural entre clases que describe un conjunto de enlaces, los cuales son conexiones entre objetos que son instancias de clases. La agregación es un tipo especial de asociación, que representa una relación estructural entre un todo y sus partes. Gráficamente, una asociación se representa como una línea continua, posiblemente dirigida, que a veces incluye una etiqueta, y a menudo incluye otros adornos, como la multiplicidad y los nombres de rol, como se muestra en la Figura 2.15.

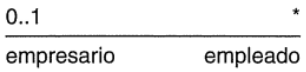

Figura 2.15: Asociaciones.

Figura 19: Relación de *asociación*, fuente [Booch et al., 2006]

En tercer lugar, una *generalización* es una relación de especialización/generalización en la cual el elemento especializado (el hijo) se basa en la especificación del elemento generalizado (el padre). El hijo comparte la estructura y el comportamiento del padre. Gráficamente, una relación de generalización se representa como una línea continua con una punta de flecha vacía apuntando al padre, como se muestra en la Figura 2.16.

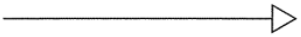

Figura 2.16: Generalizaciones.

Figura 20: Relación de *generalización*, fuente [Booch et al., 2006]

En cuarto lugar, una *realización* es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos sitios: entre interfaces y las clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan. Gráficamente, una relación de realización se representa como una mezcla entre una generalización y una relación de dependencia, como se muestra en la Figura 2.17.

Figura 2.17: Realizaciones.

Figura 21: Relación de *realización*, fuente [Booch et al., 2006]

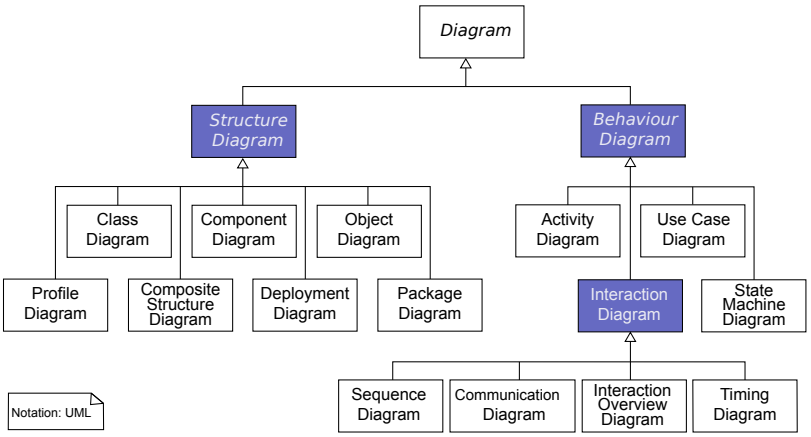

Figura 22: Jerarquía de diagramas UML, modificado de [Wikipedia, 2014b]

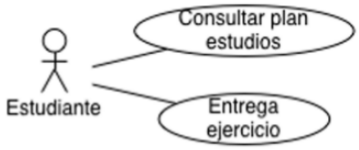

Figura 23: Ejemplo de diagrama de casos de uso (fuente [Miquel and Martos, 2010a])

Diagrama de objetos

Diagrama de objetos (*object diagram*): similar al de clases en el que lo que representamos no son clases, sino instancias de clase (objetos). Es novedad en UML 2.

Diagrama de paquetes

Un **paquete** es un grupo de elementos UML, como las clases o los casos de uso, o cualquier otra agrupación de elementos.

Diagrama de paquetes (*package diagram*): modelar paquetes y dependencias entre ellos.

Diagrama de comunicación

Diagrama de comunicación (*communication diagram*): sirve para modelar la interacción entre varios objetos: qué objetos están conectados y cómo colaboran entre ellos.

Diagrama de secuencia

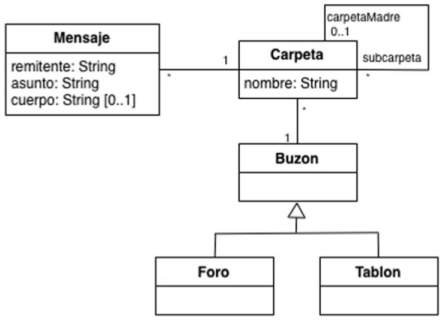

Figura 24: Ejemplo de diagrama de clases (fuente [Miquel and Martos, 2010a])

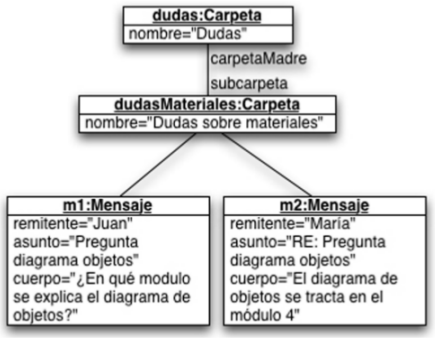

Figura 25: Ejemplo de diagrama de objetos (fuente [Miquel and Martos, 2010a])

Diagrama de secuencia (*sequence diagram*): modeliza lo mismo que el **diagrama de comunicación** pero resaltando la secuencia temporal.

Diagrama de actividades

Diagrama de actividades (*activity diagram*): tiene la función de describir algoritmos o procesos de forma similar a los **diagramas de flujo** (de hecho en Wikipedia hay una única entrada para los dos[4])

Diagrama global de interacciones

Diagrama global de interacciones (*interaction overview diagram*): el diagrama global de las interacciones es un *diagrama de interacción*. Muestra una cierta vista sobre los aspectos dinámicos de los sistemas modelados combinando los diagramas de actividades con secuencia o comunicación.

[4] http://es.wikipedia.org/wiki/Diagrama_de_flujo

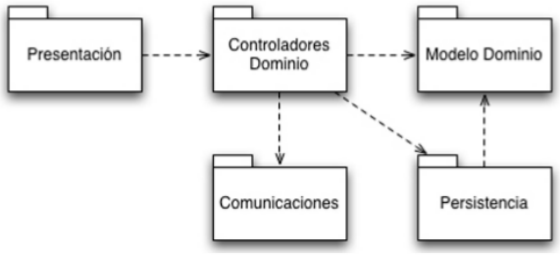

Figura 26: Ejemplo de diagrama de paquetes (fuente [Miquel and Martos, 2010a])

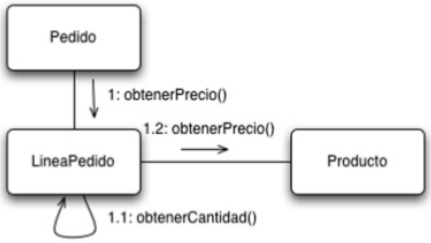

Figura 27: Ejemplo de diagrama de comunicación (fuente [Miquel and Martos, 2010a])

Diagrama de estados

Diagrama de estados (*state machine diagram*): describe estados y transiciones reflejando cómo afentan a uno o varios objetos los eventos del sistema. También es novedad su nombre en UML 2.

Diagramas de componentes

Diagrama de componentes (*component diagram*): describe la interconexión e interfaces entre los distintos componentes que a su vez forman otros componentes mayores o el propio sistema software.

Los componentes incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes.

Diagrama de estructura compuesta

Diagrama de estructura compuesta (*composite structure diagram*): modeliza la estructura interna en tiempo de ejecución de una clase o componente.

Diagrama de despliegue

Diagrama de despliegue (*deployment diagram*): modeliza la distribución física, en tiempo de ejecución, de los diferentes artefactos de software.

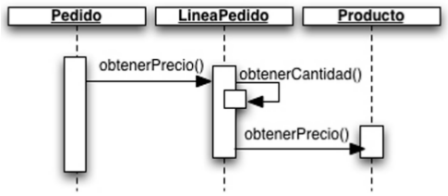

Figura 28: Ejemplo de diagrama de secuencia (fuente [Miquel and Martos, 2010a])

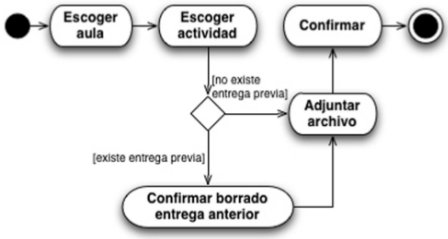

Figura 29: Ejemplo de diagrama de actividades (fuente [Miquel and Martos, 2010a])

Diagrama de tiempo/tiempos

Diagrama de tiempo/tiempos (*timing diagram*): está centrado en las restricciones temporales. Explorar el comportamiento de objetos durante un periodo de tiempo. Es una forma especial del diagrama de secuencia.

5.4. Reglas de UML

Reglas de UML

- Los bloques de construcción UML no pueden combinarse de cualquier manera.
- Un **modelo bien formado** es aquel que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados.

UML tiene reglas sintácticas y semánticas para:

- Nombres: cómo llamar a los elementos, relaciones y diagramas.
- Alcance: el contexto da un significado específico a un nombre.
- Visibilidad: cómo se pueden ver y utilizar los nombres.
- Integridad: relación apropiada y consistente entre elementos.

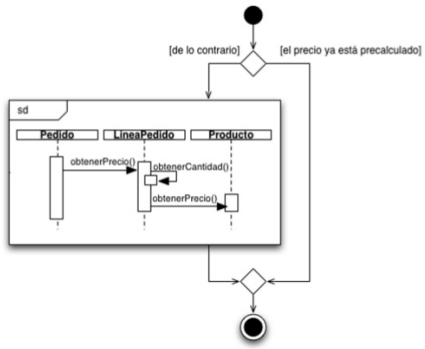

Figura 30: Ejemplo de diagrama global de interacciones (fuente [Miquel and Martos, 2010a])

Figura 31: Ejemplo de diagrama de estados (fuente [Miquel and Martos, 2010a])

- Ejecución : qué significa ejecutar o simular un modelo dinámico.

Según Booch et al. [2006], es habitual que el equipo de desarrollo no sólo construya modelos bien formados sino que los modelos sean:

- **Abreviados**: algunos elementos se ocultan para simplificar la vista (ver recomendaciones de James Heumann de IBM Rational Software para escribir buenos casos de uso [Heumann, 2008]).
- **Incompletos**: pueden estar ausentes ciertos elementos.
- **Inconsistentes**: no se garantiza la integridad del modelo.

Los autores afirman que estos modelos que no llegan a ser *bien formados* son inevitables según aparecen detalles del sistema y se avanza en el desarrollo. **Las reglas de UML favorecen (pero no obligan)** a considerar las cuestiones más importantes de análisis, diseño e implementación que permiten obtener sistemas bien formados con el paso del tiempo.

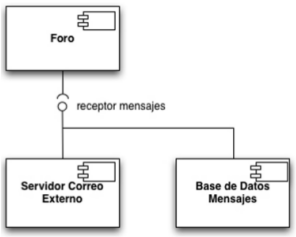

Figura 32: Ejemplo de diagrama de componentes (fuente [Miquel and Martos, 2010a])

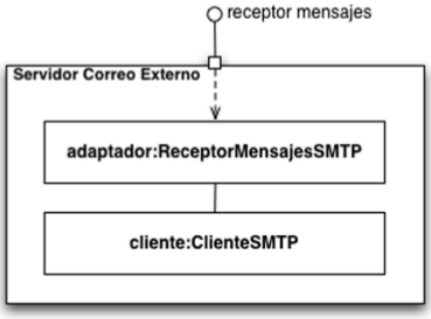

Figura 33: Ejemplo de diagrama de estructura compuesta (fuente [Miquel and Martos, 2010a])

5.5. Mecanismos comunes en UML

Mecanismos comunes en UML

UML tiene cuatro **mecanismos comunes** que se aplican de forma consistente a través de todo el lenguaje:

1. **Especificaciones**. Detrás de cada elemento con su notación gráfica hay una explicación textual de la sintaxis y semántica del bloque.
2. **Adornos**. UML dispone de formas para aportar más información a los gráficos. Por ejemplo, poner el nombre de una clase en cursiva indica que ésta es una clase abstracta, usar el símbolo '+' indica que un método o atributo es público, etc.
3. **Divisiones comunes**. Casi todos los bloques UML presentan divisionestipo clase/objeto, casos-de-uso/ejecuciones-de-casos-de-uso.
4. **Mecanismos de extensibilidad**. UML permite extender el lenguaje de manera controlada. Los mecanismos comprenden:
   - Estereotipos.
   - Valores etiquetados.
   - Restricciones.

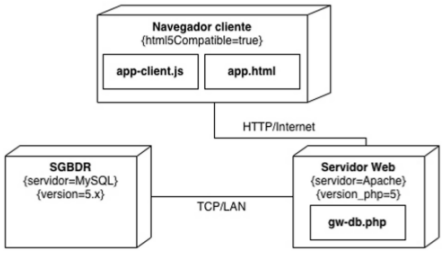

Figura 34: Ejemplo de diagrama de despliegue (fuente [Miquel and Martos, 2010a])

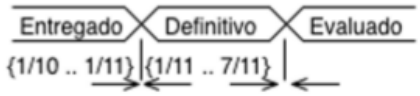

Figura 35: Ejemplo de diagrama de tiempo (fuente [Miquel and Martos, 2010a])

Divisiones comunes

1. Se permite dividir entre **clase y objeto**: una clase es una abstracción y un objeto una manifestación concreta de esa abstracción.
2. Separación entre **interfaz e implementación**. La interfaz declara un contrato y una implementación representa una realización concreta del contrato.
3. Separación entre **tipo y rol**. El tipo declara la clase de la entidad (objeto, atributo, parámetro). El rol describe el significado de la entidad en un **contexto**. Cualquier entidad que forme parte de otra entidad, tiene características de tipo y de rol.

Mecanismos de extensibilidad

- Los **estereotipos** permiten extender el vocabulario UML: podemos crear nuevos bloques que extiendan los existentes para un problema concreto.
- Un **valor etiquetado** extiende las propiedades de un estereotipo para añadir nueva información.
- Una **restricción** extiende la semántica de un bloque para añadir o modificar reglas.

Ejemplo de mecanismos de extensibilidad

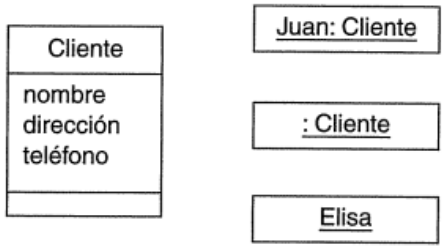

Figura 36: Ejemplo de división clase/objetos (fuente [Booch et al., 2006])

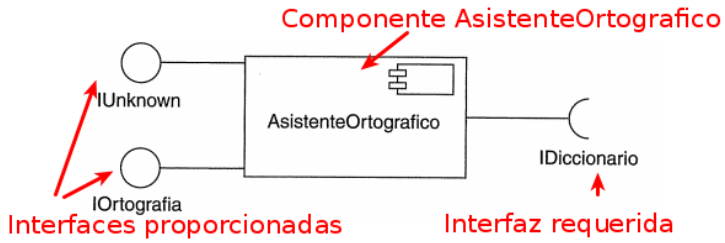

Figura 37: Ejemplo de interfaces/implementaciones (fuente [Booch et al., 2006])

6. Arquitectura software

Arquitectura software

La arquitectura es el conjunto de decisiones significativas sobre:

- La **organización** de un sistema software.
- La selección de **elementos estructurales** y sus **interfaces** a través de los cuales se construye el sistema.
- Su comportamiento en la colaboraciones entre elementos.
- La composición de los elementos estructurales y de comportamiento en subsistemas.

Vista arquitectónica 4+1 I

El modelo de vista 4+1 fue propuesto por Philippe Kruchten [Kruchten, 1995] y es ligeramente adaptado a UML por sus autores para mostrar las 5 vistas que permite UML sobre la arquitectura software.

Ejercicio: ¿qué diagramas UML serían más apropiados para cada vista?

Vista arquitectónica 4+1 en UML II

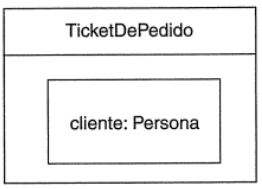

Figura 38: Ejemplo de tipo/rol (fuente [Booch et al., 2006])

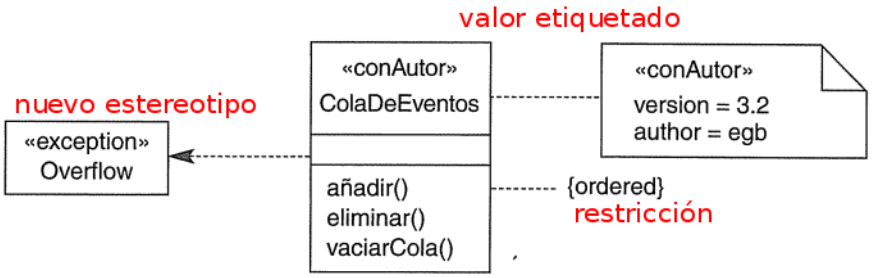

Figura 39: Ejemplo de mecanismos de extensibilidad (fuente [Booch et al., 2006])

Tenemos la solución a este ejercicio en la página de Wikipedia sobre la vista 4+1 [Wikipedia, 2014a] y en el libro Booch et al. [2006] al final del Capítulo 2. Cada vista puede existir por si misma, pero también interactuar entre si. Diferentes usuarios se pueden centrar en diferentes vistas según sus intereses.

Vista arquitectónica 4+1 en UML: vista de casos de uso

La **vista de casos de uso** de un sistema comprende los casos de uso que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas.

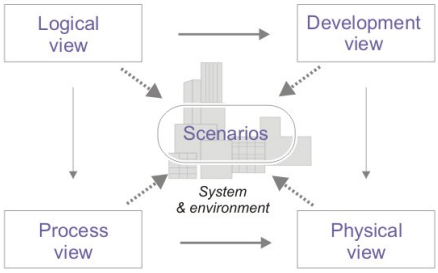

Figura 40: Modelo de vista 4+1 de Philippe Kruchten.

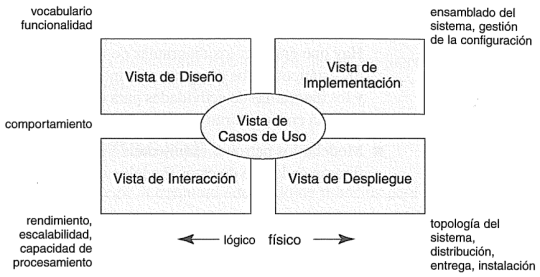

Figura 41: Modelo de vista 4+1 de los autores de UML

Esta vista no especifica realmente la organización de un sistema software pero sirve para identificar los factores que influyen en la arquitectura.

Con UML:

- Aspectos estáticos: diagramas de casos de uso
- Aspecto dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades.

Vista arquitectónica 4+1 en UML: vista de diseño

La **vista de diseño** de un sistema comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución.

Esta vista soporta principalmente los **requisitos funcionales** del sistema, entendiendo por ello los servicios que el sistema debería proporcionar a sus usuarios finales.

Con UML:

- Aspectos estáticos: diagramas de clases y de objetos, así como el diagrama de estructura compuesta (o diagrama de estructura interna de una clase).
- Aspecto dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades.

Vista arquitectónica 4+1 en UML: vista de interacción

La **vista de interacción** del sistema muestra el flujo de control entre sus diversas partes, incluyendo los posibles mecanismos de concurrencia y sincronización.

Con UML, los aspectos estáticos y dinámicos de esta Vista se capturan con los mismos tipos de diagramas que en la vista de diseño, pero con mayor atención sobre las clases activas que controlan el sistema y los mensajes que fluyen entre ellas.

Vista arquitectónica 4+1 en UML: vista de implementación

La **vista de implementación** de un sistema comprende los artefactos que se utilizan para ensamblar y poner en producción el **sistema físico**. Esta Vista se ocupa principalmente de la gestión de configuraciones de las distintas versiones del sistema, a partir de archivos más o menos independientes que pueden ensamblarse de varias formas para producir un sistema en ejecución. También está relacionada con la correspondencia entre clases y componentes lógicos con los artefactos físicos.

- Aspectos estáticos: diagramas de artefactos.
- Aspecto dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades.

Vista arquitectónica 4+1 en UML: vista de despliegue

La **vista de despliegue** de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Esta vista se ocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema físico.

- Aspectos estáticos: diagramas de despliegue.
- Aspecto dinámicos: diagramas de interacción, diagramas de estados y diagramas de actividades.

7. Conclusiones y artículos de interés

Conclusiones

UML es un lenguaje de modelado independiente del proceso. Sin embargo, para obtener el máximo beneficio de UML se debería considerar un proceso (ver final Cap. 1 de [Booch et al., 2006]):

- Dirigido por los casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

A la hora de modelar software tratando de reducir al máximo el número de líneas de código generadas por el programador, existen más alternativas no estandarizadas en si, pero muy extendidas, conviene informarse sobre las siguientes tecnologías:

- Django framework, Ruby on Rails, Symphony...
- Executable UML (xUML).
- Otras soluciones específicas) como por ejemplo el uso de XML para definir software como XUL (*XML User Interface Language*) y otros.

8. Bibliografía

Recursos y enlaces

- UML Forum: http://www.umlforum.com/
- Model-Driven Development Processes & Methods: http://www.umlforum.com/model-driven-processes/
- Rational Software Architect: Types of modeling diagrams

Fuentes

El contenido de estos apuntes se basa a su vez en otros apuntes [Miquel and Martos, 2010b] y [Ruiz, 2013a], y en las referencias citadas.

Se recomienda consultar el libro de los autores de UML [Booch et al., 2006]. En concreto los dos primeros capítulos y utilizar el resto a modo de referencia, junto con los apéndices. Algunos diagramas que hemos desarrollado en clase se encuentran en las páginas 260 (casos de uso), 278 (secuencia) y 299 (actividades).

Bibliografía y enlaces

Referencias

S. W. Ambler. An Introduction to Agile Modeling. Introduction to the Diagrams of UML 2.X, 2006. URL http://www.agilemodeling.com/essays/umlDiagrams.htm.

S. W. Ambler. UML 2.5: Do You Even Care?, 2014. URL http://www.drdobbs.com/architecture-and-design/uml-25-do-you-even-care/240163702. Consultado el 10 de noviembre de 2014.

A. E. Bell. Death by UML Fever. *ACM Queue*, 2(1), April 2004. URL http://queue.acm.org/detail.cfm?id=984495.

G. Booch, J. Rumbaugh, and I. Jacobson. *El Lenguaje Unificado de Modelado*, volume Segunda edición. Pearson Education. Addison-Wesley, 2006.

J. Heumann. Tips for writing good use cases. Technical report, IBM, May 2008. URL ftp://ftp.software.ibm.com/software/rational/web/whitepapers/RAW14023-USEN-00.pdf. White paper.

P. Kruchten. Architectural Blueprints – The "4+1" View Model of Software Architecture. *IEEE Software*, 12(6): 42–50, November 1995. URL http://dx.doi.org/10.1109/52.469759.

J. P. Miquel and J. R. Martos. *Análisis UML*. Universitat Oberta de Catalunya, 2010a. FUOC PID_00171155.

J. P. Miquel and J. R. Martos. *Requisitos*. Universitat Oberta de Catalunya, 2010b. FUOC PID_00213633.

O. M. G. OMG. Introduction to OMG's Unified Modeling Language (UML), Octubre 2014a. URL http://www.omg.org/gettingstarted/what_is_uml.htm. Consultado el 9 de noviembre de 2014.

O. M. G. OMG. Unified Modeling Language (UML) Resource Page, 2014b. URL http://www.uml.org/.

I. T. L. Ruiz. Apuntes de la Asignatura Ingeniería del Software. Tema 4: Análisis de Requisitos. 2013a.

I. T. L. Ruiz. Apuntes de la Asignatura Ingeniería del Software. Tema 5: Técnicas de Especificación y Modelado. 2013b.

A.-T. Schreiner. *Object-Oriented Programming With ANSI-C (Objekt-orientierte Programmierung mit ANSI-C)*. 2011. ISBN 9781105105685. URL http://www.cs.rit.edu/~ats/books/ooc.pdf.

Wikipedia. 4+1 architectural view model — wikipedia, the free encyclopedia, 2014a. URL http://en.wikipedia.org/w/index.php?title=4%2B1_architectural_view_model&oldid=610933807. [Online; accessed 18-November-2014].

Wikipedia. Unified modeling language — wikipedia, the free encyclopedia, 2014b. URL http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=633248379. [Online; accessed 10-November-2014].

Wikipedia. Applications of uml — wikipedia, the free encyclopedia, 2014c. URL http://en.wikipedia.org/w/index.php?title=Applications_of_UML&oldid=624611567. [Online; accessed 11-November-2014].