Métodos Formales Orientados a Objetos

Francisco José Galán Morillo
José Miguel Cañete Valdeón
Dept. Lenguajes y Sistemas Informáticos
ETSI Informática
Av. Reina Mercedes s/n, 41012 Sevilla
954552773
galanm@lsi.us.es

ÍNDICE

1. INTRODUCCIÓN

El área de los Métodos Formales Orientados a Objetos (MFOO en adelante) se ocupa de la descripción de software de manera precisa y rigurosa. Tal objetivo obliga a la utilización de lenguajes de especificación de software de naturaleza matemática.

Es interesante destacar, que en los últimos años se ha evidenciado un acercamiento entre dos comunidades de ingeniería de software aparentemente muy distintas: la "comunidad de los métodos formales" y la "comunidad de los métodos convencionales". Syntropy [CoD94] y Catalysis [SoW98] representan dos buenos ejemplos de este acercamiento. Históricamente, los métodos formales centraron sus objetivos en la calidad, descuidando, en gran medida, el entorno en el que debían aplicarse. La falta de educación general y el déficit de herramientas los convirtieron en recursos bastante ideales y difíciles de manejar. Si embargo, esta crítica no recae sólo en el debe de los métodos formales. La sistematización progresiva de la producción de software necesita de notaciones expresivas y, evidentemente, formales. No hay automatización sin formalización. El camino, sin duda, es largo y las fertilizaciones cruzadas entre ambas comunidades, serán necesarias.

La organización del presente documento es la siguiente: en la sección 2 se establecen la definición de método formal orientado a objetos, para ello, se definen los conceptos formal y orientación a objetos. A continuación, se establece una organización del conocimiento en el área MFOO mediante una descomposición en temas. La descomposición distingue conceptos básicos y fundamentales, conocimientos sobre formalismos matemáticos y métodos formales orientados a objetos. La sección 4 establece una justificación razonada de la descomposición. La sección 5 permite al lector identificar rápidamente las referencias de interés para un determinado tema. Finalmente, se establece una selección de las referencias recomendadas.

2. DEFINICIÓN DE MÉTODO FORMAL ORIENTADO A OBJETOS

Un MFOO es *un conjunto de técnicas de modelado para especificar, desarrollar y verificar sistemas software mediante el uso del lenguaje matemático y características orientadas a objetos* [FM].

Por sistema software orientado a objetos se entiende, de forma general, a toda colección de objetos que colaboran entre sí para conseguir un propósito (objetivos del sistema software).

Muchos de los métodos formales presentes en la literatura aparecen como métodos formales extendidos con conceptos de orientación a objetos. Es decir, la existencia de un lenguaje formal suele ser previa a la existencia del lenguaje orientado a objetos. Este hecho ha condicionado en gran medida las diferentes propuestas y así queda reflejado en el presente documento.

Dada la diversidad de trabajos y formalismos existentes en la actualidad, es necesario establecer qué se entiende por formal y qué se entiende por orientación a objetos. El término formal queda caracterizado por la categoría de modelos matemáticos utilizados. Ejemplos de formalismos son: lógica de primer orden, álgebras, redes de Petri y lógica temporal. Por otra parte, el término orientado a objeto queda caracterizado por los conceptos de objeto, ocultación, interacción, abstracción, concurrencia, clase, herencia, (sub)tipo y genericidad.

Los formalismos basados en *lógica de primer orden y teoría de conjuntos* [MaW85] permiten especificar el sistema mediante un concepto formal de estado y operaciones sobre estados. Con este propósito, datos y relaciones/funciones se describen en detalle y sus propiedades se expresan en lógica de primer orden. La semántica de los lenguajes está basada en la teoría de conjuntos. Durante el diseño e implementación del sistema, los elementos descritos matemáticamente pueden modificarse preservando las características esenciales de la especificación inicial.

Los formalismos *algebraicos* [EhM85]proponen la descripción de estructuras de datos estableciendo el nombre de los diferentes

conjuntos de datos, funciones básicas y propiedades. No hay concepto de estado modificable en el tiempo. Las propiedades se describen mediante fórmulas, generalmente, ecuaciones. Diferentes modelos semánticos se han propuesto para caracterizar las especificaciones algebraicas: modelos iniciales, finales y laxos. Los modelos iniciales identifican términos cuya igualdad puede ser probada desde los axiomas. Los modelos finales identifican términos diferentes cuya desigualdad puede ser probada desde sus axiomas y los modelos laxos identifican todas las álgebras cuyos elementos pueden denotarse con términos sin variables. Para describir sistemas (de estructuras de datos) se propone una amplia gama de operadores de composición y parametrización.

Los formalismos basados en *redes de Petri* [MaP91] establecen la noción de estado de un sistema mediante lugares que pueden contener marcas. Un conjunto de transiciones (con pre y post condiciones) describe la evolución del sistema entendida como consumo y producción de marcas en varios puntos de la red. Existe una extensa literatura sobre las redes de Petri y sus caracterizaciones semánticas: semántica de entrelazado y semántica basada en concurrencia real.

Los formalismos basados en *lógica temporal* [MaP91] se usan para describir sistemas concurrentes y reactivos. Un aspecto común asociado a las diferentes lógicas temporales es la noción de tiempo y estado. Una especificación escrita en lógica temporal describe las secuencias admisibles de estados (incluyendo estados concurrentes) para el sistema especificado.

Por otra parte, con el propósito de unificar terminologías sobre conceptos básicos orientados a objetos se proponen las siguientes definiciones:

Objeto: entidad discreta con límites bien definidos e identidad que encapsula estado y comportamiento [RJB99].

Ocultación: habilidad para ocultar el estado de los objetos para el exterior, la única forma de interactuar con un objeto es a través de unos de sus servicios (comportamiento) [GBD+97].

Interacción: Especificación de la comunicación entre objetos. Se proponen dos modelos: interacción síncrona e interacción asíncrona [RJB99].

Abstracción: Características esenciales de un objeto que lo distingue de otros. Las características esenciales son relativas al observador [Boo94].

Concurrencia: La realización de dos o más actividades durante el mismo intervalo de tiempo [RJB99].

Clase: Habilidad para describir aspectos comunes de objetos (intensión) y habilidad para describir colecciones existentes de objetos (extensión) [GBD+97].

Herencia: La reutilización o modificación de una clase para obtener otra nueva [GBD+97].

Subtipo: Dada una jerarquía de tipos, todo subtipo respeta el principio de sustitución respecto al tipo padre en la jerarquía. El principio de sustitución se puede establecer a tres niveles: nivel débil (sólo se consideran los perfiles de las operaciones), nivel fuerte (se considera además la semántica de las operaciones) y nivel observacional (sólo se consideran un subconjunto de las operaciones) [GBD+97].

Genericidad: Capacidad para establecer descripciones parametrizables de clases y tipos [GBD+97].

3. DESCOMPOSICIÓN DE TEMAS

Actualmente, los MFOOs representan un área de investigación joven y en continua evolución. Por ello, muchas de las referencias utilizadas en este estudio son artículos de investigación.

Los MFOOs representan un área de conocimientos exigente porque obligan a amalgamar conocimientos de índole matemático-deductivos, conocimientos sobre diseño de sistemas OO y conocimientos sobre modelos de producción de software OO. Por lo tanto, la aproximación al tema debe hacerse de forma incremental. Inicialmente se necesita introducir los principios básicos y generales sobre los conocimientos previamente aludidos. Posteriormente, se asume que los MFOOs se encuentran fuertemente influenciados por el modelo matemático subyacente y por tanto, existe la necesidad de conocer dichos modelos. Finalmente, una vez sentadas las bases, se presentan los MFOOs más representativos en la literatura. La figura 1 muestra una descomposición de temas por niveles.

Nota: para precisar la localización de un tema en una determinada referencia se anexa a la referencia el rango de capítulos donde se encuentra. Por ejemplo, [Men87:1-2,7] significa referencia Men87, capítulos 1, 2 y 7.

3.1 Conceptos Básicos.

Los conceptos básicos no sólo incluyen fundamentos matemáticos y sobre orientación a objetos sino también fundamentos sobre procesos de producción de software. Un déficit detectado en la bibliografía es la falta de guías para aplicar MFOOs. Muchas referencias sobre métodos formales se puede catalogar como referencias descriptivas (definen lenguajes) escasamente constructivas (no definen la forma de utilizarlo para producir software). Tal información es necesaria para contextualizar adecuadamente los MFOOs.

- Entre los *fundamentos matemáticos* básicos se encuentran las nociones de lenguaje formal, sintaxis, semántica, modelo, satisfacción, teoría y prueba. La aproximación a tales temas debería ser genérica para evitar una presentación sesgada hacia determinado formalismos [Men87:1-2], [LeP81:1-2], [Duf91:1-2], [MaW85:1-2], [Wir90]. A partir de estos conceptos, se debe introducir los conceptos de especificación formal vs. informal, validez, completitud y nociones sobre computabilidad y expresividad.

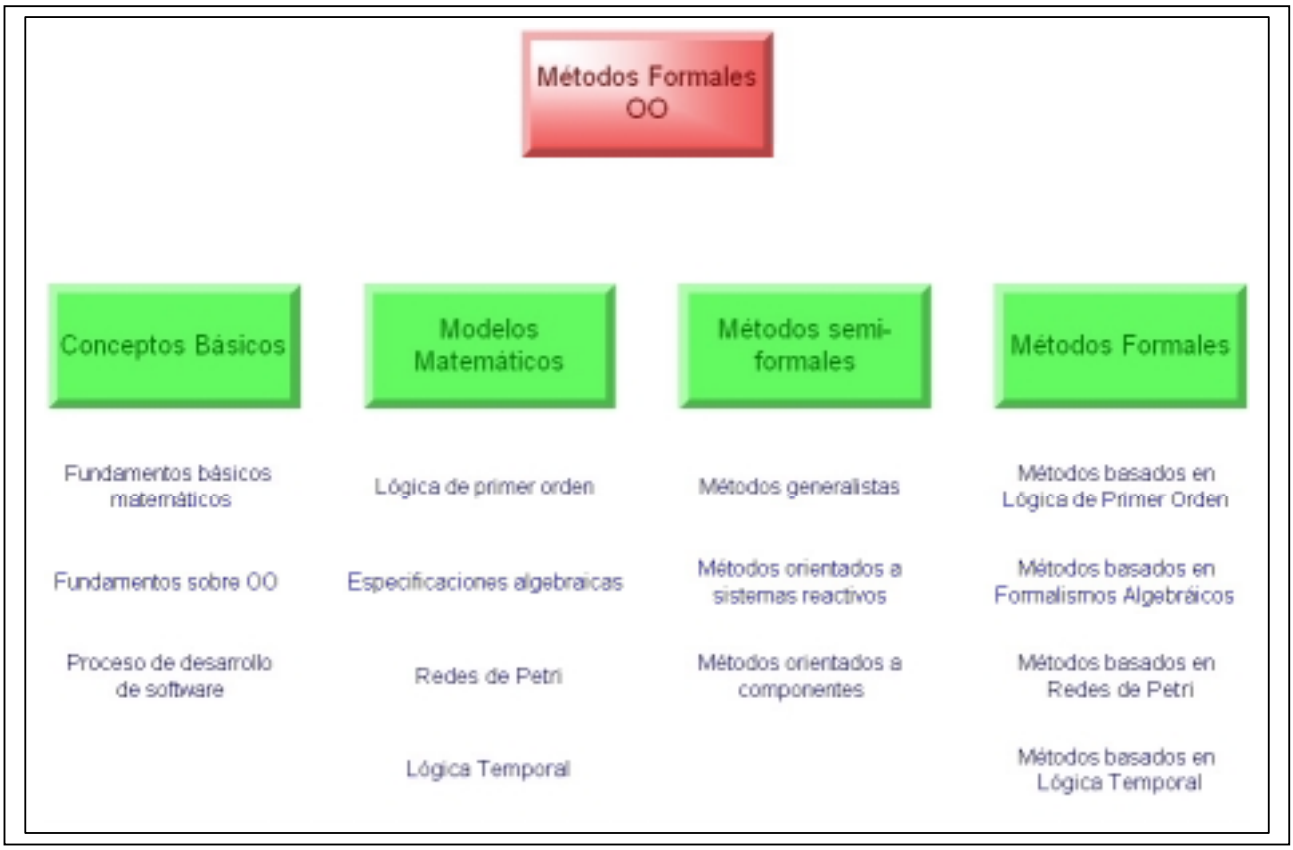

Fig. 1 Descomposición de temas por niveles

- Entre los *fundamentos sobre orientación a objetos* básicos destaca el concepto de concurrencia. Diferentes modelos de concurrencia (entrelazado y ejecución solapada) y conceptos como interferencia, justicia y sincronización son fundamentales para entender la evolución de sistemas OO en el tiempo [MaP91].
- Los MFOOs no se deben considerar de forma aislada al *proceso de desarrollo de software*. Un conocimiento básico de un proceso resulta fundamental para contextualizar adecuadamente el uso y resultados esperables para un MFOO [GJM91:1-3,5], [RBP+91:1,2], [Mey97], [CoD94:1].

3.2 Modelos Matemáticos.

Una vez establecidos los conceptos básicos, se establecen los distintos formalismos matemáticos subyacentes al MFOO.

- *Lógica de primer orden*. Es necesario establecer la definición del lenguaje utilizado (lógica de predicados de primer orden) y establecer la definición de prueba haciendo uso de diversos sistemas de deducción tales como sistemas natural y basado en secuentes. Las reglas de inferencia utilizadas, la forma de construir una prueba [Duf91:2-3], [Dah92:2] y el estudio de teorías con igualdad y con inducción representan aspectos claves [MaW90], [Men87:3-4]. Otros temas de interés son decibilidad y computabilidad efectiva. Tales temas son necesarios para evaluar el grado de automatización de las deducciones [MaW90:13], [Men87:5], [Man74:1].
- *Especificaciones Algebraicas*. Las nociones de Álgebra y Especificación ecuacional son necesarias para introducir el concepto de especificación algebraica de tipo abstracto de dato. El significado de una especificación se establece en base a tres tipos de semánticas: inicial, final y laxa [Wir90]. El cálculo ecuacional (pruebas mediante ecuaciones) y los sistemas de reescrituras representan el soporte para la deducción [EhM85:5], [Duf91:7-8]. Para especificar software a gran escala es necesario el estudio de mecanismos de extensión, parametrización y composición de especificaciones [Wir90], [EhM85:6-8], [EhM90].
- *Redes de Petri*. Existe una amplia bibliografía sobre la adecuación de dicho formalismo para expresar concurrencia. Dos caracterizaciones semánticas destacan en la literatura: entrelazado y concurrencia real. En la primera, dos procesos paralelos no ejecutan sus instrucciones en el mismo instante de tiempo mientras que en la segunda sí existe esa posibilidad. Dichas caracterizaciones son fundamentales para representar el concepto de estado y transición en un sistema. Finalmente se debe estudiar el problema de la justicia en las redes de Petri [MaP91:1], [Rei91].
- *Lógica Temporal*. Se debe establecer una clasificación inicial de los diferentes sistemas de lógica temporal basada en distintos criterios: proposicional vs. primer orden, tiempo lineal vs. tiempo ramificado [Lam83], [Eme89], evaluación instantánea o por intervalos [SMV83] y tiempo discreto vs. tiempo continuo (ej. Lógica de tiempo real (RTL)) . Una vez establecida la formalización del tiempo [AEF90:1-3], [MaP91:3] se procede a estudiar determinados formalismos. Entre los formalismos lineales [AEF90:4-7] (en cada momento, hay un solo posible futuro) cabe destacar Lógica Temporal Lineal Proposicional (PLTL) y Lógica Temporal Lineal de Primer Orden (FOLTL) [Eme90]. Entre los formalismos ramificados [AEF90:8-9] (cada momento puede tener diferentes posibles futuros) destacan la Lógica Temporal Ramificada Proposicional (CTL y CTL*) y Lógica Temporal Ramificada de Primer Orden [Eme90]. Una vez establecidos los marcos formales de interés se debe introducir el concepto formal de concurrencia. Modelo basado en entrelazado frente a concurrencia real [MaP91:2]. Posteriormente, se abordará la aplicación de la lógica temporal a la verificación [Pnu86] e incluso síntesis de programas concurrentes [AEF90:10-11].

3.3 Métodos Semi-formales OO.

Un conocimiento de sistema resulta necesario para contextualizar adecuadamente los MFOOs. Los denominados métodos semi-formales representan una aproximación muy interesante para la correcta comprensión de los métodos formales porque representan "transiciones suaves" hacia los métodos formales.

Métodos Generalistas. Dentro de los métodos de desarrollo de software generalistas destaca *Syntropy* [CoD94]. Sus principales aportaciones son: (a) el concepto de sistema, modelo y vista y, sobre todo, (b) la clarividencia a la hora de distinguir entre modelos del "mundo real", modelos de software y la relación entre ambos, (c) la capacidad de sopesar entre corrección y utilidad y (d) la integración entre notaciones formales como Z y semi-formales como los statecharts.

Métodos orientados al desarrollo de sistemas reactivos. Dentro de los métodos orientados al desarrollo de sistemas reactivos destaca *Statemate* [HaP98]. Aunque no se puede considerar como un método orientado a objetos de plena naturaleza, representa una excelente base para comprender y desarrollar este tipo de sistemas. Sus principales aportaciones son: (a) integración de lenguajes de modelado tales como diagramas de actividad, statecharts modulares, lenguaje natural, (b) concepción de sistema descrito a base de vistas y (c) base transformacional en sus producciones.

Métodos orientados al desarrollo de componentes. Dentro de los métodos orientados a desarrollo de componentes destaca *Catalysis* [SoW98]. Sus principales aportaciones son (a) modelado mediante integración de UML y OCL (lógica de primer orden orientada a objetos), (b) integración del uso de patrones de diseño en el método y (c) construcción de componentes desde especificaciones con distintos grados de abstracción.

3.4 Métodos Formales OO.

Es importante presentar los métodos formales clasificados por el formalismo matemático subyacente, por los criterios de orientación a objetos considerados y por la integración en un entorno de desarrollo de software con uso práctico y con existencia o no de herramientas. De esta forma, el lector

interesado en el tema podría evaluar de forma efectiva qué método elegir según sus necesidades.

- El formalismo matemático subyacente condiciona en gran medida al MFOO. Por ello, se propone una primera clasificación de los MFOOs basada en el formalismo matemático. Se distinguen cuatro catgorías: MFOOs basados en Lógica de Primer Orden, MFOOs basados en Álgebras, MFOOs basados en Redes de Petri y MFOOs basados en Lógica Temporal. Una vez realizada la primera clasificación se mide el nivel de orientación a objeto de tales métodos. Los criterios OO se dividen en dos categorías: criterios basados en objeto y criterios orientados a objeto. Los criterios basados en objeto suministran principalmente encapsulación y métodos para acceder al estado de una forma controlada. Los criterios orientados a objetos suministran capacidad de herencia, tipado/subtipado y criterios de genericidad/parametrización. Se han seleccionado un conjunto representativo (no completo) de MFOOs presentes en la literatura. Una vez clasificados los MFOOs, se ha establecido una breve caracterización para cada uno de ellos mediante el uso de tablas. En dichas tablas, la 1ª columna indica el criterio o característica estudiado y la 2ª columna indica si tal característica se encuentra en lenguaje (valor SI o NO respectivamente). También se aceptan los valores ¿SI? y ¿NO? si la característica está presente parcialmente y ¿? para situaciones inciertas. La 3ª columna permite aclaraciones y comentarios.

3.4.1 MFOOs basados en Lógica de Primer Orden.

Los métodos formales basados en Lógica de Primer Orden más conocidos son Z y VDM. Estos lenguajes no presentan características orientadas a objetos tales como identidad de objetos, encapsulación, herencia, creación de objetos, agregaciones, colecciones, etc. Varias extensiones orientadas a objetos se han propuesto: Object-Z [DKR+91], Z++ [LH94b] y VDM++ [DvK92].

- ***Object-Z.*** Se trata de una extensión de Z [Spi92] con conceptos OO. Una especificación Z define un sistema software mediante esquemas estado y esquemas operación. En Object-Z, una operación se refiere sólo al estado del objeto al que pertenece. Una clase se obtiene mediante la definición de estado y operaciones asociadas. Sintácticamente, una clase es una (típica) caja Z con nombre y (opcionalmente) parámetros. Los posibles constituyentes de una clase son: una lista de visibilidad, clases heredadas, definición de tipo y constantes, un esquema de estado, un esquema (de estado) inicial, esquema de operaciones y un invariante de historia. Las definiciones de constantes y tipos son como en Z. El esquema estado no tiene nombre y contiene declaraciones de variables estado y predicado estado. El predicado estado representa el invariante de la clase y está implícitamente incluido en todo esquema operación y esquema inicial. El esquema inicial puede no ser único. Los esquemas operación describen los métodos definidos por la clase y contiene una lista de variables estado cuyos valores pueden cambiar cuando la operación se aplica a un objeto de la clase. El invariante de historia es un predicado sobre las historias de los objetos restringiendo su comportamiento. (ej. restricción de justicia). Tales invariantes se establecen mediante lógica temporal. Una declaración del tipo *c:C* declara *c* como referencia a un objeto de la clase *C*. Una declaración no implica ni la creación del objeto ni la inicialización del mismo.

Tabla 1: Criterios basados en objetos en Object-Z

Criterio basado en Objeto	**Presencia**	**Comentario**
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	Llamada a método
Interacción Asíncrona	NO	No hay mensajes
Abstracción de Datos	SI	Herencia de Z
Identidad	SI	Tipo referencia
Intra-concurrencia	SI	Especificación de historias en términos de lógica temporal
Inter-concurrencia	SI	

Tabla 2: Criterios orientados a objetos en Object-Z

Criterio Orientado a Objeto	**Presencia**	**Comentario**
Clase intensional	SI	Ver [LH94a]
Clase extensional	NO	
Herencia	SI	
Subtipado	SI	
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	SI	
Clases como objetos	NO	
Colección de objetos	¿SI?	Sólo colecciones de referencias
Instanciación dinámica/estática	NO	Colecciones de referencias usadas con tal propósito
Genericidad	SI	Parámetros tipo

Tabla 3: Grado de formalización en Object-Z

Semántica	Presencia	Comentarios
Semántica	SI	Ver [DKRS91], [Smi95]
Cálculo	SI	Ver [Smi92], [Smi95-1], [Smi95-2]

La inicialización de los objetos se consigue mediante el esquema (predefinido) *init*. No existe creación de objetos explícita en Object-Z, si este concepto fuera necesario en una especificación entonces se debería modelar mediante conjuntos de objetos existentes.

Los objetos pueden tener referencias como atributos. Una operación es una relación entre dos estados sucesivos en la historia del objeto. Conjunciones, disyunciones y otras operaciones entre esquemas permiten construir nuevas operaciones desde otra previamente construidas. Existen los operadores [] (elección no determinista de una operación), ‖ (composición paralela con intercomunicación entre objetos). Las siguientes tablas se ha confeccionado con la información contenida en las referencias: [LH94a], [DD90], [DKRS91], [DKS91], [Smi92] y [Smi00].

- ***VDM++.*** Es un lenguaje de especificación basado en VDM-SL [Jon90]. Una especificación VDM++ consta de un conjunto de definiciones de clases. Una definición de clase representa una plantilla para objetos con atributos, operaciones (métodos) y propiedades sobre concurrencia y tiempo real. Una clase VDM++ consta de varias cláusulas para definir constantes, variables, invariantes, métodos, tareas y sincronizaciones. Los tipos, constantes y métodos se heredan de VDM-SL. Un invariante restringe los valores iniciales de las variables y los valores de las variables antes y después de la ejecución de un método. La clausula *init* restringe el estado inicial (puede no ser único). Un método puede definirse de varias maneras: preliminar (será definido en una futura etapa del desarrollo), diferido (definido en una subclase), declarativo (mediante pre-post condiciones) y procedural (código). Exiten tipos referencias. Si *C* es una clase en un sistema entonces *@C* representa las referencias a los objetos en un instante de tiempo. Hay un operador "new" para crear nuevas referencias en el sistema. Una clase puede ser declarada como subclase de otra. Toda característica de la clase se convierte en característica de la subclase. Adicionalmente, se pueden añadir nuevos atributos, invariantes, métodos, etc. Se admiten redefiniciones consistentes de los métodos. Las agregaciones se crean con un número fijo de instancias y todos sus elementos dependen en existencia del agregado. Los elementos de un agregado, por separado, ni se destruyen ni comparten. Una cláusula *sync* en una clase permite especificar sincronizaciones y secuencias de invocaciones permitidas sobre los métodos. Las siguientes tablas se han confeccionado con la información contenida en las referencias: [DvK92], [LH94a] y [Gro96]:

Tabla 4: Criterios basados en objetos en VDM++

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	Llamada a método
Interacción Asíncrona	NO	No hay mensajes
Abstracción de Datos	SI	Herencia de VDM
Identidad	SI	Tipo referencia
Intra-concurrencia	SI	Trazas de ejecución con entrelazado
Inter-concurrencia	SI	

Tabla 5: Criterios orientados a objetos en VDM++

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	SI	
Clase extensional	NO	
Herencia	SI	
Subtipado	¿SI?	Mediante tipos unión
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	SI	
Clases como objetos	NO	
Colección de objetos	¿SI?	Colección de referencias
Instanciación dinámica/estática	¿SI?	Objetos creados con NEW
Genericidad	NO	

Tabla 6: Grado de formalización en VDM++

Semántica	Presencia	Comentarios
Semántica	¿SI?	Sólo semántica estática establecida informalmente. Ver [Gro96]
Cálculo	¿NO?	

- ***Z++.*** Se trata de un lenguaje basado en Z [Spi92].Se propuso como lenguaje de diseño, además de poseer elementos para estructurar incrementalmente desarrollos a gran escala. El concepto de refinamiento es esencial

en Z++. Clase, tipo y teoría representan el mismo concepto (igualmente subclase, subtipo y refinamiento). Objeto, elemento y modelo también son conceptos coincidentes. Sintácticamente, una clase consta de nombre, parámetros opcionales de tipos y un conjunto de cláusulas que describen las siguientes características: extensión desde otras clases previamente definidas, definición de tipos locales a una clase, variables locales representando atributos al estilo Z, operaciones sin efectos laterales, invariante sobre el estado interno, operaciones con efectos laterales (acciones) que se pueden especificar como predicados Z o como código B0 [Abr96]. Las acciones pueden cualificarse como espontáneas (demonios) y un predicado historia describirá las ejecuciones admisibles para los objetos de la clase. Los predicados de historia se describen en lógica temporal lineal (LTL) o lógica temporal real (RTL). En vez de utilizar la notación de cajas de Z se utiliza una notación ASCII similar a la de un lenguaje de programación orientado a objeto. Se admite la parametrización de tipo, sin embargo una clase genérica no es un tipo en sí mismo, debe ser instanciada para ser considerada como tipo. Para cada clase *C* hay un tipo *@C* de referencias de objetos de dicha clase. Se admiten métodos de clase. Z++ permite varias operaciones y relaciones sobre clases. Hay operaciones para ocultar y renombrar características de una clase, instanciar clases genéricas, unión, conjunción, refinamiento, equivalencia y herencia entre clases. En [LH94b] se encuentra definida una semántica para Z++ y en [LH94a] se establece una correspondencia de Z++ a Z por lo tanto, todo trabajo desarrollado para Z (ej. Cálculo Z) puede ser utilizado para especificaciones Z++. Las siguientes tablas se han confeccionado con la información contenida en las referencias: [LH94a] y [LH94b]:

Tabla 7: Criterios basados en objetos en Z++

Criterio basado en Objeto	**Presencia**	**Comentario**
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	Llamada a método
Interacción Asíncrona	NO	No hay mensajes
Abstracción de Datos	SI	Herencia de Z
Identidad	SI	Tipo referencia
Intra-concurrencia	SI	Historia especificada en LTL o RTL
Inter-concurrencia	SI	

Tabla 8: Criterios orientados a objetos en Z++

Criterio Orientado a Objeto	**Presencia**	**Comentario**
Clase intensional	SI	
Clase extensional	NO	
Herencia	SI	
Subtipado	SI	Mediante tipos unión
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	¿NO?	
Clases como objetos	NO	
Colección de objetos	SI	
Instanciación dinámica/estática	SI	
Genericidad	SI	Parámetros tipo

Tabla 9: Grado de formalización en Z++

Semántica	**Presencia**	**Comentarios**
Semántica	SI	Ver [LH94b]
Cálculo	¿SI?	Previa transformación de la especificación Z++ a Z

3.4.2 MFOOs basados en Formalismos Algebraicos.

Se ha desarrollado una amplia investigación con el propósito de extender las especificaciones algebraicas hacia la orientación a objetos. Se han elegido HOSA (Hidden Order Sorted Algebras), TROLL, Maude y AS-IS (Algebraic Specifications with Implicit States) como ejemplos representativos, sin embargo, no incluimos muchos otros interesantes trabajos como el lenguaje ABEL [Dah87] o Larch [CL94].

- ***HOSA.*** Fue desarollado por Goguen, Diaconescu y Meseguer [GD94] y [MG94] y se puede considerar más como un marco semántico para orientación a objeto (puede usarse a nivel de especificación y a nivel de programación) que como un lenguaje de especificación. Se basa en especificaciones algebraicas (lenguaje OBJ) donde algunos sorts (tipos) se cualifican como ocultos y otros como visibles. Los primeros se corresponden con los estados ocultos de los objetos y los segundos con los valores que pueden ser observados. Existe un orden parcial sobre los tipos (subtipado). La principal diferencia con las especificaciones algebraicas clásicas reside en la noción de satisfacción. Se basa en equivalencia de comportamiento, un tipo de

equivalencia observacional basada en los elementos visibles. Es decir, una ecuación $i = d$ en un tipo oculto se satisface sii para todo contexto C de tipo visible, $C(i) = C(r)$ se satisface. Es decir, lo que no es observable a través de la interfaz de un objeto no es relevante; esto hace que el desarrollo de pruebas en este tipo de sistemas formales sea más difícil que en otras "instituciones más clásicas". La noción de clase se formaliza mediante un *sort* (tipo). La herencia se simplifica mediante la importación de módulos con preservación de propiedades. Los objetos se describen añadiendo un nuevo parámetro (de identidad) a todas las operaciones. No hay creación ni eliminación de objetos. En conclusión, es más un trabajo teórico que un formalismo "listo para ser usado". Los conceptos de álgebras con sorts ocultos y satisfacción mediante comportamiento representan dos importantes conceptos semánticos. Las siguientes tablas se han confeccionado con la información contenida en las referencias: [GD94], [MG94], [Gog97] y [GoM00]:

Tabla 10: Criterios basados en objetos en HOSA

Criterio basado en Objeto	**Presencia**	**Comentario**
Objeto y estado	SI	Todos los objetos don especificaciones OBJ3 y los estados se definen mediante sorts ocultos
Encapsulación	SI	
Interacción Síncrona	NO	No relevante
Interacción Asíncrona	NO	No relevante
Abstracción de Datos	SI	Herencia de Z
Identidad	SI	Predefinida en OBJ3
Intra-concurrencia	NO	
Inter.-concurrencia	SI	

Tabla 11: Criterios orientados a objetos en HOSA

Criterio Orientado a Objeto	**Presencia**	**Comentario**
Clase intensional	SI	Es un sort
Clase extensional	NO	
Herencia	SI	Mediante importación de modulos
Subtipado	SI	"Sub-sorting"
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	SI	
Clases como objetos	NO	
Colección de objetos	SI	Como estructuras de datos
Instanciación dinámica/estática	NO	
Genericidad	SI	Parámetros tipo

Tabla 12: Grado de formalización en HOSA

Semántica	**Presencia**	**Comentarios**
Semántica	SI	
Cálculo	SI	

- ***TROLL.*** TROLL y OBLOG son dos versiones (textual y gráfica) del mismo lenguaje de especificación [JSH+91]. Los sistemas de información y bases de datos constituyen su dominio de aplicación. TROLL-light [CGH92] es un sublenguaje de TROLL completamente formalizado. La principal diferencia entre TROLL y TROLL-light es que no hay herencia en el segundo. Sin embargo, un tratamiento formal de la herencia aparece en [EGS93]. La originalidad de esta aproximación reside en distinguir entre plantillas de objetos (definición intensional de la clase) y clases de objetos (definición extensional). Esta visión hace posible considerar a las clases como objetos, introduciendo el concepto de metaclase. Hay una clase especial *INIT*, todas las otras clases son subobjetos de ella. Por otro lado, la herencia se modela con una relación entre plantillas. Los objetos se describen como conjuntos de objeto-aspectos. Un objeto-aspecto posee identidad y una plantilla. Un objeto es un aspecto junto a todos los aspectos derivados por herencia (es decir, misma identidad más las plantillas heredadas). Los aspectos dinámicos se formalizan mediante eventos y procesos. Un evento es una mezcla de observaciones y acciones sobre un objeto en un instante de tiempo. Los procesos representan secuencias de eventos permitidas para el

objeto de una determinada plantilla. Las interacciones son síncronas La literatura muestra su aplicación a problemas relacionados sistemas de información y bases de datos, no está claro que sea adecuado para la especificación de sistemas de control en tiempo real. Hay otro dialecto de OBLOG, llamado GNOME [RS94], con un conjunto fijo de tipos básicos y con mayor énfasis en interacciones entre objetos y encapsulaciones mediante el concepto de región. Las siguientes tablas se han confeccionado con la información contenida en las referencias: [Con94], [CGH92], [EGS93] y [JSH+91]:

Tabla 13: Criterios basados en objetos en TROLL

Criterio basado en Objeto	**Presencia**	**Comentario**
Objeto y estado	SI	Objeto como varios aspectos y estado como atributos
Encapsulación	SI	
Interacción Síncrona	SI	Variables compartidas
Interacción Asíncrona	NO	
Abstracción de Datos	SI	
Identidad	SI	
Intra-concurrencia	SI	Mediante ‖
Inter.-concurrencia	SI	

Tabla 14: Criterios orientados a objetos en TROLL

Criterio Orientado a Objeto	**Presencia**	**Comentario**
Clase intensional	SI	
Clase extensional	SI	
Herencia	SI	Entre plantillas
Subtipado	NO	
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	NO	No relevante
Clases como objetos	SI	
Colección de objetos	SI	
Instanciación dinámica/estática	SI	
Genericidad	¿?	

Tabla 15: Grado de formalización en TROLL

Semántica	**Presencia**	**Comentarios**
Semántica	¿SI?	TROLL-light
Cálculo	SI	

- ***Maude.*** Desarrollado por Meseguer y sus colaboradores [Mes93], [MW92] como lenguaje para especificar y programar sistemas OO concurrentes. Maude extiende el paradigma de especificación algebraica e incluye como sub-lenguaje funcional (esencialmente) OBJ3 [GWM+93]. Maude hereda de OBJ3 sus mecanismos de estructuración modular, incluyendo varias formas de importación, parametrización (muy expresiva) y sobrecarga mediante renombrado. El modelo básico de concurrencia es reescritura concurrente. Se trata de una generalización de la reescritura clásica de términos. La reescritura concurrente se puede considerar como una formalización de los sistemas de transición de estados concurrentes, con estados descritos mediante términos de una estructura algebraica y reglas expresando posibles transiciones de estados. Un objeto Maude se representa mediante una estructura registro con nombre de objeto, nombre de clase y una lista de atributos (pares de nombre atributo-valor). Los mensajes se definen mediante constructores que toman uno o más objetos entre sus argumentos. Una configuración se define como multiconjunto de mensajes y objetos. Las reglas de reescrituras especifica como configuraciones pueden transformarse mediante procesamiento de mensajes. En su forma más general, la aplicación de una regla transforma los mensajes y objetos del lado izquierdo de la regla mediante absorción de mensaje, modificación de atributos del objeto, eliminación de objetos, creación de nuevos objetos o producción de nuevos mensajes. Las reglas se pueden aplicar concurrentemente a diferentes partes de una configuración. Maude permite expresar un comportamiento bastante complejo, incluyendo creación y eliminación dinámica de objetos y comportamiento síncrono o asíncrono. El formalismo matemático subyacente es la lógica de reescritura [Mes92] con reescritura de términos módulo asociatividad, conmutatividad e identidad. Los términos representan proposiciones sobre el sistema, y reescritura representa derivación o deducción. De esta manera, computación y deducción coinciden como en otros sistemas. Maude distingue dos tipos de herencia. Herencia de clase (atributos, mensajes y reglas) y herencia de módulo (reutilización de código). La reescritura concurrente representa un modelo de concurrencia muy general. De hecho, una gran variedad de modelos de concurrencia pueden ser descritos con Maude (Redes de Petri, Actores y CCS por ejemplo). En su máxima generalidad, Maude es un lenguaje de especificación, sin embargo, una variante denominada Simple Maude puede utilizarse de forma efectiva porque sus reglas son terminantes y confluentes y sus lados izquierdos sólo

poseen un objeto y, como máximo, un mensaje. Las siguientes tablas se han confeccionado con la información contenida en las referencias: [Mes93], [MW92] y [Mau99]:

Tabla 16: Criterios basados en objetos en Maude

Criterio basado en Objeto	**Presencia**	**Comentario**
Objeto y estado	SI	Objeto como registros de atributos-valores
Encapsulación	¿SI?	Los atributos se pueden ocultar pero las reglas no
Interacción Síncrona	SI	
Interacción Asíncrona	SI	
Abstracción de Datos	SI	
Identidad	SI	
Intra-concurrencia	SI	
Inter.-concurrencia	SI	

Tabla 17: Criterios orientados a objetos en Maude

Criterio Orientado a Objeto	**Presencia**	**Comentario**
Clase intensional	SI	Se corresponde con sorts
Clase extensional	¿SI?	Se debe programar
Herencia	SI	
Subtipado	SI	"Subsorting"
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	SI	
Clases como objetos	NO	Se puede expresar mediante metaobjetos
Colección de objetos	SI	Se debe programar
Instanciación dinámica/estática	SI	
Genericidad	SI	

Tabla 18: Grado de formalización en Maude

Semántica	**Presencia**	**Comentarios**
Semántica	SI	
Cálculo	SI	

- ***AS-IS*** *(Especificaciones Algebraica con Estados Implícitos)*. AS-IS se utiliza para especificar sistemas cuyo comportamiento externo depende de un estado interno. Considera dos tipos de operaciones: los accesos que observan el estado implícito (interno) y los modificadores que se utilizan para cambiar el estado. Un estado es un álgebra, y un cambio de estado, una transformación definida sobre un álgebra [Zam97].

  Se diferencia de HOSA en la formalización del estado (no es un *sort*), está más cercano a la noción de estado de VDM o Z. Es diferente de TROLL porque no hay distinción entre atributos y accesos previniendo de decisiones tempranas de implementación. Esta aproximación permite modelar sistemas con un estado centralizado. Se ha validado con especificaciones para sistemas de control complejos [GDK96]. La semántica del lenguaje se puede encontrar en [KGD97]. Los creadores de AS-IS dejaron pendiente la formalización de una extensión del lenguaje basada en la encapsulación de agregaciones de objetos. No conocemos de la finalización de dicho trabajo.

  AS-IS utiliza tipos de datos predefinidos, operaciones, accesos y modificadores para especificar clases y colecciones de objetos. Las clases se corresponden con plantillas. Las colecciones son especificaciones de sistemas multi-objetos. Son posibles la creación y eliminación de objetos, conexiones y desconexiones y paso de señales entre objetos conectados. Entre las características predefinidas en AS-IS se encuentran los nombres (identidades de objetos), predicados para determinar la existencia de objetos y conexiones y señales entre objetos conectados. La comunicación entre objetos es síncrona y no hay noción de herencia. La gestión de las identidades de objetos está influenciada por TROLL y por [Wie91]. El formalismo está orientado a la descripción de sistemas reactivos síncronos. Los modificadores se formalizan como ejecuciones idealmente atómicas y los conceptos conexión y señal están influenciados por el modelo BOSO [BV96].

Tabla 19: Criterios basados en objetos en AS-IS

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	
Interacción Asíncrona	NO	
Abstracción de Datos	SI	
Identidad	SI	
Intra-concurrencia	SI	simultaneidad
Inter.-concurrencia	SI	simultaneidad

Tabla 20: Criterios orientados a objetos en AS-IS

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	SI	Clases no son tipos
Clase extensional	NO	Se debe programar
Herencia	NO	
Subtipado	SI	"Subsorting"
Múltiple Herencia/Subtipado	NO	
Herencia≠Subtipado	NO	No relevante
Clases como objetos	NO	
Colección de objetos	SI	
Instanciación dinámica/estática	SI	
Genericidad	NO	

Tabla 21: Grado de formalización en AS-IS

Semántica	Presencia	Comentarios
Semántica	SI	
Cálculo	NO	

3.4.3 MFOOs basados en Redes de Petri.

Las redes de Petri han sido criticadas por adolecer de mecanismos de estructuración. Se han propuesto distintas soluciones [JR91], [Kie89]. La formalización de aspectos orientados a objetos en las redes de Petri siguen dos aproximaciones: (a) una clase se formaliza con una red y los objetos que circulan por la red son marcas y (b) cada objeto se formaliza con una red y los lugares de la red juegan el papel de atributos. En ambas aproximaciones los métodos se corresponden con transiciones.

- ***CO-OPN/2*** [BBG97] (Concurrent Object-Oriented Petri Nets) es un lenguaje de especificación para modelar sistemas concurrentes a gran escala. Se fundamenta en especificaciones algebraicas y redes de Petri. Las especificaciones algebraicas se necesitan para formalizar los aspectos funcionales y las redes de Petri modelar el comportamiento concurrente. Un objeto es una entidad autónoma compuesto de estado interno y servicios. La única forma de interaccionar con un objeto es a través de sus servicios. CO-OPN/2 define un objeto como una red algebraica encapsulada con lugares que representan el estado interno y transiciones que representan eventos. Las transiciones son de dos tipos: transiciones parametrizadas (métodos que representan servicios ofertados al exterior) y transiciones internas (representan comportamiento interno no visibles desde el exterior). Las interacciones entre objetos son síncronas. Cada objeto posee su propio proceso y evoluciona concurrentemente con el resto de objetos del sistema. Cada objeto tiene identidad. A cada clase se le asocia un tipo. El lenguaje distingue entre herencia y subtipado [BB94], [BBG97] y [BBG01].

Tabla 22: Criterios basados en objetos en CO-OPN/2

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	Expresiones de sincronización
Interacción Asíncrona	NO	
Abstracción de Datos	SI	ADT algebraicos
Identidad	SI	
Intra-concurrencia	SI	
Inter.-concurrencia	SI	

Tabla 23: Criterios orientados a objetos en CO-OPN/2

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	SI	
Clase extensional	NO	
Herencia	SI	Sintáctica
Subtipado	SI	Semántico
Múltiple Herencia/Subtipado	SI/NO	
Herencia≠Subtipado	SI	
Clases como objetos	NO	
Colección de objetos	NO	
Instanciación dinámica/estática	SI	
Genericidad	SI	

Tabla 24: Grado de formalización en CO-OPN/2

Semántica	Presencia	Comentarios
Semántica	SI	
Cálculo	SI	Válido sólo para modelos basados en algebra inicial para los ADTs.

3.4.4 MFOOs basados en Lógica Temporal.

Las formalizaciones basadas en lógica temporal son adecuadas para describir sistemas reactivos y concurrentes. Un aspecto común asociado con las diferentes lógicas temporales es la noción de tiempo-estado. Una especificación escrita en términos de lógica temporal describe las secuencias de estados admisibles para el sistema de forma declarativa. Los lenguajes propuestos provienen de lenguajes preexistentes con la adición de características orientadas a objetos tales como encapsulación de estado, definición de interfaces (atributos, métodos) y operadores de composición.

- ***TRIO+.*** Es un lenguaje para especificar sistemas empotrados y de tiempo real [MP94]. Está basado en el lenguaje TRIO [MMG92]. TRIO es un lenguaje lógico temporal de primer orden ideado para construir especificaciones ejecutables de sistemas en tiempo real. Modela el tiempo de forma cuantitativa. El modelo de tiempo está parametrizado permitiendo tanto estructuras de tiempo (finitas) discretas como estructuras de tiempo (finitas) densas. TRIO proporciona soporte para diferentes actividades de validación como testar especificaciones, simulación y pruebas. En [FM94] podemos encontrar una axiomatización de TRIO. Los autores proponen la ejecutabilidad de las especificaciones como medio de asegurar la adecuación de los requisitos. Sobre TRIO se han desarrollado un conjunto de herramientas, destacan un conjunto de herramientas generadoras de tests. TRIO+ se diseño para construir especificaciones de sistemas complejos de forma sistemática y modular. El universo de objetos se divide en clases. Una especificación TRIO+ consta de un conjunto de definiciones de clases. Cada clase es un conjunto de axiomas. Las clases pueden ser simples o estructuradas. Las clases simples contienen una interfaz (métodos), un dominio temporal (real racional, entero), declaración de predicados dependientes del tiempo, variables, funciones, colección de axiomas (TRIO formulas). Las clases complejas son clases con módulos. La semántica de una clase compleja se establece mediante clases simples equivalentes sin los mecanismos de estructuración utilizados. Es importante destacar la utilización de TRIO+ en la industria (ENEL Italian Electric and Energy Board).

Tabla 25: Criterios basados en objetos en TRIO+

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	Toda entidad posee un estado. Pueden existir conexiones entre objetos.
Encapsulación	SI	Mediante interfaces
Interacción Síncrona	SI	
Interacción Asíncrona	NO	
Abstracción de Datos	NO	Solo tipos básicos predefinidos
Identidad	SI	
Intra-concurrencia	SI	
Inter.-concurrencia	SI	Entre objetos conectados

Tabla 26 Criterios orientados a objetos en TRIO+

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	SI	
Clase extensional	SI	Universo de objetos dividido en clases
Herencia	SI	
Subtipado	NO	
Múltiple Herencia/Subtipado	SI/NO	
Herencia≠Subtipado	NO	
Clases como objetos	NO	No considerados en TRIO
Colección de objetos	SI	
Instanciación dinámica/estática	NO/SI	
Genericidad	SI	Sólo en tipos simples

Tabla 27: Grado de formalización en TRIO+

Semántica	Presencia	Comentarios
Semántica	SI	Ver [MP94]
Cálculo	SI	No hay un cálculo explícito aunque si una axiomatización

- ***OO-LTL.*** Es un lenguaje lógico temporal con estructura de tiempo lineal y discreta (TLA [Lam94]). Emplea el concepto de módulo para el desarrollo formal de sistemas concurrentes, incluyendo agregación y parametrización. Sin embargo, OO-LTL es una aproximación híbrida, es decir, existen elementos en el lenguaje que no se consideran objetos y otros elementos del lenguaje sí. Los objetos encapsulan estados, suministran interfaces (atributos y operaciones) y poseen actividad propia. El estado es observable sólo a través de los atributos y puede ser modificado mediante la activación de las operaciones. El estado también puede cambiar como resultado de la propia actividad del objeto, este comportamiento se expresa mediante una fórmula TLA. OO-LTL permite modelar colecciones (estáticas) de objetos. Los objetos complejos se pueden refinar en términos de objetos interactuando entre sí mediante los servicios que cada uno proporciona y comunicándose a través de objetos compartidos y argumentos comunes en sus parámetros. Las siguientes tablas se han construido desde la información contenida en [CvH97]:

Tabla 28: Criterios basados en objetos en OO-LTL

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	Llamadas a métodos
Interacción Asíncrona	NO	
Abstracción de Datos	SI	ADT primer orden
Identidad	SI	Restringida in TLA fórmulas
Intra-concurrencia	SI	
Inter.-concurrencia	SI	

Tabla 29 Criterios orientados a objetos en OO-LTL

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	SI	
Clase extensional	NO	
Herencia	¿SI?	No directamente soportado por el lenguaje. Ciertos mecanismos de reutilización pueden interpretarse como herencia
Subtipado	SI	
Múltiple Herencia/Subtipado	SI	
Herencia≠Subtipado	SI	
Clases como objetos	NO	
Colección de objetos	SI	
Instanciación dinámica/estática	NO/SI	
Genericidad	SI	Objetos parametrizables con tipos, predicados, funciones y objetos

Tabla 30: Grado de formalización en OO-LTL

Semántica	Presencia	Comentarios
Semántica	SI	En términos de TLA
Cálculo	SI	Cálculo para TLA

- ***ATOM.*** Se trata de un método formal basado en objetos para el desarrollo de sistemas en tiempo real. El método integra técnicas formales TAM (Temporal Agent Model) con metodología HRT-HOOD. ATOM es una aproximación formal basada en refinamientos. Una especificación ATOM contiene una descripción del tiempo mediante lógical temporal de intervalos (ITL), descripción de funciones y descripción de las comunicaciones del sistema. El sistema es un conjunto de actividades concurrentes encapsuladas en objetos mediante threads y métodos. Se distinguen cinco tipos de objetos según su actividad (esporádico, cíclico, pasivo-protegido, pasivo, activo o subsistema) La especificación formal se analiza y refina mediante un cálculo. La semántica de ATOM se ha establecido denotacionalmente usando ITL. A diferencia de los anteriores, se trata de un método formal que hace uso guías explícitas para construir especificaciones formales. Las siguientes tablas se han construido desde la información contenida en [ZCC99]:

Tabla 31: Criterios basados en objetos en ATOM

Criterio basado en Objeto	Presencia	Comentario
Objeto y estado	SI	
Encapsulación	SI	
Interacción Síncrona	SI	
Interacción Asíncrona	SI	
Abstracción de Datos	SI	
Identidad	SI	
Intra-concurrencia	SI	
Inter.-concurrencia	SI	

Tabla 32 Criterios orientados a objetos en ATOM

Criterio Orientado a Objeto	Presencia	Comentario
Clase intensional	NO	
Clase extensional	NO	
Herencia	NO	
Subtipado	NO	
Múltiple Herencia/Subtipado	NO	
Herencia≠Subtipado	NO	No aplicable
Clases como objetos	NO	
Colección de objetos	NO	
Instanciación dinámica/estática	SI/SI	
Genericidad	NO	

Tabla 33: Grado de formalización en ATOM

Semántica	Presencia	Comentarios
Semántica	SI	
Cálculo	SI	

4. MATRIZ TEMAS-REFERENCIAS

Existe un consenso general en admitir que los métodos formales orientados a objetos todavía no constituyen un área de conocimiento madura. Actualmente existe mucha investigación en curso. Este hecho queda reflejado en las referencias aportadas: la mayoría son artículos en revistas y congresos, informes técnicos relativos a resultados parciales de proyectos de investigación y escasas son las referencias bibliográficas. Considerando este hecho, se ha establecido la siguiente tabla relacionando los temas propuestos con las principales referencias. Las claves utilizadas son las siguientes: una "*" representa documento completo de interés. Un número indica una sección concreta de artículo o capítulo de libro. La expresión n1-n2 indica una secuencia de secciones o capítulos. El símbolo "+" representa nivel de dificultad considerable por contener conocimiento especializado. La ausencia de tales símbolos representa conocimiento no especializado (para un ingeniero (técnico/superior) informático).

Tabla 34. Matriz Conceptos Básicos-Referencias

	Fundamentos Matemáticos	Fundamentos Orientación Objetos	Fundamentos Proceso Desarrollo Software
[CoD94]			1
[Duf91]	1-2		
[GJM91]			1-3,5
[Men87]	1-2		
[Mey97]		1-18	
[RBP+91]			1-2

Tabla 35. Matriz Modelos Matemáticos-Referencias

	Lógica Primer Orden	Especificaciones Algebraicas	Redes Petri	Lógica Temporal
[AEF90]				1-9+
[Duf91]	2-3	7-8		
[EhM85]		5-8+		
[MaP91]			1+	3+

Tabla 36. Matriz Modelo Matemático-Métodos Formales OO

	Lógica Primer Orden	Especificacion Algebr.	Redes Petri	Lógica Temporal
Object-Z	[Smi00]*+			
VDM++	[DvK92]*+			
Z++	[LH94b]*+			
HOSA		[GD94]*+		
TROLL		[JSH+91]*+		
Maude		[Mes93]*+		
AS-IS		[KGD97]*+		
CO-OPN/2			[BBG97]*+	
TRIO+				[MP94+]*+
OO-LTL				[CvH97]*+
ATOM				[ZCC99]*+

Empiezan a existir recursos bibliográficos sobre lenguajes formales orientados a objetos publicados en la Web. Para los métodos formales propuestos en este trabajo se han encontrado las siguientes direcciones:

Tabla 37. Webs sobre métodos formales orientados a objetos

	Web
Object-Z	http://www.itee.uq.edu.au/~smith/objectz.html
VDM++	http://www.csr.ncl.ac.uk/vdm
Z++	
HOSA	http://www.cs.ucs.edu/users/goguen/projs/halg.html
TROLL	
Maude	
AS-IS	
CO-OPN/2	
TRIO+	
OO-LTL	
ATOM	http://www.cms.dmu.ac.uk/SRTL/

5. LECTURAS RECOMENDADAS

En esta sección presentamos una breve presentación de cada referencia recomendada. Intentamos que la lista sea mínima, suficientemente detallada e identificando su utilidad con respecto a los temas propuestos en este estudio.

[AEF90] E. Audereau, P. Enjalbert y L. Fariñas del Cerro. Logique Temporelle. Semantique et validation de programes paralleles. Masson, 1990.

Estudio detallado sobre formalismos temporales lineal y arborescente aplicados a la verificación de programas. La

presentación de conceptos teóricos y su aplicación a la verificación, incluso automática, de programas lo caracteriza como referencia fundamental para el lector interesado en algo más que introducciones.

[BBG97] Olivier Biberstein y Nicolas Guelfi. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. En G. Agha y F. De Cindio, eds. Advances in Petri Nets on Object-Orientation, LNCS 1997.

Definición del MFOO CO-OPN/2.

[CoD94] Steve Cook y John Daniels. Designing Object Systems: Object-Oriented Modelling with Syntropy. Prentice Hall, 1994

Se trata de un trabajo pionero en la integración de notaciones formales y semiformales dentro un proceso de desarrollo de software orientado a objetos. La capacidad de modelar sistemas a distintos niveles de abstracción y la organización de los modelos de software mediante vistas o aspectos parciales ha dado lugar a toda una filosofía de modelado de sistemas. El método propuesto no preconcibe ningún tipo de sistema (metodología generalista) por lo tanto sus ideas son transportables tanto a la construcción de sistemas de información como a la construcción de sistemas de control. El libro se puede tomar como referencia de proceso de desarrollo de software orientado a objetos y como referencia de método semiformal.

[CvH97] E. Canver y F. W. Von Henke. Formal specification and verification of object-based systems in temporal logic setting. Informe técnico proyecto Esprit 20072 DEVA (Design for Validation) Univerisity of Newcastle, 1997.

Definición del MFOO OO-LTL

[Duf91] David Duffy. Principles of Automated Theorem Proving. Wiley Professional Computing, 1991.

Los modelos matemáticos subyacentes en MFOOs basados en lógica de primer orden y especificaciones algebraica se presentan de una manera precisa y simple, facilitando la compresión del lector escasamente iniciado en el tema.

[DvK92] Dürr, E. H. y van Katwijk, J. VDM++. A Formal Specification Language for Object-Oriented Design. In Computer Systems and Software Engineering, pp. 214-219. IEEE Computer Society Press, 1992. Proceeding of CompEuro'92.

Definición del MFOO VDM++.

[GD94] Goguen, J. y Diaconescu, R. Towards an Algebraic Semantics for the Object Paradigm. In RECENT trends in data type specification: workshop on specification of abstract data types: COMPASS: selected papers, number 785 in LNCS, Springer-Verlag, 1996.

Definición del MFOO HOSA

[JSH+91] Jungclaus, R., Saake, G. Hartmann, T. y Sernadas, C. Object-oriented specification of information systems: The Troll language. 91-4, Informatik Bericht, Technische Universitat Braunschweig, 1991.

Definición del MFOO TROLL

[KGD97] C. Khoury, M.C. Gaudel y P. Dauchy. As-is. Technical report 1119, Laboratoire de Recherche en Informatique, 1997.

Definición del MFOO AS-IS

[MaP91] Z. Manna y A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems.

Se trata de una referencia importante para cubrir fundamentos matemáticos de MFOOs basados en Pedes de Petri y Lógica Temporal a un nivel de complejidad intermedio. Se supone que el lector debe tener conocimiento previo sobre algún sistema formal (Lógica de Primer Orden preferiblemente). En su contenido destaca la formalización (genérica) de sistema reactivo permitiendo tratar de manera uniforme muchos mecanismos de comunicación y coordinación presentes en recursos de implementación. También destaca el establecimiento de modelos para la concurrencia (real o simulada) y la formalización de propiedades exigibles a sistemas reactivos mediante lógica temporal.

[Mes93] Meseguer, J. A Logical Theory of Concurrent Objects ans its realization in the Maude language. En G. Agha, P. Wegner, y A. Yonezawa, eds. Research Directions in Concurrent Object-Oriented Programming, págs. 314-390. The MIT Press, 1993.

Definición del MFOO Maude.

[Mey97] B. Meyer. Object-Oriented Software Construction. 2nd ed. Prentice-Hall International series, 1997.

Se trata de una referencia básica en la construcción de software basado en la filosofía orientada a objetos. La presentación de los conceptos de forma abstracta y cómo tales conceptos se han realizado (parcialmente) en los lenguajes de programación prepara al lector entender los modelos abstractos de objetos propuestos en los diferentes MFOOs.

[MP94] A. Morzenti y P. San Pietro. Object-oriented logical specifications of time-critical systems. ACM Transactions on Software Engineering and Methodology, 3 pág 56-98, 1994.

Definición del MFOO TRIO+

[ZCC99] H. Zedan, A. Cau, Z. Chen, H. Yang. ATOM : An Object-Based Formal Method for Real-Time Systems. Informe Técnico en Software Technology Research Laboratory, SRE Centre, De Montfort University, The Gateway, Leicester http://www.cms.dmu.ac.uk/STRL/, 1999

Definición del MFOO ATOM.

[Smi00] G. Smith. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers 2000.

Definición del MFOO Object-Z. Resultado de la tesis doctoral del autor y de sus posteriores trabajos de investigación.

6. REFERENCIAS

En esta sección, se presentan otras referencias importantes. Muchas de ellas podrían competir y sustituir a las referencias recomendadas. No incluirlas se debe únicamente a la necesidad de satisfacer la restricción de mantener una bibliografía mínima.

[Abr96] Abrial, J. R. The B Book-Assigning Programs to Meanings. Cambridge University Press, 1996.

[AEF90] Auderau, E., Enjalbert, P. y Fariñas del Cerro, L. Logique Temporelle. Sémantique et validation de programmes parallèles.

[BB94] Olivier Biberstein y Didier Busch. An Object-Oriented Specification Language based on Hierarchical Algebraic Petri Nets. En R. Wieringa y R. Feenstra, eds. Working papers of the International Workshop on Information System Correctness and Reusability IS-CORE'94 pág 47-92, Amsterdam, Septiembre 1994.

[BBG97] Olivier Biberstein y Nicolas Guelfi. Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism. En G. Agha y F. De Cindio, eds. Advances in Petri Nets on Object-Orientation, LNCS 1997.

[BBG01] Olivier Biberstein, Didier Buchs and Nicolas Guelfi, Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 formalism, *Advances in Petri Nets on Object-Orientation*, G. Agha, F. De Cindio, y Grzegorz Rozenberg (Eds.), Lecture Notes in Computer Science, Springer-Verlag, Mayo 2001, ISBN 3-540-41942-X.

[Boo94] Booch, G. Object-Oriented Analysis and Design with Applications, 2nd ed. Benjamin/Cummings, 1994.

[BV96] F. Boulanger y G. Vidal-Naquet. Object execution model for reactive modules with a C++ implementation. En ECOOP'96 Workshop on Object and Real Time, 1996.

[CGH92] Conrad, S. Gogolla, M. y Herzig, R. Troll-light: a core language for specifyingobjects. 92-02, Technische Universitat Braunschweig, 1992.

[CL94] Cheon, Y. y Leavens, G. The Larch/Smalltalk interface specification language. ACM Transactions on Software Engineering and Methology, 3(3):221-253, 1994.

[CMJ98] D. Carrighton, I. MacCooll, J. McDonald, L. Murray y P. Strooper. From Object-Z Specifications to ClassBench Test Suites. Oct. 1998.

[CoD94] Steve Cook y John Daniels. Designing Object Systems: Object-Oriented modelling with Syntropy. Prentice-Hall, 1994.

[Con94] S. Conrad. On certification of Specification for troll-light objects. In RECENT trends in data type specification: workshop on specification of abstract data type. COMPASS: selected papers, LNCS 785 158-172, Springer-Verlag, 1994.

[CvH97] E. Canver y F. W. Von Henke. Formal specification and verification of object-based systems in temporal logic setting. Informe técnico proyecto Esprit 20072 DEVA (Design for Validation) Univerisity of Newcastle, 1997.

[Dah87] Dahl, O. Object-oriented specification. En Bruce Shriver y Peter Wegner eds. Research Directions in Object Oriented Programming, pág. 561-576. The MIT Press, Cambridge, 1987.

[Dah92] Ole-Johan Dahl. Verifiable Programming. Prentice Hall International Series, 1992.

[DD90] Duke, D. y Duke, R. Towards a Semantics for Object-Z. In VDM'90: VDM and Z! vol. 428 LNCS Springer-Verlag, 1990.

[DKR+91] Duke, R., King, P. Rose, G. A. y Smith, G. The Object-Z specification language. En T. Korson, V. Vaishnavi, y B. Meyer eds. Technology of Object-Oriented Languages and Systems: TOOLS 5, pp 465-483.

[DKS91] Duke, R., King, P. y Smith, G. Formalising Behavioural Compatibility for Reactive Object-Oriented Systems. In Proc. Of 14th Australian Computer Science Conference (ACSC-14), 1991.

[Duf91] David Duffy. Principles of Automated Theorem Proving. John Wiley & Sons, 1991.

[DvK92] Dürr, E. H. y van Katwijk, J. VDM++. A Formal Specification Language for Object-Oriented Design. In Computer Systems and Software Engineering, pp. 214-219. IEEE Computer Society Press, 1992. Proceeding of CompEuro'92.

[EGS93] Ehrich, H.D., Gogolla, M. y Sernadas, A. Objects and their specification. In RECENT trends in data type specification: workshop on specification of abstract data types joint with the 4th COMPASS. LNCS 655, Springer-Verlag, 1993.

[EhM85] Helmut Ehrig y Bernd Mahr. Fundamental of Algebraic Specification 1. Equations and Initial Semantics. Springer-Verlag 1985.

[EhM90] Helmut Ehrig y Bernd Mahr. Fundamental of Algebraic Specification 2. Formal Requirements and Modules. Springer-Verlag 1990.

[Eme89] Emerson E.A. y J.Y. Halpern. “Sometimes” and “Not Never” revisited: on Branching versus Linear Time Temporal Logic. J. ACM 33 (1)1986 151-178.

[Eme90] E. A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science. Elsevier Publishers B.V. 1990.

[FM] Formal Methods http://www.afm.sbu.ac.uk/

[FM94] M. Felder y A. Morzenti. Validating Real Time Systems by History-checking TRIO Specifications. ACM Transactions on Software Engineering and Methodology, 3 (4) 308-339, Oct 1994.

[GBD+97] Nicolas Guelfi, Olivier Biberstein, Didier Buchs, Ercüment Canver, Marie-Claude Gaudel, Friedrich von Henke y Detlef Schwier. Comparison of Object-Oriented Formal Methods. Technical Report of the Esprit Long Term Research Project 20072, 1997.

[GJM91] Carlo Ghezzi, Mehdi Jazayeri y Dino Mandrioli. Fundamentals of Software Engineering. Prentice Hall International, 1991.

[GD94] Goguen, J. y Diaconescu, R. Towards an Algebraic Semantics for the Object Paradigm. In RECENT trends in data type specification: workshop on specification of abstract data types: COMPASS: selected papers, number 785 in LNCS, Springer-Verlag, 1996.

[GDK96] M.C. Gaudel, P. Dauchy and C. Khoury. A Formal Specification of the Steam Boiler Control Problem by Algebraic Specifications with Implicit States. In H. Langmaack, J-R Abrial, E. Boeger eds. Formal Methods for Industrial Applications. LNCS, Speinger Verlag 1996.

[Gog97] J. A. Goguen. Hidden Algebraic Engineering. Dept of Computer Science. University of California at S. Diego. Informe Técnico CS97-569 Dec. 1997

[GoM00] A Hidden Agenda. Theoretical Computer Science vol 245, nº 1 pág 55-101.

[Gro96] The VDM Tool Group. The IFAD VDM++ Language. Odense, Denmark, 1996, http://www.ifad.dk/publications.htm

[GZ99] Marie-Claude Gaudel, A V. Zamulin: Algebraic Imperative Specifications. Ershov Memorial Conference 1999, pág. 17-39

[HaP98] Harel, D. y Politi M. The Statemate Approach. Modeling Reactive Systems with Statecharts. McGrawHill, 1998.

[Jon90] Jones, C.B. Systematic Software Construction Using VDM. Prentice Hall International Series, 1990.

[JR91] Kurt Jensen y Grzegorz Rozenberg, eds. High-level Petro Nets, Theory and Application. Springer-Verlag, 1991.

[JSH+91] Jungclaus, R., Saake, G. Hartmann, T. y Sernadas, C. Object-oriented specification of information systems: The Troll language. 91-4, Informatik Bericht, Technische Universitat Braunschweig, 1991.

[KGD97] C. Khoury, M.C. Gaudel y P. Dauchy. As-is. Technical report 1119, Laboratoire de Recherche en Informatique, 1997.

[Kie89] Astrid Kiehn. Petri net systems and their closure properties. En G. Rozenberg ed. Advance in Petri Nets, 424 LNCS, pág. 306-328, Springer-Verlag, 1989.

[LaL90] L. Lamport y N. Lynch. Distributed Computing: Models and Methods. Handbook of Theoretical Computer Science. Elsevier Publishers B.V. 1990.

[Lam83] Lamport, L. Sometimes is sometimes “Not never”-on the Temporal Logic of Programs. Proc. 7th Ann. ACM Symp. On Principles of Programming Languages 1980 (174-185).

[Lam94] L. Lamport. The temporal logic of actions. ACM Transaction on Programming Languages and Systems, 16 (3) pág 872-923, 1994.

[LeP91] Harry L. Lewis y Christos H. Papadimitriou. Elements of the Theory of Computation. Prentice Hall International Editions, 1981.

[LH94a] Lano, K. y Houghton, H. Object-oriented Specification Case Studies. Object-oriented series. Prentice Hall, 1994.

[LH94b] Lano, K. y Houghton, H. The Z++ manual, Octubre 1994.

[Man74] Zohar Manna. Mathematical Theory of Computation. McGrwHill, 1974.

[MaP91] Zohar Manna y Amir Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Specification. Springer-Verlag 1991.

[MaW85] Zohar Manna y Richard Waldinger. The Logical Basis for Computer Programming. Vol 1: Deductive Reasoning.

[Mau] The Maude System. http://maude.csl.sri.com/

[Mau99] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer y José Quesada. Maude: Specification and Programming Language in Rewriting Logic. Informe Técnico Computer Science Laboratory. SRI International. 1999.

[Men87] Elliot Mendelson. Introduction to Mathematical Logic. 3rd ed. Wadsworth & Books/Cole Mathematics Series. 1987.

[Mes92] Meseguer, J. Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science, 96:73-155, 1992.

[Mes93] Meseguer, J. A Logical Theory of Concurrent Objects ans its realization in the Maude language. En G. Agha, P. Wegner, y A. Yonezawa, eds. Research Directions in Concurrent Object-Oriented Programming, págs. 314-390. The MIT Press, 1993.

[Mey97] Bertrand Meyer. Object-Oriented Software Construction. 2nd ed. Prentice-Hall International Series, 1997.

[MG94] Malcom, G. y Goguen, J. Proving Correctness of Refinement and Implementation. Prg-114, Oxford Technical Monograph, Oxford University, 1994.

[Mil90] R. Milner. Operational and Algebraic Semantics of Concurrent Process. Handbook of Theoretical Computer Science. Elsevier Publishers B.V. 1990.

[MMG92] A. Morzenti, D. Mandrioli y C. Ghezzi. A model-parametric real-time logic. ACM Transaction on Programming Language and Systems, 14 (4) pág. 521-573, Oct 1992.

[MP94] A. Morzenti y P. San Pietro. Object-oriented logical specifications of time-critical systems. ACM Transactions on Software Engineering and Methodology, 3 pág 56-98, 1994.

[MW92] Meseguer, J. y Winkler, T. Parallel Programming in Maude. En J. P. Banatre y D. Le Metayer, eds. Research Directions en High-Level Parallel Programming Languages, LNCS 574 págs 253-293 Springer-Verlag, 1992.

[Pnu86] Pnueli, P. Application of temporal logic to the specification and verification of reactive systems: a survey of current trends, en Current Trends in Concurrency: Overviews and Tutorials, LNCS, Vol 224 Springer, 1986.

[RBP+91] James Rumbaugh, Michel Blaha, William Premerlani, Frederick Eddy y William Lorensen. Modelado y Diseño Orientado a Objetos. Metodología OMT. Prentice-Hall, 1996.

[Rei97] Wolgang Reising. Petri nets and algebraic specifications. In Theoretical Compuetr Science, volume 80, page 1-34. Elsevier, 1991.

[RJB99] James Rumbaugh, Ivar Jacobson y Grady Booch. The Unified Modelling Language Reference Manual. Addison-Wesley, 1999.

[RS94] Ramos J. y Sernadas A. A brief introduction to gnome. http://www.cs.math.ist.utl.pt/cs/lcg/gnome.html.

[Smi92] G. Smith. An Object-Oriented Approach to Formal Specification. PhD thesis, Dept. of Computer Science, Oct. 1992.

[Smi95-1] G. Smith. A Logic for Object-Z. Informe Técnico 95-26. Software Verification Research Centre, The University of Queensland Dept. of Computer Science, Sept. 1995.

[Smi95-2] G. Smith. Formal Verification of Object-Z Specification. Informe Técnico 95-55. Software Verification Research Centre, The University of Queensland Dept. of Computer Science, Dec. 1995.

[Smi00] G. Smith. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers 2000.

[[SMV83] Schwartz, R, P. Melliar-Smith y F. Vogt. An interval logic for higher-level temporal reasoning in Proc.2nd Ann. ACM Symp. On Principles of Distributed Computing (1983) 173-186.

SoW98] D'Souza, D. y Wills, A. C. Catalysis—Objects, components and Frameworks with UML. Addison-Wesley 1998.

[Spi92] Spivey, J. M. The Z Notation. Prentice Hall series in computer science, 1992.

[Wie91] Roel Wieringa. Equational Specification of dynamic objects. In Object-Oriented Data Bases: Analysis, Design and Construction. North-Holland, 1991.

[Wir90] Martin Wirsing. Algebraic Specification. Handbook of Theoretical Computer Science. Volumen B. Formal Models and Semantics, Capítulo 13. Elsevier, 1990.

[ZCC99] H. Zedan, A. Cau, Z. Chen, H. Yang. ATOM : An Object-Based Formal Method for Real-Time Systems. Informe Técnico en Software Technology Research Laboratory, SRE Centre, De Montfort University, The Gateway, Leicester http://www.cms.dmu.ac.uk/STRL/,1999.