El proceso de desarrollo de software basado en modelos

Claudia Pons

Lifia-Universidad Nacional de La Plata
Calle 50 esq.115 1er.Piso
(1900) La Plata, Argentina
e-mail cpons@sol.info.unlp.edu.ar

Resumen

El objetivo de este artículo es describir las principales características del proceso de desarrollo de software basado en modelos, destacando la necesidad de integrar lenguajes de modelado gráficos, cercanos a las necesidades del dominio de la aplicación, con lenguajes de modelado formales, provistos de herramientas de análisis y verificación. A partir de la estandarización del lenguaje gráfico de modelado Unified Modeling Language (UML) han surgido activas discusiones acerca de la precisión semántica de sus construcciones. Mientras que el OMG fue responsable por la estandarización de UML como notación, la semántica de UML aún es un tema de investigación. Existe un número importante de trabajos teóricos que tratan diferentes partes de UML definiendo formalmente su semántica. En este artículo hemos seleccionamos los más representativos y los hemos clasificado en dos grupos: formalizaciones basadas en el modelo y formalizaciones basadas en el metamodelo. Realizamos un análisis comparativo de ambos grupos y finalmente describimos una propuesta intermedia que formaliza UML mediante una teoría formal de primer orden.

1. Introducción

En los finales de los años 70 se observó un cambio importante en la filosofía del desarrollo de software, tendiente a solucionar los problemas descriptos arriba. Tom DeMarco en su libro *Structured Analysis and System Specification* [DeMarco 79] introdujo el concepto de *ingeniería de software basada en modelos*. DeMarco destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo del sistema, tal como se realiza en otros sistemas ingenieriles. De esta forma, el modelo de un sistema provee un medio de comunicación y negociación entre usuarios, analistas y desarrolladores. Actualmente todos los métodos de desarrollo de software han adoptado esta filosofía. Lo que varía de un método a otro es la clase de modelos que deben construirse, la forma de representarlos, manipularlos, etc.

El punto de partida en el proceso de desarrollo de software es la construcción de un modelo, el cual actúa como una especificación precisa de los requerimientos que el sistema debe satisfacer. Un modelo del sistema consiste en una conceptualización del dominio del problema. El modelo se focaliza sobre el mundo real: identificando, clasificando y abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal.

1.1 Utilidad de los modelos

El modelo del sistema se usa básicamente para los siguientes propósitos:

- ***Para definir las necesidades del usuario***. El propósito principal de la especificación de un producto es definir las necesidades de los usuarios del producto.
- ***Como un medio de comunicación y negociación*** entre los usuarios y los desarrolladores y entre los distintos desarrolladores entre sí.
- ***Como un documento de referencia durante la corrección de errores*** en el producto. Luego de introducir modificaciones en el sistema, la especificación es necesaria para chequear que la nueva implementación realmente está corrigiendo los errores contenidos en la versión previa del producto.

- ***Como un documento de referencia durante la evolución*** del producto. En el caso de tener que adaptar el producto debido a cambios en los requerimientos, la especificación original debe ser adaptada para reflejar estos cambios consistentemente.

Se ha observado que la construcción de modelos es una técnica muy efectiva para detectar y resolver discrepancias entre los divergentes puntos de vista de los usuarios acerca de sus requerimientos, brindando así bases firmes para las siguientes etapas del proceso de desarrollo.

1.2 Los modelos a través del proceso de desarrollo del software

En Ingeniería de Software, tal como en otras disciplinas, existen distintas metodologías para la construcción de modelos. Los dos principales enfoques son:

- Algorítmico
- Orientado a Objetos

El desarrollo de software tradicional ha tenido un enfoque algorítmico, donde las principales piezas de un sistema son procedimientos y funciones que se ejecutan sobre estructuras de datos estáticas. En cambio el desarrollo de software contemporáneo sigue un enfoque orientado a objetos, donde el elemento central del sistema de software es el Objeto (o clase de objetos). Un objeto es un elemento perteneciente al dominio del problema o al dominio de la solución; una clase es una descripción de un grupo de objetos; cada objeto tiene una identidad (es decir, es distinguible de los otros objetos), tiene un estado (dado por los valores de sus atributos y relaciones) y un comportamiento (dado por la forma en que el objeto reacciona al recibir determinados mensajes). Una descripción más detallada del paradigma de objetos puede encontrarse en [Booch 94, Coad and Yourdon 91, Shlaer and Mellor 88].

Durante el proceso de desarrollo de software diferentes modelos del sistema en construcción son creados, tal como es ilustrado en la figura 1. Las diferencias entre estos modelos residen en los aspectos del sistema que son contemplados (ningún modelo representa al sistema completo, sino que cada modelo hace énfasis en una parte del sistema) y en el grado de abstracción (en las primeras etapas del ciclo de vida se construyen modelos más abstractos, que luego son sustituidos y/o complementados por modelos más concretos). Por ejemplo los modelos de análisis capturan sólo los requerimientos esenciales del sistema de software, describiendo lo que el sistema hará independientemente de cómo se implemente. Por otro lado, los modelos de diseño y los modelos de implementación describen como el sistema será construido en el contexto de un ambiente de

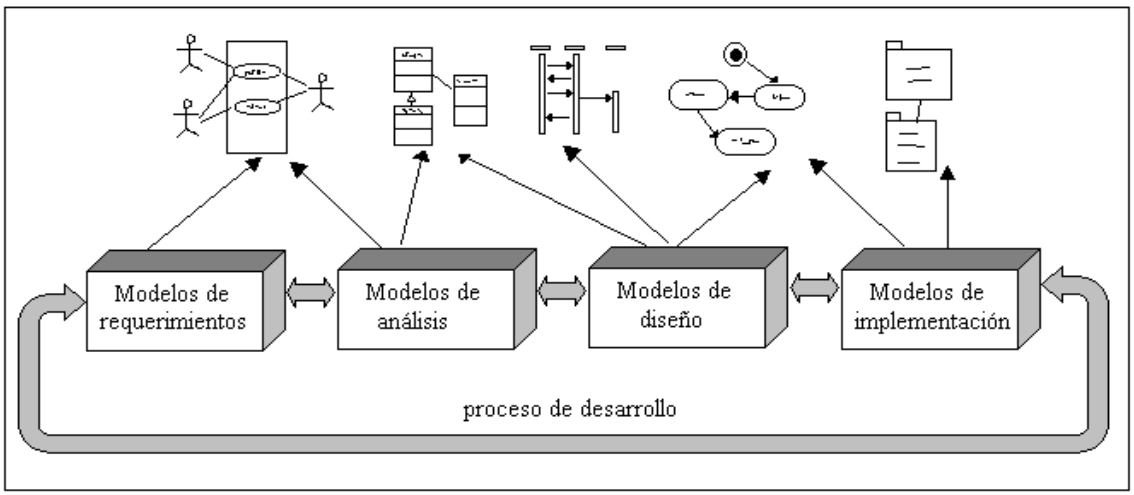


Figura 1 : los modelos a través del proceso de desarrollo

implementación determinado (plataforma, sistema operativo, bases de datos, lenguajes de programación, interfaces, etc.).

Los distintos modelos están relacionados entre sí en dos direcciones: horizontal y vertical. Cada plano vertical está formado por un grupo de submodelos que conforman una visión completa del sistema a un cierto nivel de abstracción (o en un cierto punto de su ciclo de vida). Las relaciones horizontales

representan evolución de un modelo a través del proceso de desarrollo, por ejemplo la relación entre un modelo de análisis y un modelo de diseño.

1.3 Cualidades de los modelos

El modelo de un problema es esencial para describir y entender el problema, independientemente de cualquier posible sistema informático que se use para su automatización. El modelo constituye la base fundamental de información sobre la que interactúan los expertos en el dominio del problema, los analistas y los desarrolladores de software. Por lo tanto es de fundamental importancia que exprese la esencia del problema en forma clara y precisa. Por otra parte, la actividad de construcción del modelo es una parte crítica en el proceso de desarrollo. Los modelos son el resultado de una actividad compleja y creativa y por lo tanto son propensos a contener errores, omisiones e inconsistencias. La verificación del modelo es muy importante, ya que los errores en esta etapa tienen un costoso impacto sobre las siguientes etapas del proceso de desarrollo de software.

En esta sección discutiremos las cualidades relevantes para los modelos de análisis:

- ***El modelo debe ser claro, no ambiguo y entendible***. Para que el modelo resulte útil debe ser accesible (es decir entendible y manejable) para todos sus usuarios y debe poseer una semántica precisa. Si el modelo del problema es incompleto, vago o inconsistente, o si se presta a interpretaciones erróneas, el resultado será un sistema de software cuya funcionalidad real difiera considerablemente de la funcionalidad esperada por sus usuarios.
- ***El modelo debe ser consistente,*** es decir que no debe contener información contradictoria. Dado que un sistema es representado a través de diferentes sub-modelos relacionados horizontal y verticalmente (tal como fue discutido en la sección anterior), debería ser posible especificar precisamente cual es la relación existente entre ellos, de manera que sea posible garantizar la consistencia del modelo compuesto.
- ***El modelo debe ser completo***, es decir que debe documentar todos los requerimientos necesarios. Dado que en general, no es posible lograr un modelo completo desde el inicio del proceso, es importante poder incrementar el modelo.
- ***El modelo debe ser modificable.*** Debido a la naturaleza cambiante de los sistemas actuales, es necesario contar con modelos flexibles, es decir que puedan ser fácilmente adaptados para reflejar las modificaciones en el dominio del problema.
- ***El modelo debe ser reusable***. El modelo de un sistema, además de describir el problema, también debe proveer las bases para el reuso de conceptos y construcciones que se presentan en forma recurrente en una amplia gama de problemas. El reuso permite economizar esfuerzo intelectual, tiempo y dinero.
- ***El modelo debe ser verificable.*** Existen dos aspectos a tener en cuenta, primero el modelo en sí debe ser verificado para asegurar que cumple con las expectativas del usuario. Luego, asumiendo que el modelo es correcto, puede usarse como referencia para verificar la correctitud de las implementaciones del sistema.

1.4 Modelos formales vs. modelos no-formales

El modelo del sistema se construye utilizando un lenguaje de modelado (que puede variar desde lenguaje natural o diagramas hasta formulas matemáticas). Los modelos informales son expresados utilizando lenguaje natural, figuras, tablas u otras notaciones. Hablamos de modelos formales cuando la notación empleada es un formalismo, es decir posee una sintaxis y semántica (significado) precisamente definidos. Existen estilos de modelado intermedios llamados semi-formales, ya que en la práctica los ingenieros de software frecuentemente usan una notación cuya sintaxis y semántica están sólo parcialmente formalizadas.

El éxito de los lenguajes gráficos de modelado, tales como Object Oriented Analysis (OOA) [Coad and Yourdon 91], Object Oriented system Analysis [Schlaer and Mellor 88], Object Modeling Technique (OMT) [Rumbaugh et al. 91], Booch's design method [Booch 94], and the Unified Method Language (UML) [UML 97] se basa principalmente en el uso de construcciones gráficas que

transmiten un significado intuitivo; por ejemplo un cuadrado representa un objeto, una línea uniendo dos cuadrados representa una relación entre ambos objetos. Estos lenguajes resultan atractivos para los usuarios ya que aparentemente son fáciles de entender y aplicar. Sin embargo, la falta de precisión en la definición de su semántica puede originar problemas, por ejemplo:

- Malas interpretaciones de los modelos: la interpretación que realiza el usuario que lee el modelo puede no coincidir con la interpretación que realizó el creador del modelo.
- Inconsistencia entre los diferentes modelos del sistema: la relación existente entre los diferentes sub-modelos (por ejemplo modelos de la estructura estática, modelos del comportamiento dinámico, etc.) que componen el modelo de un sistema no está precisamente especificada. Por lo tanto no es posible analizar si su integración es consistente o no lo es.
- Discusiones acerca del significado del lenguaje: dado que el significado de algunas construcciones del lenguaje no está precisamente definido, las personas involucradas en el proyecto suelen perder tiempo discutiendo las diferentes posibles interpretaciones que pueden asignarse al lenguaje.

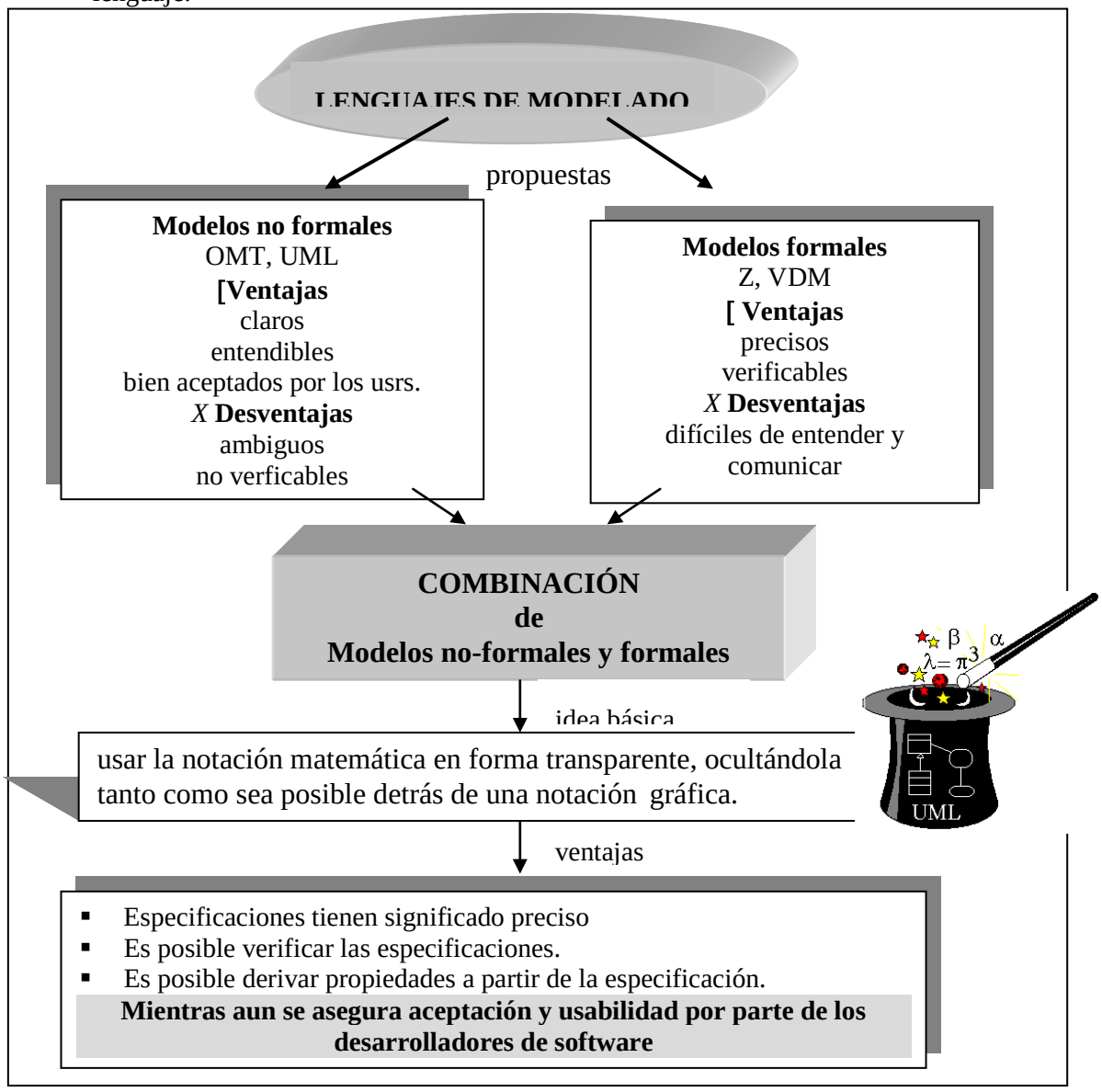


Figura 2: combinando lenguajes de modelado

Por otro lado, los lenguajes formales de modelado, tales como Z [Spivey 92], VDM [Jones 90], F-Logic [Kifer and Lausen 90], DS-Logic [Wieringa and Broersen 98] poseen una sintaxis y semántica bien definidas. Sin embargo su uso en la industria es poco frecuente. Esto se debe a la complejidad de

sus formalismos matemáticos que son difíciles de entender y comunicar. En la mayoría de los casos los expertos en el dominio del sistema que deciden utilizar una notación formal, focalizan su esfuerzo sobre el manejo del formalismo en lugar de hacerlo sobre el modelo en sí. Esto conduce a la creación de modelos formales que no reflejan adecuadamente al sistema real.

La necesidad de integrar lenguajes gráficos, cercanos a las necesidades del dominio de aplicación con técnicas formales de análisis y verificación puede satisfacerse combinando ambos tipos de lenguaje. La idea básica para obtener una combinación útil consiste en ocultar los formalismos matemáticos detrás de la notación gráfica. De esta manera el usuario solo debe interactuar con el lenguaje gráfico, pero puede contar con la base formal provista por el esquema matemático subyacente. Esta propuesta ofrece claras ventajas sobre el uso de un lenguaje informal así como también sobre el uso de un lenguaje formal, ya que permite que los desarrolladores de software puedan crear modelos formales sin necesidad de poseer un conocimiento profundo acerca del formalismo que los sustenta. Un lenguaje que posea estas características será fácilmente aceptado tanto por parte de los ingenieros de software, como por parte de los usuarios. La figura 2 esquematiza los conceptos discutidos en esta sección.

1.5 Organización

La parte restante de este artículo está organizada de la siguiente forma: en la sección 2 discutimos las distintas propuestas existentes para lograr la integración de las técnicas de modelado formales y no-formales. En la sección 3 mostramos una nueva propuesta: la M&D-theory. Finalmente la sección 4 contiene conclusiones acerca del trabajo presentado.

2. Combinando técnicas formales con técnicas no-formales

En la sección anterior hemos destacado la necesidad de integrar lenguajes de modelado gráficos, cercanos a las necesidades del dominio de aplicación con lenguajes de modelado formales provistos de herramientas de análisis y verificación. Con el objetivo de clarificar el significado y los alcances del problema consistente en la integración de ambos tipos de lenguaje, dedicaremos esta sección a discutir y comparar las diferentes soluciones que han sido propuestas al mencionado problema.

Hemos identificado básicamente cuatro propuestas diferentes para realizar la integración:

- **Suplemento**. La propuesta de suplemento consiste en enriquecer un modelo informal con conceptos formales. Buenos ejemplos de esta propuesta son Syntropy [Cook and Daniels 94] y los trabajos de Lano [Goldsack and Kent 96] y Weber [Weber 96] los cuales proponen utilizar la notación formal Z[Spivey 92] para enriquecer notaciones semi-formales.
- **Extensión**. La propuesta de extensión consiste en extender una notación formal existente, con conceptos más cercanos al dominio de la aplicación, tales como los conceptos adoptados por el paradigma de orientación a objetos. De esta forma la notación formal se vuelve más fácil de entender y manejar por parte de los desarrolladores de software. Los ejemplos más relevantes de esta propuesta son las extensiones del lenguaje Z, como por ejemplo Z++[Lano 91] y Object-Z[Duke et al. 91]. También corresponden a este grupo los lenguajes TROLL [Jungclaus et al. 96] OOZE [Alencar and Goguen 91] y MAUDE [Meseguer and Winkler 91] inspirados sobre lenguajes de especificaciones algebraicas.
- **Interfaz**. Dado un método de modelado formal, esta propuesta (ver fig.3) consiste en desarrollar una notación gráfica alternativa para facilitar la creación y visualización de los modelos. Algunos ejemplos de esta propuesta son las interfaces gráficas provistas por los lenguajes formales OASIS [Pastor and Ramos 96] y LTL [Reggio and Larosa 97].

- **Semántica**. Esta propuesta (ver fig.4) consiste en definir formalmente la semántica de un lenguaje de modelado conocido y aceptado por la comunidad. Sus principales componentes son reglas para asociar estructuras sintácticas del lenguaje de modelado con elementos en un dominio semántico formalmente definido.

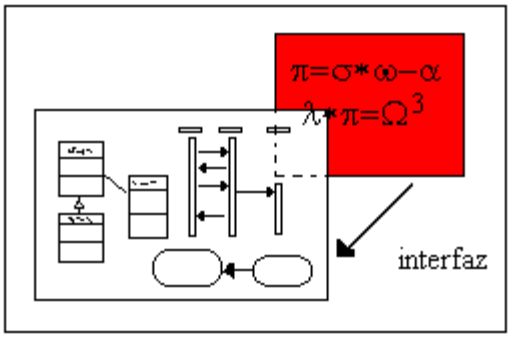


Figura 3: integración por interfaz

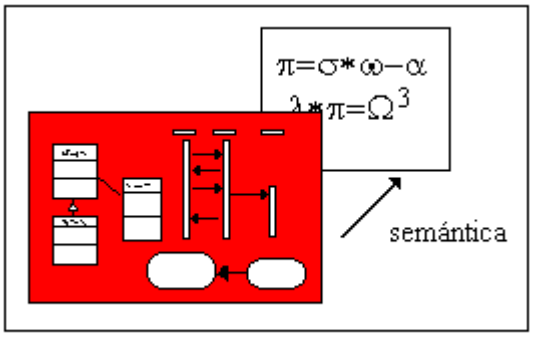


Figura 4 **:** integración semántica

De acuerdo con nuestra opinión, "semántica" constituye la propuesta más adecuada, dado que permite que especificaciones expresadas en una notación conocida y aceptada por los desarrolladores de software adquiera significado preciso a través de su "traducción" en un dominio formal. Nuestra opinión se basa en que tanto las propuestas de integración por suplemento, como las propuestas de integración por extensión requieren que los desarrolladores conozcan la notación formal, dado que ésta constituye una parte visible de la especificación. Por otra parte, la desventaja de las propuestas de integración por interfaz es que el usuario se ve forzado a adoptar un nuevo lenguaje gráfico, el cual generalmente posee notorias influencias provenientes del formalismo, que lo tornan poco intuitivo.

La principal ventaja la propuesta semántica sobre las demás propuestas reside en que el lenguaje gráfico se convierte en un lenguaje formal y por lo tanto las especificaciones escritas utilizando el lenguaje gráfico pueden ser formalmente analizadas para detectar contradicciones y ambigüedades tempranamente en el proceso de desarrollo del software. Una de las claves para el éxito de esta propuesta reside en ocultar la notación matemática tanto como sea posible tras la notación gráfica. Por ejemplo, debería ser posible utilizar la semántica formal para desarrollar herramientas CASE. Sólo los desarrolladores del lenguaje deberían usar el formalismo para construir las herramientas CASE y justificar su correctitud, mientras que los desarrolladores de software de aplicación podrían manejar los modelos gráficos sin necesidad de conocer el formalismo matemático subyacente.

A partir de la estandarización del lenguaje gráfico de modelado Unified Modeling Language (UML) [UML 97 (a)(b), UML98(a)(b)] han surgido activas discusiones acerca de la precisión semántica de sus construcciones. Mientras que el OMG fue responsable por la estandarización de UML como notación, la semántica de UML aún es un tema de investigación. Existe un número importante de trabajos teóricos (ver por ejemplo [UML98]) que tratan diferentes partes de UML definiendo formalmente su semántica. Sin embargo todavía resta un largo camino por recorrer. En particular, es difícil comparar los resultados de los respectivos artículos y es aun más difícil combinar dichos resultados con el objetivo de obtener una semántica standard para UML. Esta dificultad surge especialmente porque los diferentes trabajos utilizan diferentes métodos (o lenguajes) formales, o cubren un subconjunto de la notación o asumen una subclase particular de sistemas a ser especificados. Sin embargo, surge una clara clasificación de las propuestas en dos grupos: formalizaciones basadas en el modelo y formalizaciones basadas en el metamodelo, tal como explicaremos en las siguientes secciones de este capítulo.

2.1 Una arquitectura de dos niveles: modelo vs. metamodelo

Los lenguajes de modelado visuales son lenguajes gráficos que se usan para la especificación, visualización, documentación de productos de software como paso previo a su construcción. Generalmente el marco conceptual de las notaciones de modelado se basa sobre una arquitectura formada por distintos niveles (ver figura 5 [Odell95]). La descripción de esta arquitectura puede encontrarse por ejemplo en [UML 97b].

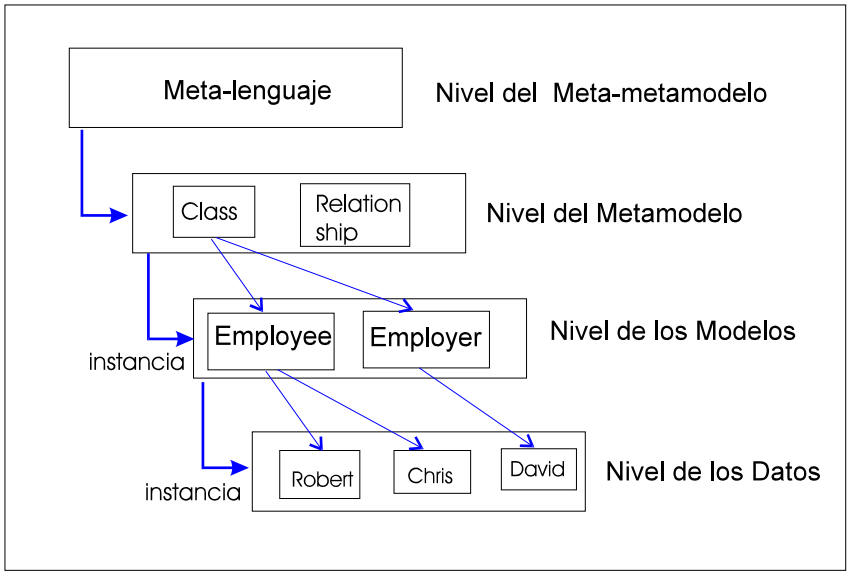


Figura 5 : arquitectura de cuatro niveles

Existen dos niveles principales:

1. **El nivel del metamodelo**. El metamodelo es un modelo para la información que puede ser expresada durante la construcción del modelo de un sistema. Básicamente, define los elementos de modelado tales como *Class_diagrams*, *State_machines*, *Sequence_diagrams* en el metamodelo de UML. Además define en que forma estos elementos se relacionan para conformar el modelo de un sistema. El metamodelo es una descripción del lenguaje de modelado en sí. Su semántica es el conjunto de todos los modelos bien formados. El metamodelo es independiente de cualquier modelo en particular, él describe los elementos del lenguaje y las restricciones que deben cumplir todos los modelos, por ejemplo: “dentro de un Classifier los nombres de los atributos no se repiten”.
2. **El nivel del modelo**. Por otro lado, el modelo es una instancia del metamodelo. El modelo (en realidad un modelo no es una unidad sino que es un conjunto de diversos modelos relacionados, tal como fue explicado en la sección 1.3) describe los objetos inherentes al dominio de la aplicación que está siendo modelada, por ejemplo: *Employer*, *Employee*, *BankAccount*, *Client*, etc. e incluye restricciones que deben ser satisfechas por los objetos del sistema, por ejemplo “no se permiten extracciones cuando el saldo de una cuenta es menor que cero”.

Existen además otros dos niveles relacionados con los anteriores:

1. ***El nivel de los datos.*** En el nivel de los datos, las entidades son instancias de las clases del modelo, por ejemplo los objetos *Robert* y *Chris* son instancias de la Clase Employee.
2. **El nivel del meta-metamodelo.** Es necesario contar con un nivel superior describiendo el lenguaje utilizado para expresar el metamodelo. Este nivel superior es llamado meta-metamodelo. Por ejemplo en esta tesis utilizaremos Dynamic Logic como meta-lenguaje.

2.2 Semántica de Lenguajes

En el área de semántica de lenguajes es reconocida la existencia de distintas dimensiones que conforman la descripción de un lenguaje: sintaxis vs. semántica, conceptos estáticos vs. conceptos dinámicos. A continuación describiremos cada una de ellas.

Sintaxis vs. Semántica:

En notación textual convencional, la sintaxis de un lenguaje es descripta por el conjunto de caracteres usados (alfabeto) y las posibles secuencias de esos símbolos (palabras, frases). Llamamos lenguaje al conjunto de todas las secuencias correctas. Cuando la notación involucra diagramas, su sintaxis es más compleja dado que ya no se limita a una secuencia lineal de caracteres, sino que comprende elementos gráficos de dos (o aún tres) dimensiones.

Por otra parte, la semántica de un lenguaje nos habla acerca del significado de cada construcción del lenguaje. Usualmente esto se hace explicando las construcciones del nuevo lenguaje en términos de conceptos ya conocidos (y bien entendidos). Este conjunto de conceptos bien entendidos es llamado el dominio semántico. Por ejemplo, dado que UML se utiliza para especificar sistemas orientados a objetos, es esperable que un dominio semántico para UML contenga todos los conceptos de la orientación a objetos tales como objetos, mensajes, generalización y especialización, composición, etc.

Notemos como estos conceptos se corresponden con los distintos niveles de la arquitectura de las notaciones de modelado (ver figura 4.5): la sintaxis de UML es descripta en el nivel del metamodelo. El nivel del modelo contiene modelos particulares, los cuales son construcciones correctas del lenguaje. Finalmente en el nivel inferior (el nivel de los datos) encontramos entidades semánticas tales como objetos. De acuerdo con esta identificación de conceptos de aquí en adelante llamaremos *modelo* a las construcciones (sintácticas) del lenguaje UML y llamaremos *sistema modelado* a la interpretación semántica de un modelo.

Estático vs. Dinámico:

En la descripción de un lenguaje existen además otras dos dimensiones ortogonales a sintaxis y semántica: aspectos estáticos vs. aspectos dinámicos. La diferenciación entre la semántica estática (es decir reglas que definen las propiedades estáticas (invariantes en el tiempo) de los objetos del dominio semántico) y la semántica dinámica (reglas que definen la evolución o comportamiento de los objetos del dominio semántico) es bien conocida y aceptada. Sin embargo, hablando de sintaxis, generalmente sólo se definen reglas de buena formación para las construcciones del lenguaje, pero no se trata el tema de su evolución o dinamismo. Es decir, la sintaxis es analizada sólo desde el punto de vista estático. La dimensión faltante, es decir sintaxis-dinámica, cobra importancia cuando se intenta dar semántica a un lenguaje de modelado en un contexto donde los modelos (las construcciones del lenguaje) pueden evolucionar (sufrir modificaciones) luego de ser creados. Es importante aclarar que no estamos hablando de cambios en la sintaxis del lenguaje (lo cual representaría evolución en el nivel del metamodelo), sino de cambios en los modelos construidos utilizando el lenguaje(es decir evolución en el nivel del modelo).

La siguiente tabla esquematiza la relación entre ambas dimensiones, donde spec es un modelo (o sea un elemento sintáctico) y sys es el sistema modelado por spec (o sea un elemento semántico).

	Modelo (dominio sintáctico)	Sistema modelado (dominio semántico)
Aspectos estáticos	Reglas de buena formación de los modelos.	Reglas de buena formación de los objetos en el sistema
Aspectos dinámicos	Evolución de los modelos.	Evolución de los objetos del sistema

En la siguiente tabla mostramos ejemplos de los conceptos generales expresados arriba:

	Modelo (spec)	Sistema modelado (sys)
Aspectos estáticos	spec no debe contener dos atributos con el mismo nombre dentro de la misma clase.	Los valores de los atributos de los objetos de sys deben corresponderse con las declaraciones en las respectivas clases.
Aspectos dinámicos	Refinamiento del modelo spec por el agregado de una nueva clase.	Los objetos de sys reaccionan al recibir mensajes.

Es importante destacar que la dimensión estática del sistema modelado constituye la semántica de los diagramas de estructura, tales como diagrama de clases y diagrama de relaciones. Mientras que la dimensión dinámica del sistema modelado puede (y debe) utilizarse para definir la semántica de los diagramas de comportamiento, tales como máquinas de estado y diagramas de interacción.

2.3 Clasificación de las propuestas para dar semántica a los lenguajes de modelado

Como hemos mencionado anteriormente, los trabajos de formalización realizados sobre notaciones de modelado (y en particular sobre UML) pueden clasificarse en dos grupos: basados en el modelo o basados en el metamodelo. El 'foco' de la formalización constituye el elemento discriminatorio entre ambos grupos. Las formalizaciones del primer grupo centran su atención sobre el nivel del modelo, mientras que las formalizaciones del segundo grupo lo hacen sobre el nivel del metamodelo.

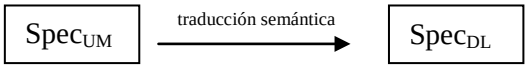
En los métodos pertenecientes al grupo basado en el modelo (ver por ejemplo [Moreira and Clark 94, France et al. 97(a), Goldsack and Kent 96, Waldoke et al. 98, Wieringa and Broersen 98, Lano and Biccaregui 98]) las especificaciones formales son generadas a partir de modelos orientados a objetos no-formales. Las especificaciones formales así obtenidas describen a los objetos del dominio de la aplicación (por ejemplo, cuentas y clientes en una aplicación bancaria). Es decir, la formalización se centra en el sistema particular que ha sido descripto utilizando la técnica de modelado.

En los métodos pertenecientes al grupo basado en el metamodelo (ver por ejemplo [France et al. 97 (b), Breu et al. 97, UML 97(b), Evans et al. 98]), el objetivo es dar una descripción precisa de los elementos que componen la técnica de modelado y proveer reglas para analizar sus propiedades. Las especificaciones formales describen a los elementos de modelado, tal como clases, asociaciones, generalizaciones, maquinas de estado, etc. Es decir, la formalización se centra en la técnica de modelado en sí misma.

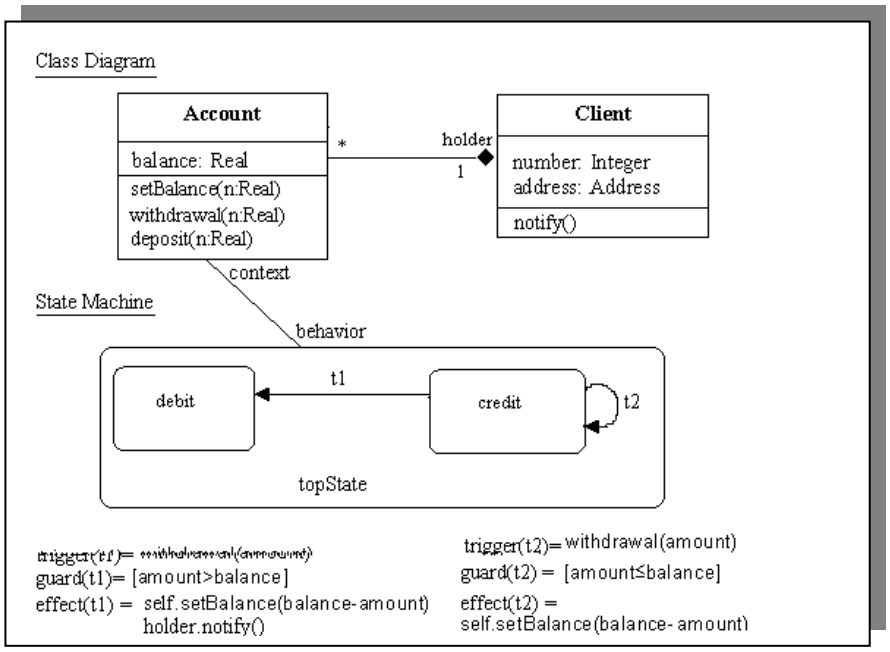
En esta sección ilustraremos las principales diferencias entre ambas propuestas mediante un ejemplo. Usaremos UML [UML 97 (a)] como notación gráfica y Dynamic Logic [Wieringa and Broersen 98] como lenguaje formal. La figura 6 muestra la especificación gráfica de un sistema bancario, la cual

llamaremos $Spec_{UML}$. El ejemplo sólo contiene las principales estructuras, otras son omitidas con el fin de ganar simplicidad y claridad.

En las siguientes secciones mostraremos la especificación formal correspondiente a $Spec_{UML}$, la cual llamaremos $Spec_{DL}$, es decir:



El contenido y la forma de $Spec_{DL}$ será diferente dependiendo de la propuesta aplicada



2.3.1 Propuestas basadas en el modelo

De acuerdo con las propuestas basadas en el modelo, la formalización $Spec_{DL}$ describe a los objetos del dominio de la aplicación, es decir Account, Client, etc. Este tipo de formalización permite la especificación de reglas de buena formación de los objetos y su comportamiento, por ejemplo: las cuentas están en estado 'credit' cuando su saldo es mayor que cero, luego de un depósito el saldo de la cuenta se incrementa, etc. A continuación mostramos una parte de $Spec_{DL}$.

Specification of Account

Sorts	BankAccount
functions	balance: BankAccount → Real
	debit, credit: BankAccount →Bool
	holder: BankAccount →Client
Actions	deposit: BankAccount x Real →Act
	withdrawal: BankAccount x Real →Act
Axioms	∀c:BankAccount ∀n:Real
	credit(c) ↔ 0≤balance
	[deposit(c,n)] balance(c)= balance(c)+n
	[withdrawal(c,n)] balance(c)= balance(c) –n
	(balance<n)→[withdrawal(c,n)]sent(notify(holde

Specification of Client

Sorts	Client
functions	number: Client → Integer
Actions	notify: Client → Act
Axioms	∀c:Client
	.....

End specification of Client

Las falencias de esta propuesta son las siguientes: $Spec_{DL}$ no incluye reglas de consistencia entre diferentes elementos de modelado, por ejemplo no es posible expresar propiedades de las clases (por ejemplo que los nombres de sus atributos no se repiten) o relaciones entre la descripción estructural de una clase (dada por el diagrama de clases) y la descripción de su comportamiento (dada por la máquina de estados). Por otra parte, modificar el modelo implicaría modificar la teoría lógica obtenida, es decir que resulta imposible representar evolución de modelos dentro de un marco formal de primer orden.

Entre las propuestas basadas en el modelo hemos seleccionado los trabajos de Kevin Lano y Jean Biccaregui [Lano and Biccaregui 98] y el método de Roel Wieringa [Wieringa 98], los cuales consisten en obtener modelos formales a partir de modelos expresados en la notación gráfica UML. En ambas propuestas los modelos formales obtenidos describen a los objetos de la aplicación:

- Lano y Biccaregui proponen una semántica axiomática para la notación UML, usando teorías estructuradas en lógica temporal. Transformaciones sobre modelos UML, tales como agregar una clase o relación son representadas en el formalismo mediante extensiones de teorías. El tratamiento que esta propuesta da a cada una de las cuatro dimensiones discutidas antes es esquematizado en la siguiente tabla:

	Modelo	Sistema modelado
Aspectos estáticos	no considerado	axiomas en lógica de primer orden
Aspectos dinámicos	transformaciones de teorías	acciones y axiomas en lógica temporal

- Roel Wieringa define una semántica formal para UML en términos de sistemas de transición rotulados. Su propuesta se enmarca dentro de una metodología para el desarrollo de software. El tratamiento que esta propuesta da a cada una de las cuatro dimensiones es esquematizado en la siguiente tabla:

	Modelo	Sistema modelado
Aspectos estáticos	no considerado	axiomas en lógica de primer orden
Aspectos dinámicos	no considerado	acciones y axiomas en lógica dinámica

2.3.2 Propuestas basadas en el metamodelo

De acuerdo con las propuestas basadas en el metamodelo, la formalización $Spec_{DL}$ describe a los elementos de modelado en sí, tal como, clases (class), atributos de las clases (attribute), operaciones

(operation), relaciones (por ejemplo: association y generalization), etc. Este tipo de formalización permite la especificación de reglas de buena formación del lenguaje, por ejemplo en la especificación de GeneralizableElement se incluyen las siguientes reglas:
[1] La raíz de una jerarquía de generalizaciones no puede tener ninguna generalización.
[2] Los elementos generalizables que son hojas no pueden tener subtipos.
Y la especificación de Classifier muestra los siguientes axiomas:
[1] Dentro de un Classifier los nombres de los atributos no se repiten.
[2] Dentro de un Classifier dos operaciones no pueden tener la misma signatura.
[3] Herencia circular no está permitida.
[4] Herencia múltiple no debe provocar conflictos de nombres.

Specification of Generalization
Sorts Generalization
Taxonomy Generalization ≤ ModelElement
functions
discriminator: Generalization → Name
supertype: Generalization → GeneralizableElement.
subtype : Generalization → GeneralizableElement.
Axioms ∀g: Generalization

[1] ¬isRoot(subtype(g))
[2] ¬isLeaf(supertype(g))
End specification of Generalization

Specification of Classifier
Sorts Classifier
Taxonomy Classifier≤GeneralizableElement
functions
attributes: Classifier→Seq of Attribute
operations: Classifier →Seq of Operation
Axioms ∀c:Classifier ∀a1,a2:Attribute

[1] (a1∈ attributes(c) ∧a2∈attributes(c) ∧name(a1) = name(a2)) → a1 =a2

[2] ∀f,g∈operations(c) hasSameSignature(f,g)) →f = g
[3] IsA(c1,c2) ∧IsA(c2 ,c1) → c2 = c1
[4] ∃c1,c2,c3:Classifier (IsA(c1,c2) ∧IsA(c1,c3) ∧c3≠c2 and
∃f:Feature (f∈allFeatures(c2) ∧ f∈allFeatures(c3))
→ ∃c:Classifier(IsA(c2,c) ∧ IsA(c3,c) ∧f∈features(c))
End specification of Classiffier

La descripción de los objetos del dominio es introducida por medio de axiomas de instanciación como se muestra en el siguiente ejemplo:

Instantiation axioms ϕ_{INST_BANK}

$\exists c_1, c_2$:Class
$(c_1 \in elements(m) \wedge name(c_1)=Account \wedge c_2 \in elements(m) \wedge name(c_2)=Client$
$\wedge \exists a$:Attribute
$(a \in attributes(c1) \wedge name(a)=balance\ type(a)=Integer)$
$\wedge \exists p1,p2$:Operation
$(p1 \in operations(c1) \wedge name(p1)=deposit \wedge parameters(p1)=<a1>$
$\wedge type(a1)=Real \wedge kind(a1)=\#in$
$\wedge p2 \in operations(c1) \wedge name(p2)=withdrawal \wedge parameters(p2)=<a2> \wedge$
$type(a2)=Real \wedge kind(a2)=\#in)$)
$\wedge \exists h$:StateMachine
$(h \in elements(m) \wedge context(h)= c_1 \wedge states(h)=\{s_1,s_2\} \wedge name(s_1)=debit \wedge$
$name(s_2)=credit \wedge transitions(h)=\{t1,t2\} \wedge trigger(t1)= p_2 \wedge source(t1)=s_2$
$\wedge target(t1)=s_1 \wedge guard(t1)=(....)\ effect(t1)= (\ ...) \wedge$
$trigger(t2)= p_2 \wedge source(t2)=s_2 \wedge target(t2)=s_2 \wedge .....\ ))$

La falencia de esta propuesta reside en que la teoría lógica describe metaentidades lo cual dificulta la representación de las entidades correspondiente al dominio de las aplicaciones (esta información particular está presente en los axiomas de instanciación).

Entre las propuestas basadas en el metamodelo hemos seleccionado los trabajos de los creadores de UML [UML 97(a)(b)] y la propuesta del "PUML group" [Evans et al. 98] integrado por Andy Evans, Robert France, Kevin Lano, Bernhard Rumpe y otros. El objetivo de estos trabajos consiste en proveer una definición precisa de la notación UML. En ambas propuestas los modelos formales se centran en la notación de modelado en sí y en las reglas para analizar sus propiedades:

- El lenguaje UML ha sido descripto básicamente en dos documentos: UML11-notation [UML 97(a)] y UML11-semantics [UML 97 (b)]. Estos documentos dan una noción clara de la sintaxis de UML, pero no logran definir precisamente la semántica del lenguaje. El documento llamado "semantics" en realidad sólo define reglas de buena formación de los modelos (es decir restricciones sobre la sintaxis) y explica la semántica de las construcciones UML de una forma poco precisa y en muchos casos contradictoria, usando lenguaje natural. El tratamiento que esta propuesta da a cada una de las cuatro dimensiones discutidas antes es esquematizado en la siguiente tabla:

	Modelo	Sistema modelado
Aspectos estáticos	Reglas OCL de buena formación de metaentidades	parcialmente considerado mediante reglas OCL de buena formación de entidades
Aspectos dinámicos	no considerado	parcialmente considerado mediante lenguaje natural

- El "PUML group" ha construido un modelo semántico preciso del lenguaje UML. El modelo incluye la descripción de la sintaxis abstracta de UML usando el lenguaje formal Z y la definición formal (también usando Z) de una función que interpreta las construcciones sintácticas en un dominio semántico formalmente definido (el cual está descripto en [Rumpe 96]). El tratamiento que esta propuesta da a cada una de las cuatro dimensiones se muestra en la siguiente tabla:

	Modelo	Sistema modelado
Aspectos estáticos	reglas de buena formación del dominio sintáctico expresadas en Z	definición de la estructura del dominio semántico expresado en Z.
Aspectos dinámicos	no considerado	función del dominio sintáctico en el dominio semántico

3. Nuestra propuesta: la M&D-theory

En esta sección presentaremos una propuesta intermedia para definir formalmente la semántica de UML. La idea básica de esta nueva formalización (la cual ha sido publicada en [Pons et al. 98] y [Pons et al. 99]) consiste en utilizar un dominio semántico que integra los dos niveles inferiores de la arquitectura de las notaciones de modelado (es decir el nivel del modelo y el nivel de los datos), permitiendo de esta manera representar los aspectos estáticos y dinámicos tanto del modelo como del sistema modelado dentro de un marco formal de primer orden, que llamaremos M&D-theory.

3.1 Dicotomía de entidades

Las entidades descriptas por la M&D-theory se clasifican en dos conjuntos disjuntos: Entidades de modelado y Entidades modeladas. Esta dicotomía puede observarse en la figura 7. Las entidades de modelado se corresponden con construcciones (sintácticas) correctas del lenguaje UML, tales como clases (Class) y a máquinas de estados (StateMachine). En contraste, las entidades modeladas, tales como objetos (Object) y conexiones (Link) representan los datos del sistema modelado. Estas entidades se relacionan de distintas formas:

- *Algunas entidades de modelado se relacionan con otras entidades de modelado*. Por ejemplo las clases (Class) están relacionadas con las máquinas de estado (StateMachine) a través de la relación rotulada con el nombre 'behavior'. Esto indica que las máquinas de estado se utilizan para definir el comportamiento de las instancias de cada clase. Otro ejemplo puede observarse en la relación entre máquina de estados y estados (State), la cual indica que una máquina de estados está compuesta por un conjunto de estados.

- *Algunas entidades modeladas se relacionan con otras entidades modeladas*. Por ejemplo, la relación rotulada con el nombre 'slot' entre Object y AttributeLink, indica que un objeto se relaciona con los valores de sus atributos.

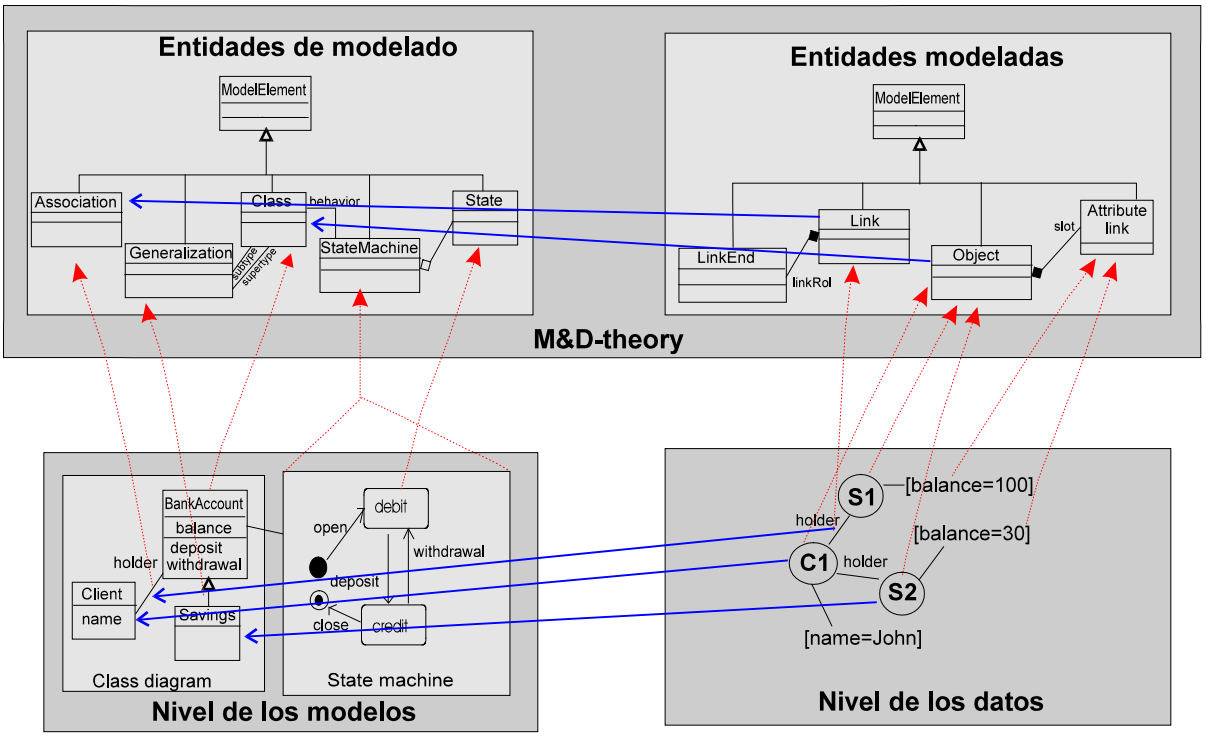


Figura 7: dicotomía de metaentidades

- *Algunas entidades de modelado se relacionan con entidades modeladas*. Existe una relación muy especial entre algunas entidades modeladas y su correspondiente entidad de modelado. Esta relación representa 'instanciación', como por ejemplo los objetos (Object) son instancias de una Clase (Class), mientras que las conexiones (Link) son instancias de una asociación (Association).

La M&D-theory provee dos clases diferentes de relación de instanciación:

- **Instanciación intra-nivel**: es la relación entre una entidad modelada y la entidad principal que la modela. En la figura 7 las líneas llenas representan algunas instanciaciones intra-nivel. Por ejemplo, Object es instancia de Class, Link es instancia de Association.

- **Instanciación inter-nivel:** es la relación de instanciación provista por el metalenguaje (dynamic logic en este caso). En la figura 7 las líneas punteadas representan algunas instanciaciones inter-nivel. Por ejemplo, BankAccount es instancia de Class, holder es instancia de Association, C1 es instancia de Object, S2 es instancia de Object.

3.2 Ventajas de la integración

La integración de entidades de modelado y entidades modeladas dentro de la misma teoría permite representar los aspectos estáticos y dinámicos tanto del modelo como del sistema modelado dentro de un marco formal de primer orden. la siguiente tabla esquematiza el tratamiento que nuestra propuesta da a cada una de las cuatro dimensiones discutidas en el capítulo anterior:

	Modelo	Sistema modelado
Aspectos estáticos	Axiomas de primer orden sobre entidades de modelado	Axiomas de primer orden sobre entidades modeladas
Aspectos dinámicos	Acciones y axiomas modales sobre entidades de modelado.	Acciones y axiomas modales sobre entidades modeladas

Contar con una estructura formal de primer orden, en contraste con una estructura de orden superior, facilita los procedimientos para calcular la validez de las fórmulas. A pesar de que la lógica de primer orden es no-decidible (y por lo tanto también lo es la lógica dinámica de primer orden), los sistemas de computación satisfacen ciertas propiedades (por ejemplo, se interpretan sobre estructuras aritméticas, el estado de un programa en un momento dado queda determinado por un conjunto finito de valores) las cuales permiten calcular la validez de las fórmulas dinámicas en forma finita y efectiva.

3.3 La M&D-theory

La M&D-theory (*Models&Data theory)* es una teoría dinámica de primer orden (la descripción detallada de esta teoría y sus aplicaciones puede leerse en [Pons et al. 98] y [Pons et al. 99]), formada por una signatura (la cual define el lenguaje de la teoría) y un conjunto de axiomas:

$$\text{M\&D-theory}=(\Sigma_{M\&D}, \phi_{M\&D})$$

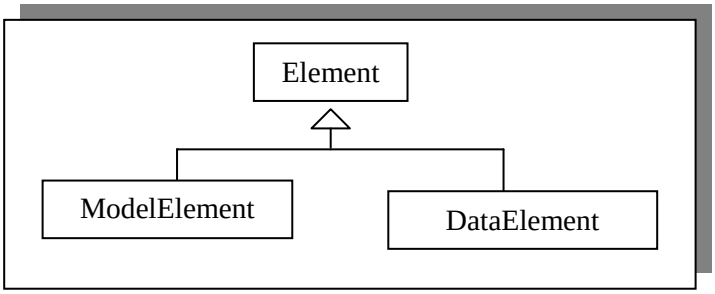
La signatura de la teoría, $\Sigma_{M\&D}=(\ (\mathbf{S}, \leq), \mathbf{F}, \mathbf{P}\ )$, es una signatura en lógica dinámica con las siguientes características especiales:

- La signatura $\Sigma_{M\&D}$ incluye a la signatura Σ_{UML}. Esto significa, por ejemplo que existe un conjunto distinguido de sorts $\mathbf{S}_{UML} \subseteq \mathbf{S}$. Estos sorts representan a los elementos de modelado, tales como Class y StateMachine. Usualmente son llamados metaclases o metasorts.

- La signatura $\Sigma_{M\&D}$ incluye a la signatura Σ_{SYS}. Esto significa, por ejemplo que existe un conjunto distinguido de sorts $\mathbf{S}_{SYS} \subseteq \mathbf{S}$. Estos sorts representan a los objetos del sistema y sus relaciones, tales como Object y Message.

- Existe un sort universal llamado Element. Es decir, $\text{Element} \in \mathbf{S} \wedge (\forall s \in \mathbf{S}) s \leq \text{Element}$.

- Los conjuntos de sorts $\mathbf{S}_{UML}$ y $\mathbf{S}_{SYS}$ son disjuntos y sus elementos no están relacionados por $\leq$. Es decir que $(\forall u \in \mathbf{S}_{UML})(\forall s \in \mathbf{S}_{SYS}) \neg(u \leq s \vee s \leq u)$. Además cada uno de estos dos conjuntos tiene un sort distinguido (DataElement y ModelElement respectivamente) encabezando la jerarquía, es decir, $\text{DataElement} \in \mathbf{S}_{SYS} \wedge (\forall s \in \mathbf{S}_{SYS}) s \leq \text{DataElement}$ y por otro lado, $\text{ModelElement} \in \mathbf{S}_{UML} \wedge (\forall s \in \mathbf{S}_{UML}) s \leq \text{ModelElement}$. La siguiente figura ilustra esta jerarquía de sorts:



Por otra parte los axiomas de la teoría, están integrados por distintos grupos de axiomas. Es decir, $\phi_{M\&D}$ es la conjunción de tres fórmulas: $\phi_{M\&D} = \phi_{UML} \wedge \gamma_{SYS} \wedge \phi_{JOINT}$. Primeramente, ϕ_{UML} es la fórmula sobre Σ_{UML} que define las características de los elementos de modelado (esta fórmula se obtiene mediante la conjunción de todos los axiomas del nivel de los modelos). Luego, γ_{SYS} es la fórmula construida sobre Σ_{sys} que describe las características de los elementos modelados (esta fórmula se obtiene mediante la conjunción de todos los axiomas del nivel de los datos). Finalmente, ϕ_{JOINT} es una fórmula construida utilizando el lenguaje integrado *M&D* y por lo tanto puede expresar tanto propiedades de los modelos, como propiedades de los datos, como propiedades relacionando ambos niveles. La fórmula ϕ_{JOINT} se obtiene uniendo dos grupos de axiomas:

- $\phi_{GENERAL}$ (describe características generales a todos los sistemas.
- $\phi_{SPECIFIC}$ (describe características específicas de cada sistema). Esta fórmula es la conjunción de:
    - axiomas de instanciación ϕ_{INST} .
    - axiomas de completación ϕ_{COMP}.

3.4 La Interpretación semántica de UML

Las principales componentes de la interpretación semántica de UML son reglas para asociar estructuras sintácticas del lenguaje de modelado con elementos en un dominio semántico formalmente definido. En las siguientes secciones describiremos el dominio semántico y las correspondientes reglas de interpretación para las construcciones de UML.

3.4.1 El dominio semántico

El dominio semántico donde las construcciones de UML serán interpretadas está formado por sistemas de transición de la forma $M=(W,w_o,R,U,I)$. Un sistema de transición es un conjunto de mundos posibles con una relación de transición entre ellos, tal como fue descripto en el capítulo 3. Los dominios para los mundos son álgebras heterogéneas cuyos elementos incluyen tanto datos (por ejemplo objetos) como meta-datos (por ejemplo clases). En la figura 5.12 mostramos una estructura de mundos posibles, donde se observa la dicotomía datos (sobre fondo gris) vs. meta-datos (sobre fondo blanco).

El conjunto de relaciones de transición entre mundos está particionado en dos conjuntos disjuntos:

- Un conjunto de transiciones que representan modificaciones sobre la especificación del sistema (evolución de los meta-datos).
- Un conjunto de transiciones que representan modificaciones sobre los datos del sistema (evolución de los datos).

La figura 8 muestra un ejemplo de evolución en ambas direcciones. Es importante notar que como consecuencia de la evolución de la especificación (es decir la modificación de la transición t2 agregándole un nuevo efecto: enviar el mensaje notify al holder) el comportamietno del objeto ***o*** ha sufrido una modificación co-lateral.

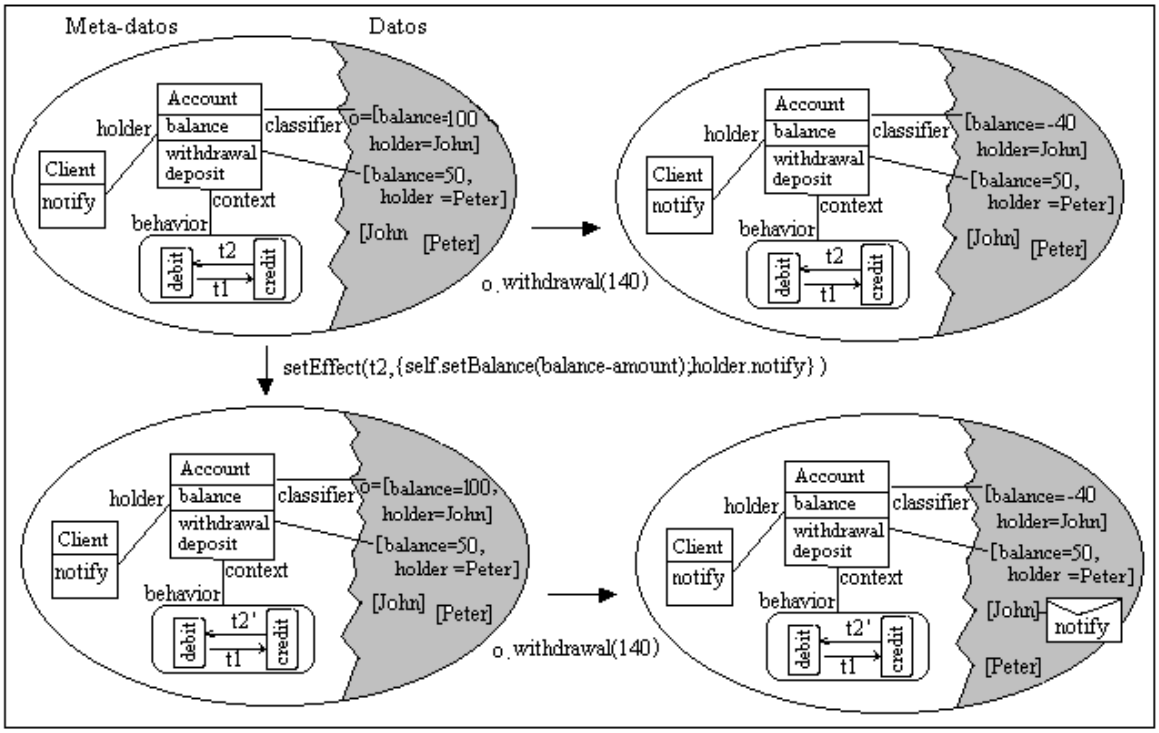


***Figura* 8** **:** evolución en ambas direcciones

3.4.2 Función de interpretación semántica

En las secciones anteriores hemos definido el dominio semántico donde interpretaremos al lenguaje de modelado UML. El paso siguiente consiste en establecer las relaciones entre los conceptos sintácticos de UML y los conceptos del dominio semántico, a través de la definición de la función de interpretación semántica que llamaremos **Sem**.

Sem: ConstruccionesUML → DominioSemántico

Es importante destacar que la interpretación semántica de UML que proporsionaremos no es directa, sino que se obtiene en dos pasos:

1- interpretación (o traducción) del lenguaje UML en una teoría M&D.

2- interpretación semántica de la teoría M&D.

Es decir,

ConstruccionesUML $\rightarrow^{translation}$ **M&D-theory** $\rightarrow^{semantics}$ **Dominio-semántico**

Por lo tanto la función de interpretación semántica **Sem** es la composición de ambas funciones, **Sem**=***semantics*** o ***translation***

La semántica para una especificación dinámica spec es el conjunto de todos los modelos (sistemas de transición entre mundos posibles) que son isomorfos al modelo min-max (el modelo min-max es el elemento **≤r** maximal del conjunto de modelos minimales de spec). Es decir, ***semantics***(spec) = {M | M≡min-max(spec) }. Para obtener información sobre semántica de especificaciones dinámicas puede consultarse [Wieringa and Broersen 98].

Para obtener la interpretación semántica de UML sólo nos resta definir la función ***translation***. La función ***translation*** asocia cada modelo UML bien formado con su correspondiente M&D-theory. Esta

función está determinada por medio de un conjunto de reglas que permiten crear una M&D-theory a partir de los distintos submodelos relacionados que componen un modelo UML. Las reglas trabajan sobre una teoría básica (Llamamos básica a una M&D-theory cuyos únicos axiomas son ϕ_{UML} , ϕ_{SYS} y $\phi_{GENERAL}$, es decir que no contiene axiomas específicos $\phi_{SPECIFIC}$). Las reglas van enriqueciendo progresivamente esta teoría, agregando dos clases de axiomas específicos:

- de instanciación ϕ_{INST} (describen cuales elementos de modelado son usados en este modelo)
- y de completación ϕ_{COMP} (describen características especiales del sistema modelado)

4. Conclusiones

Hemos descripto las diferentes propuestas para la integración de lenguajes de modelado gráficos (en particular el lenguaje standard UML) con lenguajes de modelado formales. Hemos destacado que la técnica consistente en definir formalmente la semántica de un lenguaje de modelado conocido y aceptado por la comunidad constituye la propuesta más adecuada, dado que permite que el lenguaje gráfico se convierta en un lenguaje formal y por lo tanto las especificaciones escritas utilizando el lenguaje gráfico pueden ser formalmente analizadas para detectar contradicciones y ambigüedades tempranamente en el proceso de desarrollo del software.

Entre los numerosos trabajos teóricos que tratan diferentes partes de UML definiendo formalmente su semántica hemos seleccionado los más representativos y los hemos descripto y clasificado en dos grupos: formalizaciones basadas en el modelo y formalizaciones basadas en el metamodelo. Las principales ventajas y desventajas de cada propuesta son esquematizadas en la tabla de la figura 9.

Propuestas	Ventajas	Desventajas
basadas en el modelo	▪ Es apropiado para la especificación de información inherente al dominio de la aplicación. ▪ Permite detectar inconsistencias y errores en las especificaciones escritas en el lenguaje de modelado.	▪ No permite expresar reglas de consistencia entre diferentes elementos de modelado, por ejemplo, propiedades acerca de la estructura del sistema, tales como relaciones entre clases no pueden ser expresadas. ▪ No permite representar evolución del modelo en un formalismo de primer orden.
basadas en el metamodelo	▪ Permite especificar reglas de consistencia entre diferentes elementos de modelado (por ejemplo entre clases y maquinas de estado). ▪ Permite detectar inconsistencia y errores en la definición del lenguaje de modelado en sí. ▪ Provee un marco formal claro y simple para expresar evolución y refinamiento de modelos.	▪ Es difícil representar y analizar la información correspondiente al dominio de las aplicaciones descriptas con el lenguaje.

Figura 9: ventajas y desventajas de cada grupo

Finalmente, hemos descripto la M&D-theory, una nueva propuesta para definir formalmente la semántica de UML. La idea básica de esta nueva formalización consiste en utilizar un dominio semántico que integra a las entidades de modelado y a las entidades modeladas, permitiendo de esta manera representar los aspectos estáticos y dinámicos tanto del modelo como del sistema modelado

dentro de un marco formal de primer orden. La siguiente tabla esquematiza el tratamiento que la M&D-theory da a cada una de las cuatro dimensiones discutidas en la sección 2:

	Modelo	Sistema modelado
Aspectos estáticos	Axiomas de primer orden sobre entidades de modelado	Axiomas de primer orden sobre entidades modeladas
Aspectos dinámicos	Acciones y axiomas modales sobre entidades de modelado.	Acciones y axiomas modales sobre entidades modeladas

Contar con una estructura formal de primer orden, en contraste con una estructura de orden superior, facilita los procedimientos para calcular la validez de las fórmulas. A pesar de que la lógica de primer orden es no-decidible (y por lo tanto también lo es la lógica dinámica de primer orden), los sistemas de computación satisfacen ciertas propiedades (por ejemplo, se interpretan sobre estructuras aritméticas, el estado de un programa en un momento dado queda determinado por un conjunto finito de valores) las cuales permiten calcular la validez de las fórmulas dinámicas en forma efectiva.

Referencias

[Alencar and Goguen 91] A.Alencar and J.Goguen, OOZE: an object-oriented Z environment, In P.America editor, ECOOP'91 Proceedings, Lecture Notes in Computer Science vol.512. Springer-Verlag, July 1991.

[Booch 94] G.Booch, Object Oriented Analysis and Design with Applications, Second Edition, Addison-Wesley Publishing Company, Inc, 1994.

[Breu et al. 1997] R.Breu, U.Hinkel, C. Hofmann, C.Klein, B.Paech, B.Rumpe and V.Thurner. Towards a formalization of the unified modeling language. In ECOOP'97 proceedings, LNCS 1241, Springer, June 1997.

[Coad and Yourdon 91] P.Coad and E.Yourdon, "Object Oriented Analysis", Yourdon Press, Englewood Cliffs,NJ, 1991.

[Cook and Daniels 94] S.Cook and J.Daniels, Let's get formal, Journal of Object-Oriented Programming(JOOP), July-August 1994.

[DeMarco79] Tom DeMarco, Structured Analysis and System Specification. Englewood Cliffs, NJ:Prentice Hall, 1979.

[Duke et al. 91] R.Duke, P.King, G.rose and G.Smith. The Object-Z specification language. In T.Korson, V.Vaishnavi and B.Meyer, editors, Technology of Object-Oriented Languages and Systems:TOOLS 5. Prentice Hall, 1991.

[Evans et al 98] Andy Evans, Robert France, Kevin Lano and Bernhard Rumpe, Developing the UML as a formal modeling notation. In Muller and Bezivin editors, UML'98 Beyond the notation, International workshop, France, 1998.

[France et al. 97(a)] R.France, J.Bruel and M.Larrondo-Petrie. An integrated object-oriented and formal modeling environment, Journal of Object Oriented Programming (JOOP), 1997.

[France et al. 97(b)] R.France, E.Evans and K.Lano, The UML as a formal modeling notation, In Kilov, Rumpe and Simmons editors, OOPSLA'97 Workshop on Object-oriented Semantics, TUM-I9737, Institut Fur Informatik, Technische Universitat Munchen.

[Goldsack and Kent 96] "Formal Methods and Object Technology", Chapter 3: LOTOS in the Object-oriented analysis process. Editors S.J. Goldsack, S.J.H. Kent. Serie FACIT, Springer-Verlag Berlin Heidelberg New York, 1996.

[Jungclaus et al. 96] Ralf Jungclaus,G.Saake,T.Hartmann,C.Sernadas,"TROLL- a language for o-o specifications of information systems", ACM Transactions on IS, vol.14 no.2. April 96.

[Jones 90] C B Jones, Systematic software construction using VDM. Prentice Hall, 1990.

[Kifer and Lausen 90] M.Kifer and G.Lausen, "F-Logic: a higher order language for reasoning about objects, inheritance and scheme. in proceedings of the ACM SIGMOD symposium on principles of database systems,SIGMOD RECORD, Vol.18, No.6, June 1990.

[Lano 91] K.Lano. Z++, An object-oriented extension to Z. In John Nicholls, editor, Z user workshop, Oxford 1990, Workshops in Computing, Springer Verlag, 1991.

[Lano and Biccaregui 98] Formalizing the UML in Structured Temporal Theories, Kevin Lano, Jean Biccaregui, Second ECOOP Workshop on Precise Behavioral Semantics, TUM-I9813,

Technische Universitat Munchen.
[Meseguer and Winkler 91] J.Meseguer, T.Winkler. “Parallel Programming in Maude”. Proceedings of Research Directions in High Level Parallel Programming Languages. France, 1991.
[Moreira and Clark 94] A.Moreira,and R. Clark. “Combining Object_Oriented Analysis and Formal Description Techniques”, In 8th European Conference on Object Oriented Programming, Proceedings. LNCS 821. 1994.
[Odell 95] James Odell. Meta-modeling. In OOPSLA'95 Workshop on Metamodeling in OO, October 1995
[Pastor and Ramos 95] O. Pastor, I.Ramos, "Oasis 2.2 : A Class-Definition Language to Model Information System Using an Object-Oriented Approach". SPUPV-95.788, Universitat P. de Valencia.
[Pons et al. 98] C. Pons, G.Baum and M.Felder, Integrating object-oriented model with object-oriented meta-model into a single formalism, Second ECOOP Workshop on Precise Behavioral Semantics, Brussels, Belgium, Ed: H.Kilov, B.Rumpe, Technische Universitat Munchen, Report TUM-I9813, ECOOP’98 Workshop Readers, Lectures Notes in Computer Science. July 1998.
[Pons et al. 99] Foundations of Object-oriented modeling notations in a dynamic logic framework, C.Pons, G.Baum, M.Felder, In Fundamentals of Information Systems, Chapter 1, T.Polle,T.Ripke,K.Schewe Editors, Kluwer Academic Publisher, 1999.
[Reggio and Larosa 97] g.Reggio and M.Larosa, A graphic notation for formal specification of dynamic systems, proceedings of FME’97 4th International symposium of formal methods Europe, Lecture Notes in Computer Science 1313, Springer.
[Rumbaugh et al. 91] J.Rumbaugh, M.Blaha, W.Premerlani, “Object Oriented Modeling and Design”, Prentice Hall, 1991.
[Rumpe 96] Ph.D Thesis of Bernhard Rumpe, Technische Universitat Munchen, Germany, 1996.
[Shlaer and Mellor 88] S.Shlaer and J.Mellor, Object Oriented Systems Analysis: Modeling the World in Data, Yourdon Press Computing Series, Yourdon Press, Englewood Cliffs, NJ, 1988.
[Spivey 92] M.Spivey. The Z notation: a reference manual. Prentice Hall, Englewood Cliffs, NJ, Second edition, 1992.
[UML 97] The Unified Modeling Language (UML) Specification – Version 1.1, September 1997. Joint Submission to the Object Management Group (OMG), ver http://www.omg.org.
[UML 97 (a)] UML Notation Guide, Version 1.1, September 1997. Part of [UML 97]
[UML 97 (b)] UML Semantics, Version 1.1, September 1997. Part of [UML 97].
[UML 97 (c)] UML Extension for Objectory Process for S.E. 1.1, September 1997. Part of [UML 97].
[UML 98 (a)] The UML User Guide, Booch, Rumbaugh and Jacobson. Addison Wesley Longman, Inc, 1998.
[UML 98 (b)] The UML Reference Manual, Rumbaugh, Jacobson and Booch. Addison Wesley, Inc, 1998.
[UML-conference 98] Proceedings of the UML´98 conference, Mulhouse, France, June 1998. Lecture Notes in Computer Science, Springer-Verlag.
[Waldoke et al. 98] S.Waldoke, C. Pons, C.Paz Mezzano and M. Felder, A Formal Approach to Practical Object Oriented Analysis and Design, Proceedings of Argentinean Symposium on Object Orientation, ed: SADIO (Sociedad Argentina de Informática e Inv. Operativa), Buenos Aires, 1998.
[Wieringa and Broersen 98] R.Wieringa and J.Broersen, Minimal Transition System Semantics for Lightweight Class and Behavior Diagrams, In PSMT Workshop on Precise Semantics for Software Modeling Techniques, Ed: M.Broy, D.Coleman, T.Maibaum, B.Rumpe, Technische Universitat Munchen, Report TUM-I9803, April 1998.