Tesis Doctoral

MÉTODO PARA EL ANÁLISIS INDEPENDIENTE DE PROBLEMAS

presentada en la

Facultad de Informática

de la

Universidad Politécnica de Madrid

para la obtención del

Grado de Doctor en Informática

Autor: Loïc A. Martínez Normand

Licenciado en Informática por la

Universidad Politécnica de Madrid

Director: César Montes Gracia

Madrid, Septiembre de 2003

A Yolanda,

a mis padres,

a mis hermanos.

RESUMEN

El desarrollo de software es una actividad inherentemente compleja y que implica habilidades de comprensión y resolución de problemas. Tanto es así que se puede afirmar con rotundidad que no existe el método perfecto para cada una de las etapas de desarrollo y tampoco existe el modelo de ciclo de vida perfecto: cada nuevo problema que se plantea es diferente a los anteriores en algún aspecto y esto hace que técnicas que funcionaron en proyectos anteriores fracasen en los proyectos nuevos.

Por ello actualmente se realiza un planteamiento integrador que pretende utilizar en cada caso las técnicas, métodos y herramientas más acordes con las características del problema planteado al ingeniero. Bajo este punto de vista se plantean nuevos problemas.

En primer lugar está la selección de enfoques de desarrollo. Si no existe el mejor enfoque, ¿cómo se hace para elegir el más adecuado de entre el conjunto de los existentes?

Un segundo problema estriba en la relación entre las etapas de análisis y diseño. En este sentido existen dos grandes riesgos. Por un lado, se puede hacer un análisis del problema demasiado superficial, con lo que se produce una excesiva distancia entre el análisis y el diseño que muchas veces imposibilita el paso de uno a otro. Por otro lado, se puede optar por un análisis en términos del diseño que provoca que no cumpla su objetivo de centrarse en el problema, sino que se convierte en una primera versión de la solución, lo que se conoce como diseño preliminar.

Como consecuencia de lo anterior surge el dilema del análisis, que puede plantearse como sigue: para cada problema planteado hay que elegir las técnicas más adecuadas, lo que requiere que se conozcan las características del problema. Para ello, a su vez, se debe analizar el problema, eligiendo una técnica antes de conocerlo. Si la técnica utiliza términos de diseño entonces se ha precondicionado el paradigma de solución y es posible que no sea el más adecuado para resolver el problema.

En último lugar están las barreras pragmáticas que frenan la expansión del uso de métodos con base formal, dificultando su aplicación en la práctica cotidiana.

Teniendo en cuenta todos los problemas planteados, se requieren métodos de análisis del problema que cumplan una serie de objetivos, el primero de los cuales es la necesidad de una base formal, con el fin de evitar la ambigüedad y permitir verificar la corrección de los modelos generados.

Un segundo objetivo es la independencia de diseño: se deben utilizar términos que no tengan reflejo directo en el diseño, para que permitan centrarse en las características del problema. Además los métodos deben permitir analizar problemas de cualquier tipo: algorítmicos, de soporte a la decisión o basados en el conocimiento, entre otros.

En siguiente lugar están los objetivos relacionados con aspectos pragmáticos. Por un lado deben incorporar una notación textual formal pero no matemática, de forma que se facilite su validación y comprensión por personas sin conocimientos matemáticos profundos pero al mismo tiempo sea lo suficientemente rigurosa para facilitar su verificación. Por otro lado, se requiere una notación gráfica complementaria para

representar los modelos, de forma que puedan ser comprendidos y validados cómodamente por parte de los clientes y usuarios.

Esta tesis doctoral presenta SETCM, un método de análisis que cumple estos objetivos. Para ello se han definido todos los elementos que forman los modelos de análisis usando una terminología independiente de paradigmas de diseño y se han formalizado dichas definiciones usando los elementos fundamentales de la teoría de conjuntos: elementos, conjuntos y relaciones entre conjuntos. Por otro lado se ha definido un lenguaje formal para representar los elementos de los modelos de análisis – evitando en lo posible el uso de notaciones matemáticas – complementado con una notación gráfica que permite representar de forma visual las partes más relevantes de los modelos.

El método propuesto ha sido sometido a una intensa fase de experimentación, durante la que fue aplicado a 13 casos de estudio, todos ellos proyectos reales que han concluido en productos transferidos a entidades públicas o privadas.

Durante la experimentación se ha evaluado la adecuación de SETCM para el análisis de problemas de distinto tamaño y en sistemas cuyo diseño final usaba paradigmas diferentes e incluso paradigmas mixtos. También se ha evaluado su uso por analistas con distinto nivel de experiencia – noveles, intermedios o expertos – analizando en todos los casos la curva de aprendizaje, con el fin de averiguar si es fácil de aprender su uso, independientemente de si se conoce o no alguna otra técnica de análisis. Por otro lado se ha estudiado la capacidad de ampliación de modelos generados con SETCM, para comprobar si permite abordar proyectos realizados en varias fases, en los que el análisis de una fase consista en ampliar el análisis de la fase anterior. En resumidas cuentas, se ha tratado de evaluar la capacidad de integración de SETCM en una organización como la técnica de análisis preferida para el desarrollo de software.

Los resultados obtenidos tras esta experimentación han sido muy positivos, habiéndose alcanzado un alto grado de cumplimiento de todos los objetivos planteados al definir el método.

ABSTRACT

Software development is an inherently complex activity, which requires specific abilities of problem comprehension and solving. It is so difficult that it can even be said that there is no perfect method for each of the development stages and that there is no perfect life cycle model: each new problem is different to the precedent ones in some respect and the techniques that worked in other problems can fail in the new ones.

Given that situation, the current trend is to integrate different methods, tools and techniques, using the best suited for each situation. This trend, however, raises some new problems.

The first one is the selection of development approaches. If there is no a manifestly single best approach, how does one go about choosing an approach from the array of available options?

The second problem has to do with the relationship between the analysis and design phases. This relation can lead to two major risks. On one hand, the analysis could be too shallow and far away from the design, making it very difficult to perform the transition between them. On the other hand, the analysis could be expressed using design terminology, thus becoming more a kind of preliminary design than a model of the problem to be solved.

In third place there is the analysis dilemma, which can be expressed as follows. The developer has to choose the most adequate techniques for each problem, and to make this decision it is necessary to know the most relevant properties of the problem. This implies that the developer has to analyse the problem, choosing an analysis method before really knowing the problem. If the chosen technique uses design terminology then the solution paradigm has been preconditioned and it is possible that, once the problem is well known, that paradigm wouldn't be the chosen one.

The last problem consists of some pragmatic barriers that limit the applicability of formal based methods, making it difficult to use them in current practice.

In order to solve these problems there is a need for analysis methods that fulfil several goals. The first one is the need of a formal base, which prevents ambiguity and allows the verification of the analysis models.

The second goal is design-independence: the analysis should use a terminology different from the design, to facilitate a real comprehension of the problem under study. In third place the analysis method should allow the developer to study different kinds of problems: algorithmic, decision-support, knowledge based, etc.

Next there are two goals related to pragmatic aspects. Firstly, the methods should have a non mathematical but formal textual notation. This notation will allow people without deep mathematical knowledge to understand and validate the resulting models, without losing the needed rigour for verification. Secondly, the methods should have a complementary graphical notation to make more natural the understanding and validation of the relevant parts of the analysis.

This Thesis proposes such a method, called SETCM. The elements conforming the analysis models have been defined using a terminology that is independent from design paradigms. Those terms have been then formalised using the main concepts of the set theory: elements, sets and correspondences between sets. In addition, a formal language has been created, which avoids the use of mathematical notations. Finally, a graphical notation has been defined, which can visually represent the most relevant elements of the models.

The proposed method has been thoroughly tested during the experimentation phase. It has been used to perform the analysis of 13 actual projects, all of them resulting in transferred products.

This experimentation allowed evaluating the adequacy of SETCM for the analysis of problems of varying size, whose final design used different paradigms and even mixed ones. The use of the method by people with different levels of expertise was also evaluated, along with the corresponding learning curve, in order to assess if the method is easy to learn, independently of previous knowledge on other analysis techniques. In addition, the expandability of the analysis models was evaluated, assessing if the technique was adequate for projects organised in incremental steps, in which the analysis of one step grows from the precedent models. The final goal was to assess if SETCM can be used inside an organisation as the preferred analysis method for software development.

The obtained results have been very positive, as SETCM has obtained a high degree of fulfilment of the goals stated for the method.

AGRADECIMIENTOS

En primer lugar quiero agradecer a mi mujer, Yolanda, y a mi familia todo el apoyo que me han demostrado a lo largo de todos estos largos años que han transcurrido durante la preparación de esta Tesis doctoral.

En segundo lugar quiero agradecer a mis directores de tesis, César Montes y Fernando Alonso Amo (este último no presente por absurdas normas de la Universidad), todo el esfuerzo que han dedicado para lograr que este trabajo tenga la calidad adecuada.

Mi tercer agradecimiento es para mis compañeros de trabajo, en orden alfabético: Ángel Lucas, Chema, Genoveva, Paloma, Pepe y Sonia. Sin vuestro apoyo y aportaciones este trabajo no habría sido posible.

También quiero aprovechar la ocasión para agradecer la amistad que me han ofrecido desde hace muchos años mis compañeros de carrera: Camino, Enrique y Juanma, y sus compañeros Lorena y Vicente.

Por último quiero hacer un agradecimiento general a todos los que olvido pero seguro merecerían aparecer en esta página.

Gracias a todos.

TABLA DE CONTENIDOS

ANEXOS

(EN VOLUMEN ADICIONAL)

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 EL DESARROLLO DE SOFTWARE

Desde los inicios de la informática el desarrollo del software ha sido considerado como una tarea de mucha más complejidad que el desarrollo del hardware. De hecho una afirmación muy frecuente dentro de la profesión informática es que el hardware está muy por delante del software en cuanto a capacidad o, dicho de otro modo, que el software no está a la altura de lo que el hardware podría ofrecer.

La constatación de este hecho es lo que se conoce como "*crisis del software*", cuya existencia se reconoce oficialmente por primera vez a finales de los años 60 y llevó a la OTAN a organizar sus conocidas conferencias [Naur et al., 1969].

En la primera de esas conferencias se dio origen al término "*ingeniería del software*", imponiéndose en la comunidad científica la idea de fundamentar el desarrollo de programas informáticos en los principios básicos existentes en otras ingenierías, incluyendo el propio desarrollo de hardware, con el fin de solucionar los problemas detectados en la práctica habitual. En palabras de los organizadores de la conferencia:

> *"La frase 'Ingeniería del Software' fue elegida deliberadamente como una provocación, implicando la necesidad de que la construcción de software estuviera basada en el tipo de fundamentos teóricos y disciplinas prácticas que son tradicionales en las ramas ya establecidas de la ingeniería." [Naur et al., 1969]*

Partiendo del proceso de trabajo aplicado en otras ingenierías surgió el ciclo de vida tradicional de desarrollo de programas, conocido como "modelo en cascada" [Royce, 1970] y que dividió el proceso de desarrollo de software en una serie de etapas bien diferenciadas, que se deberían llevar a cabo de forma secuencial:

1. *Análisis del sistema*: se trata de definir qué tiene que hacer el sistema completo.
2. *Análisis de los requisitos del software*: define qué tiene que hacer el software.
3. *Diseño de software*: se trata de definir cómo será el software.
4. *Codificación*: consiste en *construir el software*.
5. *Pruebas*: comprueba el funcionamiento mediante la ejecución de casos de prueba.
6. *Mantenimiento*: refleja el proceso de realización de modificaciones en el programa debidos a errores, cambios en el entorno o mejoras solicitadas por el cliente.

Con el tiempo se han desarrollado nuevos modelos de ciclo de vida, como el basado en prototipos [Gomaa, 1984] , el desarrollo de prototipos rápidos [Connell et al., 1989] y el modelo en espiral [Boehm, 1988] pero en todos ellos se ha mantenido la distinción básica entre las etapas de análisis, diseño, codificación, pruebas y mantenimiento.

1.2 SOBRE LA NO EXISTENCIA DEL MÉTODO PERFECTO

Como recoge [Shapiro, 1997], durante los más de 30 años de existencia de la ingeniería del software como disciplina han surgido multitud de teorías, técnicas, modelos y herramientas para llevar a cabo cada una de las etapas de desarrollo anteriores que han conllevado amargos y encendidos debates entre los defensores de propuestas teóricamente opuestas.

La conclusión final de todos estos debates es doble:

- En primer lugar ha quedado de manifiesto que **el desarrollo de software es una actividad inherentemente compleja y que implica tareas de resolución de problemas**. Como indica el propio Shapiro,

  "La complejidad puede ser un fenómeno fundamental y la resolución de problemas una actividad fundamental, pero ninguna de las dos es simple. Por el contrario ambas con complicadas y tienen múltiples facetas, desafiando a menudo una comprensión o resolución directa" [Shapiro, 1997]

- En segundo lugar, todos los debates existentes han llevado a la conclusión de que **no existe el método perfecto** para cada una de las etapas de desarrollo y tampoco existe el modelo de ciclo de vida perfecto: cada nuevo problema que se plantea es diferente a los anteriores en algún aspecto y esto hace que técnicas que funcionaron en proyectos anteriores fracasen en los proyectos nuevos. Como se dice en Estados Unidos, no existe la "bala de plata" en el desarrollo de software. A continuación se recogen algunas citas que corroboran esta afirmación:

  "(los desarrolladores) han descubierto que la búsqueda de la mejor metodología es vana y que deberían ser capaces de elegir entre una 'armería' de enfoques que puedan integrar" [Finkeslstein, 1984]

  "Ningún enfoque puede ser definido como 'superior' al resto. En su lugar el arte está en aplicar el enfoque más adecuado a las características internas y externas del problema" [Episkopou et al., 1986]

  "la construcción de software será siempre difícil. Intrínsicamente, no existe ninguna bala de plata" [Brooks, 1987]

  "Ningún enfoque único en cualquier aspecto de la tecnología software ha podido satisfacer las necesidades o deseos de los desarrolladores" [Shapiro, 1997]

  "En general se considera que, por mucho que se busque, un proceso aplicable de forma general seguirá siendo para la ingeniería del software un Santo Grial inalcanzable e imaginario. " [Graham et al., 1997a]

  "Ningún proceso único de desarrollo de software puede aplicarse en cualquier parte" [Jacobson et al., 1999]

Ambas ideas han llevado a ver la ingeniería del software como una ingeniería única en la que no son fácilmente aplicables los principios básicos de otras ingenierías:

"la complejidad del software ha llevado a una nueva situación más allá del alcance de esfuerzos anteriores de ingeniería, obligando a los estudiosos de la materia a 'saltarse las reglas'" [Myers, 1985]

Por lo tanto el enfoque actual en la rama de la ingeniería del software consiste en poner a disposición del ingeniero una 'caja de herramientas' con distintas técnicas, métodos, procesos, etc., cada uno con sus ventajas y sus inconvenientes, de entre los cuales debe elegir aquélla o aquéllas más relevantes al problema que se le plantea.

1.3 INTEGRANDO LA INGENIERÍA DEL CONOCIMIENTO

En un enfoque lo más integrador posible, dentro de esta 'caja de herramientas' también deben tener cabida los esfuerzos realizados en otra rama de la informática, relacionada con la Inteligencia Artificial, llamada "Ingeniería del Conocimiento" y cuyo objetivo es el desarrollo de sistemas basados en el conocimiento [Gómez et al., 1997], [Studer et al., 1998]. Puede definirse como:

> *"el conjunto de principios, métodos y herramientas que permiten aplicar el saber científico y de experiencia a la utilización de los conocimientos y de sus fuentes, mediante invenciones o construcciones útiles para el hombre"[Gómez et al., 1997].*

El nombre de esta disciplina de la Inteligencia Artificial también demuestra el deseo de sus participantes de transformarla en una ingeniería. En Ingeniería del Conocimiento cambian los nombres de las etapas de desarrollo que, sin embargo, pueden equipararse de algún modo con las correspondientes etapas clásicas de la ingeniería del software:

1. *Estudio de viabilidad*, que para muchos es una parte del análisis del sistema.
2. *Adquisición de conocimientos*, etapa propia de la IC, y que consiste en un proceso de recolección de la información y conocimientos necesarios para construir un sistema basado en el conocimiento.
3. *Conceptualización o modelado conceptual* (con propósito similar al análisis), que pretende construir un modelo conceptual a partir de los conocimientos adquiridos.
4. *Formalización* (que podría ser equiparable al diseño), consistente en representar el modelo conceptual utilizando técnicas de representación de conocimiento comprensibles para el ordenador.
5. *Implementación*, consistente en la construcción del sistema informático.

Inicialmente la Ingeniería del Conocimiento (IC) tuvo un desarrollo paralelo e independiente de la Ingeniería del Software (IS), pero hoy en día los practicantes de ambas disciplinas se han percatado de que son complementarias y que todo desarrollador de programas informáticos debería conocer los fundamentos de ambas para poder aplicarlas según interese más. En palabras de Gómez y colegas:

> *"Durante mucho tiempo la IS y la IC eran dos disciplinas que, en el mejor de los casos, se ignoraban, y en el peor se despreciaban mutuamente. Cuando, posteriormente, los practicantes de ambas disciplinas se dieron cuenta de que no sólo eran complementarias, sino que cualquier informático debería conocer los procedimientos, técnicas, métodos y herramientas, tanto de la una como de la otras, para poder desarrollar adecuadamente su labor, las interacciones entre ambas han sido habituales y provechosas por el efecto sinérgico logrado. Esta convergencia es esencial para evitar la duplicidad de esfuerzos centrándose fundamentalmente en el desarrollo de una metodología común para la producción de software que sirva, coherentemente, a ambas ingenierías." [Gómez et al., 1997]*

1.4 EL PROBLEMA DE LA SELECCIÓN DE ENFOQUES

Dentro de este planteamiento integrador que pretende utilizar en cada caso las técnicas, métodos y herramientas más acordes con las características del problema planteado al ingeniero, la gran cuestión pendiente radica en cómo llevarlo a la práctica, lo que podríamos denominar el **problema de la selección de enfoques de desarrollo**:

> *"Si no existe el mejor enfoque, ¿cómo hace uno para elegir uno de ellos dentro del conjunto de los existentes?" [Shapiro, 1997]*

Según el propio Shapiro, esta pregunta puede dividirse en tres partes bien diferenciadas:

1. **Clasificación de enfoques de desarrollo**: en primer lugar, debe definirse una forma de evaluar y caracterizar los diferentes enfoques existentes para poder razonar sobre ellos:

   "¿Cómo hace uno para caracterizar los diferentes enfoques para facilitar el razonamiento acerca de los mismos?"

2. **Análisis del problema planteado**: en segundo lugar, debe definirse un método para estudiar cada situación que se le plantea al desarrollador:

   "¿Cómo hace uno para caracterizar los atributos de la situación actual en términos de problemas o tareas, organización, cultura, etc.?"

3. **Equiparación entre enfoques y situaciones**: por último, debería detallarse una forma de comparar los atributos de los enfoques de desarrollo con las características del problema para poder elegir la mejor forma de resolverlo:

   "¿Cómo se relacionan las características de los primeros (los enfoques) con las propiedades de las segundas (las situaciones)?"

La segunda cuestión pone de manifiesto la gran importancia que tiene la etapa de estudio del problema dentro del desarrollo de programas informáticos, dado que es en esta etapa cuando se define cuáles son los objetivos del sistema y cuáles son sus características, para así poder elegir las mejores técnicas, métodos y herramientas para su desarrollo.

Esta etapa tiene nombres muy diferentes según los autores. En el campo de la Ingeniería del Software suele denominarse análisis, captura de requisitos o incluso ingeniería de requisitos. En el campo de la Ingeniería del Conocimiento suele llamarse conceptualización, modelado conceptual o modelado de conocimiento. El término utilizado en esta Tesis será el de **análisis del problema** o simplemente análisis.

1.5 RIESGOS EN LA RELACIÓN ANÁLISIS - DISEÑO

Tal y como recoge Blum [Blum, 1996] el proceso de desarrollo de software puede verse en esencia como la aplicación de tres transformaciones (Figura 1.1):

- T1: una necesidad en el dominio de la aplicación se transforma en un modelo conceptual de ese problema. Esta transformación es el análisis.
- T2: el modelo conceptual se transforma en un modelo formal que representa las propiedades del producto software. Esta transformación se denomina diseño.

- T3: el modelo formal se transforma en un modelo computable (programa) que se ejecuta en un ordenador. Esta transformación suele denominarse implementación.

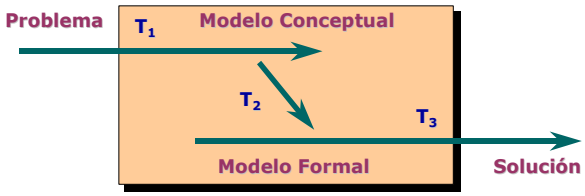


Figura 1.1 La esencia del proceso software según [Blum, 1996]

Dentro de este esquema general es clave la relación entre el modelo conceptual (modelo del problema) y el modelo formal (modelo de la solución) o, dicho de otra forma, entre el análisis y el diseño.

Hay *dos grandes riesgos* en la práctica cotidiana del desarrollo de software:

1. El primer riesgo consiste en hacer un *análisis del problema demasiado superficial* de forma que la información contenida en el modelo de análisis no sea suficientemente detallada para garantizar una buena decisión sobre el mejor diseño para resolver el problema. En situaciones de este tipo se produce una excesiva distancia entre el análisis y el diseño que muchas veces imposibilita la realización de la transición T2 (Figura 1.2).

Figura 1.2 El riesgo de un análisis poco detallado

2. El segundo riesgo consiste en la *realización de un análisis en términos del diseño* que, aparentemente, elimina la barrera existente entre los modelos conceptual y formal. Sin embargo lo que realmente ocurre es que el análisis no cumple su objetivo de centrarse en el problema, con lo que el resultado de la primera transición no es un modelo del problema, sino una primera versión de la solución, lo que se conoce como diseño preliminar (Figura 1.3).

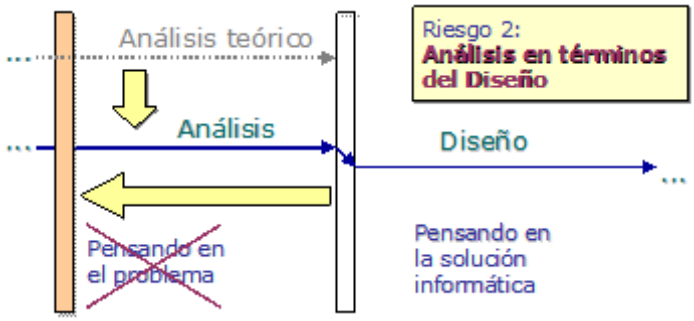


Figura 1.3 El riesgo de un análisis en términos del diseño

1.6 EL DILEMA DEL ANÁLISIS

El hecho de realizar un análisis del problema en términos del diseño adquiere una gran relevancia dentro del enfoque integrador y unificador de la ingeniería del software en la actualidad.

Es lo que el autor de esta tesis y otros colegas han definido como el **dilema del análisis,** refiriéndose a lo que ocurre cuando se utilizan metodologías tradicionales de desarrollo de software:

> *"Surge un dilema que denominamos el dilema del análisis y que puede definirse como sigue. Para elegir la mejor forma de resolver un problema el ingeniero debe analizarlo, indicando qué debe hacerse y cuáles son las características principales del problema. Este análisis debe realizarse siguiendo alguna de las metodologías actuales. Pero una vez que se ha seleccionado una metodología, el ingeniero está obligado a seguir usándola durante el resto del proceso de desarrollo, incluso aunque los resultados del análisis indiquen que otra metodología podría ser más adecuada. Esto es así porque es muy difícil cambiar de una metodología a otra en medio del proceso de desarrollo."[Alonso et al., 1999a]*

El razonamiento anterior sigue el siguiente esquema:

- En un enfoque integrador hay que elegir las técnicas más adecuadas para cada problema planteado.
- Para ello hay que conocer las características del problema, es decir, hay que analizar el problema.
- Para analizar el problema hay que elegir una técnica antes de conocer el problema.
- Si la técnica utiliza términos de diseño (como el análisis orientado a objetos o el análisis estructurado) entonces se ha precondicionado el paradigma de solución y es posible que no sea el más adecuado para el problema.

Para evitar el dilema del análisis es necesario utilizar técnicas de análisis que sean independientes de las técnicas de diseño:

> *"Se necesitan técnicas de análisis que permitan estudiar cualquier tipo de problema con el fin de proporcionar información suficiente para elegir la mejor forma de continuar el proceso de desarrollo. Estas técnicas de análisis deberían ser independientes de las técnicas utilizadas durante el diseño y la implementación"[Alonso et al., 1999a]*

Es lo que Loucopoulos y Karaostas [Loucopoulos et al., 1995] llaman **independencia de implementación** y otros autores también han defendido [Blum, 1996] [Bonfatti et al., 1994], [Høydalsvik et al., 1993].

1.7 SOBRE LOS MÉTODOS FORMALES

Una tendencia para resolver el dilema del análisis que existe desde hace tiempo consiste en la utilización técnicas formales de especificación de requisitos, como son Z [ISO, 2002], VDM [ISO, 1996] y otras.

Estas técnicas se basan en el uso de notaciones matemáticas formales para representar los requisitos de un sistema software. El formalismo matemático permite dos grandes avances:

- Por un lado se puede verificar formalmente que una especificación es completa y consistente.
- Por otro lado, se pueden aplicar técnicas de demostración de teoremas para transformar la especificación en una implementación asegurando la corrección formal del proceso.

A pesar de estas ventajas, los métodos formales no se están utilizando de forma habitual en la industria, como constatan varios autores:

> *"A pesar de sus beneficios los métodos formales todavía no se utilizan ampliamente en compañías comerciales" [Knight et al., 1997]*
>
> *"Los lenguajes de especificación formal no han sido usados ampliamente en la industria" [Dulac et al., 2002]*

En un principio se pensaba que esta situación se debía al rechazo por parte de los ingenieros a aprender nuevas técnicas de desarrollo. Sin embargo, estudios exhaustivos como el de Knight y colegas [Knight et al., 1997] han llegado a la conclusión de que **los métodos formales deben superar barreras de índole pragmática** que impiden su aplicación a proyectos reales:

> *"Los métodos formales deben salvar varios obstáculos relativamente mundanos pero importantes desde el punto de vista práctico" [Knight et al., 1997]*

1.8 APORTACIÓN DE ESTA TESIS

Esta tesis doctoral presenta una propuesta de método para realizar el análisis de problemas llamado SETCM, que viene del inglés *"SET Theory based Conceptual Modelling"* (Modelado conceptual basado en teoría de conjuntos).

La principal aportación de este método es la **combinación simultánea de aspectos formales y pragmáticos**, con el fin de resolver los dos grandes problemas planteados en esta introducción:

- Problema de Blum: la independencia del análisis respecto a las técnicas de diseño.
- Problema de Shapiro: la independencia respecto a los tipos de problemas.

Al mismo tiempo la técnica propuesta pretende cumplir todas las propiedades importantes que debería tener un modelo conceptual [Loucopoulos et al., 1995]: abstracción, no ambigüedad, facilidad para la comprensión del problema, facilidad para la verificación del modelo construido y, finalmente, independencia respecto de las técnicas que pueden usarse en el resto de las etapas de desarrollo.

Las **características** fundamentales del método propuesto son las siguientes:

- Tiene una **base formal** sencilla pero consistente, basada en los elementos fundamentales de la teoría de conjuntos: elementos, conjuntos y relaciones entre

conjuntos. Así puede cumplir con las propiedades de independencia de tipo de problema, abstracción, no ambigüedad y facilidad de verificación.

- Tiene en cuenta **aspectos pragmáticos** que se plasman en una notación textual y gráfica diseñada para ser lo más fácil posible de interpretar. Así se cumple el objetivo de la facilidad de comprensión.
- Los elementos definidos **evitan el uso de términos de diseño o de implementación**, con el fin de cumplir la independencia de las técnicas de diseño.

En cuanto al método científico seguido en la realización de la presente Tesis, se ha puesto especial hincapié en la evaluación continua de los resultados parciales que se iban obteniendo, de forma que se garantizara una aproximación progresiva a los objetivos anteriores. Esto ha requerido un proceso de experimentación ingente tanto en recursos como en tiempo, como podrá apreciarse en el correspondiente capítulo de este documento.

Finalmente el autor advierte que en ningún momento se defenderá SETCM como la única forma correcta de analizar un problema, afirmación que entraría en contradicción con las ideas expuestas al comienzo de esta introducción: "no existe la mejor solución". Por el contrario lo que el autor defiende es que SETCM es una de las posibles técnicas que pueden utilizarse para el modelado conceptual, que tiene una base formal pero que no descuida aspectos pragmáticos, lo que ha permitido utilizarla en el desarrollo de proyectos reales sin incurrir en sobrecargas de formación o de esfuerzo. El método propuesto cumple además con la importante característica de independencia con el diseño y, por lo tanto, su utilización no implica ninguna selección previa de técnicas de diseño e implementación, sino que permitirá elegir las técnicas más adecuadas en función de las características del modelo resultante.

2 ESTADO DE LA CUESTIÓN

El término *análisis* ha sido utilizado de muy diversas formas a lo largo de la historia del desarrollo de software. Con el fin de centrar correctamente el ámbito de discurso de esta tesis, conviene en primer lugar definir con precisión ese término. Al realizar esta definición se comprobará cómo el análisis del problema (su comprensión) tiene mucho que ver con el método científico, por lo que se hace necesario hacer un breve recorrido por el tratamiento del conocimiento desde la perspectiva científica con el fin de estudiar sus cualidades y limitaciones. Una vez centrado este concepto, el resto del estado de la cuestión se dedicará a ofrecer una perspectiva sobre la forma en la que las metodologías de desarrollo de software más relevantes abordan la fase de análisis. El capítulo terminará con unas conclusiones sobre la situación actual que servirán para fundamentar las raíces del método propuesto.

2.1 SOBRE EL ANÁLISIS DEL PROBLEMA

2.1.1 Definición de Análisis del Problema

Para definir con precisión el término *análisis del problema* se va a acudir al Diccionario de la Lengua Española de la Real Academia Española, en su vigésimo segunda edición [RAE, 2001], por ser éste el diccionario normativo del idioma castellano, en el cual está escrito esta Tesis. Esta definición se realizará en primer lugar fuera del ámbito informático para, más adelante, centrarse en su aplicación al desarrollo de programas.

Las dos primeras acepciones que recoge el Diccionario de la Real Academia Española del término *análisis* son las siguientes:

> ***análisis***. *(Del gr. ἀνάλυσις).* ***1***. *m. Distinción y separación de las partes de un todo hasta llegar a conocer sus principios o elementos.* ***2***. *m. Examen que se hace de una obra, de un escrito o de cualquier realidad susceptible de estudio intelectual.*

La primera acepción es la conocida tradicionalmente como análisis en la mayoría de las disciplinas, como por ejemplo, los análisis químicos y clínicos. De hecho esta misma acepción de análisis es la utilizada por Descartes para denominar el segundo paso de su método cartesiano, la regla del análisis: "*dividir cada una de las dificultades que se hallasen al paso, en tantas partes como fuese posible y requiera su más fácil solución*" [Descartes, 1977].

La segunda acepción de análisis (*"examen de cualquier realidad susceptible de estudio intelectual"*) es la que se ha utilizado cuando se habla del análisis como una etapa dentro del desarrollo de programas. Para apoyar esta afirmación debe profundizarse en los elementos que conforman esta definición.

El término *examen* tiene la siguiente primera acepción:

> ***examen***. *(Del lat. exāmen).* ***1***. *m. Indagación y estudio que se hace acerca de las cualidades y circunstancias de una cosa o de un hecho.*

Profundizando en esta acepción, *indagación* y *estudio* tienen las siguientes definiciones:

> ***indagación***. *(Del lat. indagatĭo, -ōnis).* ***1***. *f. Acción y efecto de indagar.*
>
> ***indagar***. *(Del lat. indagāre).* ***1***. *tr. Intentar averiguar, inquirir algo discurriendo o con preguntas.*

> ***estudio**. (Del lat. studĭum). **1**. m. Esfuerzo que pone el entendimiento aplicándose a conocer algo.*

Dentro de la acepción de *estudio* interesa el significado de los términos *entendimiento* y *conocer*:

> ***entendimiento**. **3**. m. Razón humana.*
>
> ***conocer**. (Del lat. cognoscĕre). **1**. tr. Averiguar por el ejercicio de las facultades intelectuales la naturaleza, cualidades y relaciones de las cosas.*

Volviendo a la definición de examen, interesa también definir los términos *cualidad* y *circunstancias*:

> ***cualidad**. (Del lat. qualĭtas, -ātis). **1**. f. Cada uno de los caracteres, naturales o adquiridos, que distinguen a las personas, a los seres vivos en general o a las cosas.*
>
> ***circunstancia**. (Del lat. circumstantĭa). **1**. f. Accidente de tiempo, lugar, modo, etc., que está unido a la sustancia de algún hecho o dicho. **2**. f. Calidad o requisito. **3**. f. Conjunto de lo que está en torno a alguien; el mundo en cuanto mundo de alguien.*

El término *examen*, que se utiliza en la segunda acepción de análisis, implica, por lo tanto, el intento de averiguar algo discurriendo o con preguntas, realizando al mismo tiempo un esfuerzo racional para conocer la realidad analizada (sus propiedades y entorno), o lo que es lo mismo, para averiguar la naturaleza, cualidades y relaciones presentes en esa realidad. Por lo tanto el objetivo del análisis es comprender lo que se analiza y para ello se debe realizar un esfuerzo, discurriendo o preguntando sobre esa realidad.

Siguiendo con la definición de análisis, su segunda parte es "*realidad susceptible de estudio intelectual"*. Empezando por el término *realidad*, su significado es:

> ***realidad**. 1. f. Existencia real y efectiva de algo. 2. f. Verdad, lo que ocurre verdaderamente. 3. f. Lo que es efectivo o tiene valor práctico, en contraposición con lo fantástico e ilusorio.*

Por lo tanto, el objeto de estudio del análisis, la realidad, es todo aquello que existe realmente o tiene valor práctico.

Para terminar con la segunda acepción de análisis, queda pendiente la definición de *estudio intelectual*. Ya se ha mostrado anteriormente el significado de estudio, por lo que sólo falta definir el término intelectual:

> ***intelectual**. (Del lat. intellectuālis). 1. adj. Perteneciente o relativo al entendimiento.*

Es decir, que el objeto del análisis es cualquier hecho real o de valor práctico que pueda comprenderse o conocerse, lo cual ya estaba implícito en la definición de análisis como examen, tal y como se ha ido viendo en las distintas definiciones del diccionario.

Teniendo esto en cuenta, ¿tiene sentido hablar de "análisis del problema"? Para ello debe definirse el término *problema*:

> ***problema**. (Del lat. problēma, y este del gr. πρόβλημα). **1**. m. Cuestión que se trata de aclarar. **2**. m. Proposición o dificultad de solución dudosa. **3**. m.*

> *Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.* ***4****. m. Disgusto, preocupación.* ***5****. m. Planteamiento de una situación cuya respuesta desconocida debe obtenerse a través de métodos científicos.*

De todas estas acepciones interesan a efectos de esta Tesis la primera y la quinta. Es decir, un problema es algo que se trata de aclarar o bien el planteamiento de una situación cuya respuesta debe obtenerse por medios científicos. Por tanto un problema sí encaja dentro de la acepción de realidad, dado que es algo que tiene una existencia real (para la persona o personas que plantean el problema) y además tienen valor práctico (para esas mismas personas).

En resumidas cuentas, puede definirse el *análisis del problema*, en cualquier ámbito, de la siguiente forma:

Análisis del problema es el intento de conocer, discurriendo o con preguntas, y con esfuerzo racional, las cualidades (características**) y circunstancias (**entorno**) de una situación cuya respuesta debe obtenerse (**el problema**).**

Por tanto, y teniendo en cuenta la definición del término conocer,

El resultado del análisis del problema es la comprensión de la naturaleza, cualidades y relaciones de las características del problema y su entorno.

Todo ello encaja con la acepción tradicional de análisis dentro del desarrollo de programas, como recoge el Diccionario de la Real Academia Española en su quinta acepción de análisis:

> ***análisis****. **5****. m. Inform. Estudio, mediante técnicas informáticas, de los límites, características y posibles soluciones de un problema al que se aplica un tratamiento por ordenador.*

Es decir, que lo que distingue al análisis del problema en informática del análisis del problema en otras disciplinas es que se aplicará un tratamiento por computador como solución del problema planteado.

Por todo ello el análisis del problema en el desarrollo de software tiene como resultado comprender (conocer) las características del problema y su entorno (lo que Blum llama modelo conceptual [Blum, 1996], según se ha visto en la introducción) para luego poder automatizar su solución. Esta automatización implica que, como parte del proceso, será necesario obtener un modelo formal que pueda ser interpretado por la computadora.

Esto tiene mucho en común con el método científico, que trata de comprender una realidad compleja y representarla mediante teorías y modelos formales. Por tanto parece relevante proceder a un breve estudio de cómo se trata el conocimiento desde la perspectiva científica y para ello debe acudirse a la disciplina que estudia el conocimiento y método científicos: la filosofía de la ciencia. En palabras de Blum,

> *"... el proceso software es una transformación desde la identificación de una necesidad en el mundo a un conjunto de programas que operan en el computador. El proceso empieza con una idea, un concepto, algo que puede desafiar una descripción completa, y termina con la entrega de un modelo formal que se ejecuta en el computador. ... Hay una tensión fundamental en*

> *esta transformación, una tensión entre lo que se quiere y cómo se hace que funcione, entre los requisitos en el mundo y su realización en el computador. ... La ciencia se enfrenta a un problema similar ... empieza con algo muy complejo y mal representado – el mundo real – y su objetivo es describir aspectos de esa realidad con teorías y modelos. Sabemos que la ciencia tiene éxito. Por lo tanto es razonable mirar sus cualidades y limitaciones ... Me vuelvo a la filosofía de la ciencia porque constituye un tipo de meta-ciencia. Examina la naturaleza de la ciencia desde una perspectiva teórica; nos ayuda a apreciar lo que puede conocerse y lo que puede representarse formalmente [Blum, 1996]*.

2.1.2 El Método Científico

Este apartado va a recoger de forma resumida las ideas fundamentales que existen sobre el conocimiento y el método científico, sin pretensión alguna de ofrecer un tratado exhaustivo acerca del tema, aunque sí con la idea de repasar con brevedad aquellos enfoques que resultan relevantes para enmarcar la presente Tesis. Para más detalles puede consultarse cualquiera de los tratados recientes sobre filosofía de la ciencia, como por ejemplo [Bechtel, 1988], [Bunge, 1998] y [Díez et al., 1997], así como el estudio de la ciencia desde una perspectiva más informática realizado en [Andrade, 2002], [Ares et al., 1998], [Blum, 1996] y [Paradela, 2001].

Debe comenzar este breve resumen por una definición de conocimiento o saber científico. La definición de saber ha ido cambiando y evolucionando a lo largo de la historia [Zubiri, 1999]. Una definición actual puede ser la dada por Güell, que dice que el saber:

> *"consiste en una aprehensión de la realidad por medio de la cual ésta queda fijada en el sujeto – no física sino representativamente - y es expresado y transmitido a otros sujetos, con lo que queda sistematizado e incorporado a un cuerpo de conocimientos*" *[Güell, 2001]*.

Así, lo importante es que el saber queda fijado en el ser humano pero que al mismo tiempo tiene una representación que permite su expresión y transmisión a otras personas.

En los últimos tiempos, y como consecuencia de la creación de la disciplina de la Gestión de Conocimientos, se ha hecho necesario formalizar la definición de conocimiento, de forma que pasa de ser algo que queda limitado a las personas, a ser algo que puede ser manipulado también por los computadores [Paradela, 2001]. Así, Pazos propone la siguiente definición de conocimiento:

> *"El conocimiento queda definido formalmente como sigue:*
>
> $$C = I + T$$
>
> *Siendo: C, el conocimiento, I, la Información; es decir, datos, noticias, conocimientos y sabiduría; y T, las transformaciones u operaciones que se puedan realizar, en un contexto determinado, sobre I." [Pazos, 2003a]*

Sea cual sea la consideración del término conocimiento, la forma de obtener el saber dentro de la ciencia se denomina método científico. Dentro de la filosofía de la ciencia

existe hoy en día un intenso debate sobre la existencia o no de un método científico, tal y como recogen, por ejemplo, [Ares et al., 1998] y [Blum, 1996].

A principios del siglo XX el método científico aceptado de forma masiva seguía las pautas del Positivismo Lógico, basado en la visión racionalista del mundo originada por filósofos griegos como Platón [Platón, 1944], Euclides [Euclides, 1944] y Arquímedes [Arquímedes, 1912], y que toma especial relevancia a partir de los trabajos de Descartes [Descartes, 1969] [Descartes, 1977]: el mundo puede ser comprendido y expresado de manera formal aplicando los principios matemáticos.

Aunque el positivismo lógico ha sido refutado de forma plenamente justificada [Suppe, 1974], merece ser mencionado porque sigue siendo la base inconsciente de lo que se considera generalmente como ciencia. En palabras de Bechtel:

> *"continúa definiendo la agenda de muchas discusiones filosóficas y proporcionando los criterios que muchos científicos ... usan para juzgar lo que es buena ciencia*" *[Bechtel, 1988]*.

La filosofía del positivismo puede resumirse en la llamada "Visión Recibida", que recoge Suppe de la siguiente forma:

1. *La teoría se formula en lógica de primer orden con igualdad, L.*
2. *Los términos no lógicos o constantes de L se dividen en tres clases disjuntas llamadas vocabularios:*
    a. *El vocabulario lógico consiste en constantes lógicas (incluyendo términos matemáticos).*
    b. *El vocabulario de observaciones, V_O, contiene términos de observación.*
    c. *El vocabulario teórico, V_T, contiene términos teóricos.*
3. *Los términos en V_O se interpretan de forma que se refieren a objetos físicos observables directamente o bien a propiedades observables de los objetos físicos.*
4. *Hay un conjunto de postulados teóricos T cuyos únicos términos no lógicos son de V_T.*
5. *Los términos de V_T tienen una definición explícita en términos de V_O mediante reglas de correspondencia C – esto es, para cada término 'F' en V_T debe darse una definición del mismo en la forma siguiente:*

    $$(x)\ (Fx \equiv Ox),$$

    donde 'Ox' es una expresión de L que contiene símbolos que sólo son de V_O más posiblemente el vocabulario lógico." [Suppe, 1974].

La premisa fundamental del positivismo lógico es que las leyes pueden ser verificadas empíricamente mediante la observación de fenómenos en el mundo real.

El problema fundamental del positivismo lógico es que depende de la inducción, tal y como recoge Blum:

> *"Aunque los eventos eran justificados desde las leyes y condiciones de forma deductiva, las leyes en sí mismas eran justificadas de forma inductiva*

> *(es decir empíricamente). El hecho de que una serie de eventos soporte una hipótesis dada no puede implicar que todos los eventos futuros también satisfagan esa hipótesis" [Blum, 1996].*

Como ejemplo de los problemas de la inducción está el ejemplo del "pollo inductivista" de Russel [Russell, 1953]. Este pollo, después de varios meses en la granja observa: "siempre me alimentan a las 9 de la mañana". Sin embargo, a las 9 de la mañana de Nochebuena, en vez de ser alimentado, le cortan la cabeza. Esta es la excepción que prueba que la regla inducida es falsa.

Este tipo de problemas con la inducción son el núcleo de la filosofía de Popper [Popper, 1959, 1965, 1974, 1985]. Al reconocer que lo único que se puede hacer es probar que una regla es falsa (es decir, falsarla), Popper propone una perspectiva en la que lo que se debería intentar es refutar hipótesis: el científico empieza haciendo conjeturas y luego trata de refutarlas. Si las refutaciones fallan y el científico es incapaz de falsar sus hipótesis, entonces su confianza en la validez de las hipótesis ha sido corroborada; esto es, la teoría corroborada es una candidata para ser aceptada como teoría verdadera. De esta forma, Popper reemplaza el empirismo de establecer la verdad con el empirismo de falsarla y su método científico se basa en la crítica:

> *"la ciencia empieza con mitos, y con la crítica de mitos; no empieza con la colección de observaciones ni con la invención de experimentos, sino con la discusión crítica de mitos y de técnicas y prácticas mágicas.*
>
> *La actitud crítica, la tradición de la discusión libre de teorías con el objetivo de descubrir sus puntos débiles de forma que puedan mejorarse, es la actitud de la racionalidad" [Popper, 1965].*

Dentro de esta línea de oposición al método de la inducción, destaca el pensamiento de Deutsch, quien en su libro "La estructura de la realidad" (en inglés, "*The Fabric of Reality*") [Deutsch, 1997], rechaza el inductivismo mediante argumentos elaborados y, en apariencia, irrefutables. Su idea fundamental es que la hipótesis inductiva es falsa porque no es cierto que las teorías se construyan a partir de la experiencia:

> *"ningún conocimiento y, lo que es más, ningún razonamiento, de la clase que sea, que haya resultado cierto, ha encajado en la descripción inductivista" [Deusch, 1997]*

Todo el trabajo en contra del inductivismo, pero a favor falsar hipótesis mediante la experimentación, deriva en una visión empirista del científico, basada en tres supuestos: un realismo ingenuo (todo puede ser conocido), un lenguaje científico universal (todo puede ser descrito formalmente) y la correspondencia de verdad (el científico observa la realidad desde el exterior de forma desapasionada) [Hesse, 1980].

Dentro de esta visión, el avance en el conocimiento científico se produce de forma evolutiva, mediante la selección natural de teorías: sobreviven aquellas que superan la refutación [Campbell, 1974] [Popper, 1965]. Sin embargo el mayor problema pendiente radica en los límites de la objetividad respecto a los datos.

La idea principal es que, en realidad, la observación no es objetiva, sino que está determinada por nuestro conocimiento previo: uno intenta encajar sus observaciones dentro de las teorías que maneja, que están reflejadas en nuestro lenguaje [Quine, 1961] [Quine, 1969].

> *"La teoría está no-determinada empíricamente. Incluso si tuviéramos un oráculo de observaciones, capaz de asignar verdad a cualquier informe de observación expresable en nuestro lenguaje, aún así esto no sería suficiente para elegir entre un conjunto de teorías físicas posibles, cada una de ellas completamente de acuerdo con el oráculo" [Quine, 1975].*

Por todo ello la ciencia empírica está débilmente justificada (*underjustified*), en el sentido de que los mismos datos pueden utilizarse para justificar más de una teoría [Campbell, 1974]. En otras palabras, la ciencia es "no-determinada" (*underdetermined*) y sólo el espíritu crítico (y no los hechos) pueden aportar luz al proceso.

> *"Por tanto, cuando el polvo finalmente se asienta, no encontramos la verdad, sino sólo una actitud crítica hacia lo que se considera como verdad" [Blum, 1996].*

A partir de aquí las teorías de la filosofía de la ciencia se han basado en estudiar el marco conceptual que afecta a nuestras observaciones (llamado Paradigma, Programa Científico, Tradiciones de Investigación o Suposiciones Guía) y en representar la evolución del conocimiento científico como un proceso de cambio de paradigmas mediante crisis generadas por un gran número de observaciones que no encajan con nuestra explicación del mundo, e incluso con nuestro lenguaje. Todo esto hace ver que no se puede separar el estudio de la ciencia de su relación con la sociedad: el científico no está aislado, pertenece a una sociedad que influye en su trabajo [Hoyningen-Huene, 1993] [Kuhn, 1970] [Lakatos, 1970] [Lakatos, 1978a] [Laudan et al., 1986].

Después de todo lo visto, queda claro que no hay una visión unificada de la ciencia. Para concluir se pueden usar las observaciones de Blum, que relaciona estos problemas con el desarrollo de software:

> *"En ciencia, el objetivo es encontrar una representación formal de algún aspecto de la realidad; en el desarrollo de software, el objetivo es ir desde una necesidad a un modelo computacional formal que responde a esa necesidad. En ambos casos, la realidad es compleja, y sólo podemos encontrar representaciones formales incompletas; pero estas representaciones deben ser formales si las queremos usar. La única alternativa, por tanto, es aceptar las limitaciones de nuestras teorías según trabajamos con ellas. En el dominio de la ciencia esto implica que las verdades eternas y universales pueden no existir" [Blum, 1996].*

Dentro de este problema, trabajos más recientes como [Lindley, 1993] han llevado a mostrar una influencia de la estética a la hora de elegir entre teorías distintas que son empíricamente ciertas. Como recoge Blum:

> *"Por lo tanto, el conocimiento científico, que representa el conocimiento humano* 'writ large'*, tiene su representación más pura en la física que, por su propia naturaleza, puede representarse matemáticamente y ser razonada de forma lógica. Pero la física, por la no determinación de la observación y la dificultad de llevar a cabo experimentos, parece ser guiada por estética en vez de por consideraciones empíricas (es decir, por lo subjetivo en vez de lo objetivo)" [Blum, 1996].*

Entonces, si todo en la ciencia se basa únicamente en el espíritu crítico y en la subjetividad (estética), ¿dónde radica su éxito? Según Blum, el éxito de la ciencia está en reconocer sus limitaciones:

> *"Empezamos el estudio de la ciencia esperando que pudiera proporcionar respuestas sólidas e irrefutables, y terminamos con la conclusión de que puede ser la estética de la respuesta lo que importe. Por supuesto, toda respuesta aceptable debe ser consistente con la realidad; pero la observación no está determinada, y muchas repuestas serán igualmente aceptables. Los científicos, por supuesto, reconocen este hecho, y por eso su 'empresa' tiene tanto éxito" [Blum, 1996].*

2.1.3 Modelado en la Ciencia

A pesar de no poder definir un método científico ideal, sí han existido métodos científicos que han dado resultado a lo largo de la historia. Ares y Pazos, en [Ares et al., 1998] proporcionan un resumen de los mismos que se recoge en la Tabla 2.1.

Tabla 2.1 Historia de la evolución de métodos científicos, según [Ares et al., 1998]

Tipo	Descripción
Descripción	Es el método más elemental, consistente en narrar pura y simplemente los hechos y fenómenos. Se usa para clasificar y formular problemas adecuadamente y para elegir el mejor método de resolución de problemas
Descripción comparativa	Evolucionando a partir de la descripción, revela relaciones mutuas e influencias entre fenómenos a través de comparaciones. Es muy eficiente cuando existen pocas correlaciones esenciales entre el sistema bajo estudio y el sistema relacionado.
Observación	Implica observar fenómenos que ocurren en la naturaleza de una forma analítica y crítica. Está enlazado a la descripción comparativa y mejora el conocimiento del fenómeno observado. Se le reprocha el ser pasivo, estéril y aleatorio, por lo que fue subvalorado, aunque puede ser utilizado para determinar el mejor instante de observación, seleccionar los recursos de investigación y tomar medidas para asegurar que el objeto bajo estudio permanece sin cambios.
Experimento cualitativo	Difiere del anterior en que los humanos intervienen para modificar las condiciones naturales o eliminar algunos factores. La observación y la descripción son parte de la experimentación y se usan para preparar experimentos y recoger resultados. Su ventaja es la posibilidad de reproducir eventos.
Experimento cuantitativo	Este método apareció mucho después porque requiere más recursos técnicos para la producción de fenómenos y su evaluación cuantitativa. Los resultados posibles son valores cuantitativos.
Modelado	A diferencia de los métodos anteriores, el modelado no opera directamente en los fenómenos bajo estudio. Los modelos se establecen basándose en unidades artificiales aisladas que dependen de un contexto específico. Son heurísticos, y éste es el método más avanzado.

Ares y Pazos concluyen que el método de modelado es el más adecuado para la visión actual de la ciencia, lo cual está de acuerdo con el resumen de la filosofía de la ciencia hecho en el apartado anterior: dado que la observación está sesgada por las teorías, se

trata de hacer modelos de la realidad que sean menos dependientes de dichas observaciones.

En este mismo sentido se expresa Blum, cuando dice:

> *"... el objetivo de la ciencia es producir modelos de la realidad. Estos modelos representan conocimiento acerca del universo (la realidad). ... los modelos de los que hablo son sólo una parte de la iniciativa científica, sin embargo son productos del proceso, y constituyen la descripción más completa de nuestra percepción de la realidad." [Blum, 1996].*

Pero, ¿qué es un modelo? Para definir este término conviene volver acudir al Diccionario de la Real Academia Española;

> ***modelo**. (Del it. modello). **1**. m. Arquetipo o punto de referencia para imitarlo o reproducirlo. **2**. m. En las obras de ingenio y en las acciones morales, ejemplar que por su perfección se debe seguir e imitar. **3**. m. Representación en pequeño de alguna cosa. **4**. m. Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento [RAE, 2001].*

La cuarta acepción es la que mejor se corresponde con la idea de modelo dentro de la ciencia: una simplificación teórica de la realidad, que sirve para facilitar su comprensión. En este sentido se expresa Andrade, cuando define modelo:

> *"Un modelo es una representación de un fenómeno del mundo real que, de alguna manera, es una simplificación de dicho fenómeno y que, sin embargo, se acepta como adecuada para determinados fines específicos. La simplificación es, en realidad, lo que hace atractivo al modelo" [Andrade, 2002].*

Aunque la forma más común de representar modelos en la ciencia son las matemáticas y la lógica, lo más importante del modelo es su relación con la realidad, como comenta Barwise:

> *"El método axiomático dice que nuestras teorías son ciertas si nuestros axiomas son ciertos. El método de modelado dice que nuestros teoremas modelan hechos en el dominio modelado si existe un ajuste suficientemente cercano entre el modelo y el dominio modelado. ... Como podría decir un filósofo, las matemáticas aplicadas pueden no garantizar el conocimiento de hechos sobre el mundo físico, pero pueden llevar a lo siguiente mejor – la creencia en una verdad justificada" [Barwise, 1989].*

Con el fin de tener una idea más clara de lo que es un modelo, se recoge seguidamente la definición descriptiva de modelo dada por Ares y Pazos:

> *"Dada una ocurrencia, objeto, fenómeno, evento, mecanismo, prototipo o sistema natural S, un objeto M es un modelo de S si:*
>
> 1. *S ha sido descompuesto de alguna manera en partes;*
> 2. *Una o más relaciones R entre elementos de S son análogas a las relaciones establecidas entre sus imágenes en M;*

3. *M permite generar datos que son similares a los datos generados en S. Esta tercera característica es esencial para que M sea una representación válida que permita examinar las propiedades de S" [Ares et al., 1998]*.

Un aspecto fundamental del modelado, con el que hay que tener especial cuidado, es con el problema de la falacia de identificación, expuesto por Barwise, y que consiste en pensar que el modelo es lo mismo que la realidad. Este problema es conocido en la ciencia como *pigmalionismo*, del mito griego de Pigmalión y Galatea [Seeman, 1968]. Blum recoge la definición de este problema en los siguientes términos:

> *"Barwise define la falacia de identificación como el fallo en distinguir entre un modelo matemático y la cosa de la cual es un modelo. En otras palabras, el modelo no es la realidad." [Blum, 1996].*

En esta afirmación están de acuerdo Ares y Pazos, cuando dicen:

> *"Las condiciones expresadas en la definición no significan que M sea un duplicado de S, ya que no estamos buscando una definición matemática, como un isomorfismo entre dos conjuntos abstractos. Aun más, los modelos más productivos son aquellos que tienen una relación homomórfica." [Ares et al., 1998]*.

Una vez definido lo que es un modelo, queda claro que es fundamental que sea útil para lograr un fin determinado. Ares y Pazos recogen los siguientes objetivos que puede tener el modelado dentro de la resolución de problemas [Ares et al., 1988]:

1. Organizar el conocimiento teórico y las observaciones empíricas sobre un problema.
2. Mejorar la comprensión del problema.
3. Apreciar la necesidad de mayor detalle.
4. Acelerar el análisis.
5. Construir un sistema de referencia para la aceptación de modificaciones en el sistema.
6. Ser más fácil de manejar que el sistema real.
7. Hacer posible controlar más fuentes de cambio de lo que sería posible observando directamente el sistema.
8. Recortar costes.

Así pues, el modelado es reconocido como el método científico más avanzado para conocer la realidad. Existen muchos tipos de modelos, tal y como recogen numerosos actores. De todos ellos los que interesan para esta Tesis son los modelos dedicados a conocer y comprender un problema antes de resolverlo, es decir, los modelos que conforman el conocimiento científico. Estos modelos son conocidos como modelos conceptuales o conceptualizaciones y se describen en el siguiente apartado.

2.1.4 La Conceptualización

Para comenzar este apartado se va a tratar de definir el término conceptualización. Una definición informal es la dada por Andrade:

> *"Modelado conceptual, conceptualización o análisis del problema son denominaciones dadas al proceso de modelado que lleva a cabo la persona que desea comprender el problema o la necesidad planteada para llegar a su resolución" [Andrade, 2002].*

Como ya se mencionó en el apartado 2.1.2, cuando se definió el saber según [Güell, 2001], lo importante es que el saber queda fijado en el ser humano pero que al mismo tiempo tiene una representación que permite su expresión y transmisión a otras personas.

Esta comprensión del mundo por parte de la persona es vista hoy en día como la conceptualización. Así, Heisenberg define lo que significa conceptualizar:

> *"Comprender algo significa penetrar realmente en sus interrelaciones, saber ciertamente que se conoce su funcionamiento interior. Entender quiere decir, muy en general, poseer representaciones (modelos) y conceptos mediante los cuales se puede captar una multitud de fenómenos como unitariamente coherentes y esto significa concebir o Conceptualizar" [Heisenberg, 1996].*

Esta conceptualización afecta a nuestra forma de ver el mundo, es decir, afecta a nuestra observación de los hechos, según las ideas ya descritas de Popper, Campbell y Quince, entre otros. En los siguientes párrafos se va a hacer un rápido recorrido por la historia de la conceptualización, basándose en la recopilación realizada, con mucho más detalle, en [Andrade, 2002].

Según recoge Andrade, la historia de la conceptualización tiene su origen en los griegos clásicos:

> *"La conceptualización de los conocimientos tiene una historia que se remonta a los griegos clásicos, en donde Empédocles de Agrigento [Empédocles et al., 1996], en cuanto a la Gnoseología o Teoría del Conocimiento, sostuvo una postura enantiodrómica - de fusión de Heráclito y Parménides [Heráclito et al., 1996] - según la cual se llega al conocimiento de la verdad tanto por la razón como por los sentidos." [Andrade, 2002].*

Mucho más tarde, en el siglo XIX, Frege creó su Ideografía o Conceptografía [Frege, 1879] que permite reflejar el pensamiento puro dando importancia al signo que representa el contenido. Para Frege lo primero es el concepto y los segundo el signo con el cual se representa.

Dentro de los trabajos realizados en el siglo XX, Andrade destaca la importancia de Einstein, muy preocupado por la relación entre los conceptos formados por el hombre y la realidad [Einstein, 1978]. En estos trabajos se constata cómo los conceptos que manejan interiormente las personas afectan en gran medida a su percepción de la realidad y a las conclusiones que derivan de los hechos ocurridos en el mundo.

> *"Einstein examina por una parte las impresiones de los sentidos y, por otra, los conceptos que pueden derivarse unos de otros por un razonamiento puramente especulativo o siguiendo las reglas estrictas de la lógica, a partir de conceptos iniciales que pueden ser arbitrarios. ... La lógica como tal no confiere a las proposiciones la verdad en el sentido de una*

> *concordancia con la realidad objetiva. ... Einstein ilustra este punto de vista cuando habla de la sorpresa que parece surgir cuando una experiencia viene a contradecir un mundo de conceptos que está suficientemente arraigado en el ser humano" [Andrade, 2002].*

Según Einstein, estas sorpresas son las que van haciendo que cambie nuestra concepción de la realidad (nuestras conceptualizaciones), haciendo que evolucione el conocimiento de la ciencia. Esto está de acuerdo con las teorías ya comentadas de la evolución del conocimiento científico [Hoyningen-Huene, 1993] [Kuhn, 1970] [Lakatos, 1970, 1978b] [Laudan et al., 1986].

Así pues una conceptualización es fundamental para comprender el mundo y, dados los mismos hechos, siempre habrá varias formas diferentes de conceptualizar. Teniendo esto en cuenta, Andrade define tres características importantes de la conceptualización:

> *"1. Una inconveniente conceptualización puede impedir resolver el problema.*
>
> *2. Una única conceptualización puede no ser suficiente para entender el problema.*
>
> *3. Algunas conceptualizaciones son más efectivas y, o, eficientes que otras." [Andrade, 2002].*

En primer lugar, una conceptualización puede no ser congruente con nuevas observaciones, lo que puede impedir comprender y resolver el problema. En este sentido Andrade habla del ejemplo de la conceptualización de la luz como una onda, que no permitía explicar completamente su comportamiento. Cabe destacar como apoyo de esta afirmación a Galileo, que en su "Diálogo de Galileo" [Galileo, 1968] da numerosos ejemplos de cómo aun observando dos personas los mismos hechos, distintos análisis conceptuales dan resultados totalmente diferentes, siendo el más conocido el experimento en el que se deja caer una piedra desde una torre para comprobar si la tierra se mueve o no. Como la piedra cae paralela a la torre, un aristotélico mantiene que la tierra no se mueve y, sin embargo, Galileo mantiene que sí lo hace (y con ella la torre y la piedra).

En segundo lugar, en algunos casos pueden ser necesarias varias conceptualizaciones simultáneas para poder comprender la realidad. Siguiendo con el ejemplo de la luz, Andrade explica cómo la combinación de las teorías "luz como onda" y "luz como materia" permitió explicar adecuadamente el comportamiento de la luz.

Por último, Andrade destaca que, en función del fin perseguido, algunas conceptualizaciones serán más efectivas y eficientes que otras, para lograr ese objetivo de la forma más adecuada posible.

Para finalizar este apartado conviene presentar una definición más formal de la conceptualización y sus características, que complemente la definición informal dada por Andrade.

Siguiendo a autores del campo de la Ingeniería del Conocimiento como [Genesereth et al., 1986], [Gómez et al., 1997] y [Ares et al., 1998], la conceptualización puede definirse como una terna (C, R, F) formada por tres conjuntos:

- *Conceptos*. Un concepto es una idea general o abstracta, con frecuencia obtenida a partir de la generalización sobre instancias específicas, y expresada a través de términos lingüísticos. El conjunto de todos los conceptos empleados en un modelo conceptual recibe el nombre de universo de discurso.
- *Relaciones*. Una relación entre conceptos es un subconjunto del producto cartesiano de esos conceptos. Las relaciones se pueden definir por extensión (enumerando todos su elementos) o por comprensión (definiendo condiciones que deben cumplir). En [Hammer, 1969] puede encontrarse una descripción completa de todos los tipos de relaciones que pueden manejarse. Por otro lado, en [Ares et al., 1998] se recogen las relaciones más habituales que aparecen en los modelos conceptuales orientados al desarrollo de aplicaciones informáticas.
- *Funciones*. Son un caso especial de relaciones que deben ser consideradas de forma separada, según muchos autores como [Gómez et al., 1997], [Ares et al., 1998] y [Andrade, 2002]. Suelen utilizarse como forma de representar las operaciones realizadas sobre el estado del sistema.

Dentro de esta definición es importante destacar que tanto las relaciones como las funciones sólo pueden definirse entre los elementos del universo de discurso.

Ahondando en el significado de la conceptualización y, más en concreto, en el significado de sus elementos fundamentales, los conceptos, Díez y Moulines proporcionan en [Díez et al., 1997] una enumeración de sus características más importantes, llamadas hipótesis de la conceptualización y que se resumen a continuación (puede encontrarse una explicación más detallada en [Andrade, 2002]):

1. *Entidades abstractas*. Los conceptos son entidades abstractas, en principio identificables, a las que tienen acceso los seres humanos en tanto sujetos epistémicos y que le permiten a éstos conocer el mundo real y orientarse en él.
2. *Contraposición de un sistema de conceptos al mundo real*. Los conceptos permiten identificar, diferenciar, comparar, etc., los objetos de los que consta el mundo real. Todo objeto cae bajo algún concepto (un objeto queda subsumido bajo un concepto) pero puede haber conceptos "vacíos" sin correspondencia en el mundo real ("habitante del sol").
3. *Conexión entre un sistema de conceptos y otro lingüístico*. Existe una conexión íntima entre un sistema de conceptos y un sistema lingüístico: los conceptos deben identificarse con palabras o expresiones de un lenguaje dado. Podrían existir conceptos no expresables, pero normalmente no es así.
4. *Expresión de conceptos por términos no-sincategoremáticos*. Se refiere a que para identificar conceptos se usan términos que no son sincategoremáticos (es decir, términos categoremáticos que son palabra que tienen sentido por sí mismas). Además, estos términos suelen ser predicados de uno o más parámetros (n-ádicos), por lo que se pueden aplicar las facilidades proporcionadas por la lógica de predicados.
5. *Necesidad de la teoría de conjuntos*. En ocasiones es útil sustituir el tratamiento de los propios conceptos (o de los predicados que los expresan) por el de las extensiones de los mismos, esto es, por el de los conjuntos de objetos que caen bajo

cada concepto. Así, se define la extensión de un concepto como el conjunto de objetos del mundo real que caen bajo ese concepto. Al disponer de conjuntos, se pueden aplicar los principios y operaciones de la Teoría de Conjuntos [Cantor, 1962] y establecer o revelar indirectamente determinadas conexiones entre los conceptos que tienen tales extensiones.

Con esto termina la exposición del tratamiento del conocimiento y del análisis del problema (también llamado modelado conceptual o conceptualización) fuera de la informática. Ha llegado el momento de centrarse en el ámbito de trabajo de esta Tesis, es decir, el análisis del problema dentro del desarrollo de aplicaciones informáticas.

2.2 EL ANÁLISIS EN EL DESARROLLO DE SOFTWARE

Como punto de partida para estudiar el análisis del problema dentro del desarrollo de software, se va a tomar el proceso esencial del software definido por Blum y ya comentado en la introducción (véase la figura 1.1 en el capítulo 1. Introducción).

Dentro de este proceso esencial del software Blum define dos actividades fundamentales, el modelado conceptual y el modelado formal:

> *"El **Modelado Conceptual** es la actividad que establece una respuesta realizable por el software de una necesidad de aplicación. La orientación del modelo conceptual es la del dominio templada por las demandas del proceso de desarrollo; aunque se modelan conceptos del software, el objetivo del modelo es que sea comprensible para los promotores y usuarios.*
>
> *El **Modelado Formal** es formal en el sentido de las matemáticas y la lógica. Es más que sólo preciso y no ambiguo; es deseable que el esquema de representación sea procesable por el computador para identificar errores" [Blum, 1996].*

La perspectiva de Blum de considerar el análisis del problema como una actividad de modelado es coincidente con la tendencia existente desde mediados de los años 90 dentro de las disciplinas de la Ingeniería del Software y de la Ingeniería del Conocimiento [Chan et al. 1995], [Nissen et al. 1996], [Rumbaugh 1997a], [Rumbaugh, 1997b], [Schreiber et al. 1994], [Snowdon 1996].

Esta visión del proceso software es compatible con la visión del análisis del problema desde la perspectiva de la ciencia, ya que el modelado conceptual es precisamente el estudio del problema que se pretende automatizar.

Lo que Blum llama modelado conceptual (el estudio del problema) es lo que los autores de la disciplina de la ingeniería del software llaman análisis del problema [Davis, 1993] y los autores de ingeniería del conocimiento suelen llamar conceptualización [Buchanan et al., 1983], [Wielinga et al., 1992].

Es de destacar la importancia que se da a esta fase del desarrollo de software en la literatura de la disciplina. A modo de muestra se recogen algunas citas:

> *"El modelado conceptual en el desarrollo de software se ha ido presentando con el discurrir de la historia de la Informática como una actividad crucial, ya que no se puede prescindir de ella en unos entornos*

> *cada vez más heterogéneos, con el dominio del problema cada vez menos familiar para los desarrolladores y con sistemas software cada vez más complejos en su concepción como los actuales" [Andrade, 2002].*
>
> *"la importancia de comprender y modelar el problema, ya que es una condición necesaria y a veces suficiente para desarrollar software de buena calidad" [Ares et al., 1998].*
>
> *"Creo que la parte difícil de construir software es la especificación, diseño y pruebas de este modelo conceptual" [Brooks, 1987].*
>
> *"Identificar y establecer los requisitos, de forma correcta y precisa es crítico para producir software de buena calidad" [Fowler et al., 1998].*
>
> *"posiblemente el factor más crítico en el desarrollo de un sistema software sea éste: entender y representar adecuadamente lo que ha de satisfacer el sistema a desarrollar" [Jackson, 1995a].*

A pesar de esta importancia reconocida y del nombre de análisis del problema, en la práctica habitual del desarrollo de software, ese estudio del problema ha estado fuertemente sesgado por la influencia de las técnicas de diseño e, incluso, de programación. Como sustento de esta aseveración en la literatura pueden encontrarse multitud de citas como las siguientes:

> *"tradicionalmente, la Ingeniería del Software se ha centrado en soluciones de desarrollo y ha puesto una ínfima o nula atención en los problemas que los sistemas ayudan a resolver y su conceptualización. Incluso los marcos que consideran el término 'Análisis del Problema' normalmente atienden a aspectos relativos a la solución." [Andrade, 2002].*
>
> *"se ha hecho poco o ningún hincapié en lo que parece el corazón de la buena práctica en el desarrollo de software: el Problema, su comprensión y conceptualización" [Ares et al., 1998].*
>
> *"a menudo afrontamos los problemas desde la perspectiva del computador en vez de la perspectiva del problema" [Blum, 1996].*
>
> *"el análisis de esta aproximación [la orientación a objetos] está muy afectado por aspectos de implementación" [Bonfatti et al., 1994].*
>
> *"Uno de los principales inconvenientes [de la orientación a objetos] es que está orientada a la implementación y no al problema" [Høydalsvik et al., 1993].*
>
> *"Los DFD son dibujos imprecisos sugiriendo lo que alguien piensa que puede ser la forma de un sistema para resolver un problema, pero no dice cuál es el problema" [Jackson, 1995b].*
>
> *"Tradicionalmente, el pensamiento e investigación en el desarrollo de software se ha centrado en las soluciones: en los programas y las diferentes abstracciones que pueden ser útiles en el diseño y escritura de un programa. Se ha puesto muy poca o ninguna atención en los problemas que esos programas ayudan a resolver. Incluso los métodos y aproximaciones que reclaman para sí el título de "análisis del problema", tras un estudio de los mismos, normalmente se comprueba que manejan soluciones. El problema a ser resuelto no se establece detalladamente ni se analiza*

> *explícitamente; el lector debe inferir el problema a partir de su solución." [Jackson, 2001a]*.

Por lo tanto parece claro que la visión tradicional del análisis del problema dentro del desarrollo de software es una visión constreñida por una perspectiva de la solución: no se trata de estudiar el problema planteado en términos del problema, sino en términos de la tecnología utilizada para resolverlo.

Una excepción a esta regla son los métodos basados en la especificación formal de requisitos [NASA, 1997] [NASA, 1998], que pretenden especificar las características del problema usando términos que le son propios, expresados mediante la aplicación de notaciones formales de la matemática y la lógica, aunque como se explicará más adelante, presentan otros tipos de inconvenientes.

Teniendo en cuenta todo lo anterior, y sin ánimo de ser exhaustivos, se podrían identificar tres enfoques tradicionales en el análisis del problema dentro del desarrollo de software:

- *Métodos basados en el paradigma de programación*. Estos métodos analizan el problema en términos fuertemente influenciados por los paradigmas de programación que se utilizarán en la etapa de implementación. Dentro de esta categoría se sitúan las metodologías estructuradas, las orientadas a objetos y las orientadas a agentes.
- *Métodos basados en formalismos matemáticos*. Estos métodos analizan el problema de manera formal usando el lenguaje de las matemáticas. Esta categoría recoge los métodos formales de especificación.
- *Métodos basados en el paradigma del conocimiento*. Estos métodos provienen de la Ingeniería del Conocimiento y están muy centrados en el tipo de problemas al que van dirigidos: los problemas que requieren un manejo intensivo de conocimientos (en inglés, *knowledge intensive problems*). Dadas las características de estos problemas, fueron los primeros en poner un énfasis especial en estudiar y comprender el problema, perspectiva que luego ha sido adoptada por la Ingeniería de Requisitos. Esta virtud de ser métodos orientados a los conocimientos se ha convertido con el tiempo en un ligero sesgo en el tipo de análisis que se efectúa en muchas de estas metodologías, como CommonKADS [Schreiber et al., 1999], que está dirigido por los conocimientos, generándose una dependencia entre las actividades y modelos de análisis y el paradigma al que van dirigidos, que en muchos casos implica una arquitectura específica de programa.

Por último existe una nueva categoría, de reciente creación y aún no bien definida, formada por los métodos que ponen énfasis en el problema evitando en lo posible utilizar los términos del diseño y el exceso de formalidad.

Los siguientes apartados de este capítulo se van a dedicar describir brevemente la forma en la que las metodologías más relevantes de cada una de estas cuatro categorías abordan la fase de análisis. Este estudio estará especialmente centrado en las etapas y notaciones relacionadas con la fase de estudio del problema y servirá para establecer una base para su posterior discusión.

2.3 ANÁLISIS BASADO EN PARADIGMAS DE PROGRAMACIÓN

Dentro de este apartado se van a considerar las fases de análisis de metodologías que fueron creadas como respuestas a paradigmas de diseño y programación. Esto provoca que sus fases de análisis estén dirigidas a situar el problema en términos de la solución.

Dentro de este grupo pueden destacarse, por su importancia en la historia de la Ingeniería del Software, *tres paradigmas*:

- El *paradigma estructurado*, basado en que un programa se compone de una serie de procedimientos que manejan unos datos comunes.
- El *paradigma orientado a objetos*, basado en que un programa se compone de una serie de objetos, cada uno de los cuales agrupa datos y procedimientos, que colaboran para resolver el problema.
- El *paradigma orientado a agentes*, basada en la descomposición del programa en agentes con objetivos, creencias y motivaciones, que resuelven el problema de forma conjunta.

2.3.1 Metodologías Estructuradas

Estas metodologías tienen su origen en los años 70 y están basadas en los principios básicos de la programación estructurada [Parnas, 1972], [Wirth, 1971]. Su principio básico consiste en representar un programa como:

- Una serie de *procedimientos* que realizan el trabajo solicitado.
- Una serie de *datos*, que son manejados por los procedimientos durante la realización del trabajo solicitado. Los datos son independientes de los procedimientos y es el programador el que debe decidir qué datos usar en cada instante.

El proceso de desarrollo de software bajo esta perspectiva está dirigido por las siguientes ideas fundamentales, todas ellas relacionadas con técnicas de programación:

- *Refinamiento progresivo* [Wirth, 1971] y su correspondiente tratamiento descendente del problema.
- *División en Módulos* [Stevens et al., 1994], con la identificación de las características de bajo acoplamiento y alta cohesión para definir una organización adecuada de un programa.
- *Ocultación de información* [Parnas, 1972], en la que los módulos inferiores ocultan detalles que no deben ser conocidos por los superiores.
- *Énfasis en estructuras de control cerradas* [Dijkstra, 1968] con el fin de facilitar la comprensión y posible verificación de los programas.

Las metodologías estructuradas tradicionales son las descritas en [DeMarco, 1979], [Gane et al., 1988] y [Yourdon, 1989].

2.3.1.1 La Metodología Estructurada Clásica

Dentro de las metodologías de esta línea se va a describir como más representativa la metodología propuesta por Yourdon [Yourdon, 1989]. La descripción de la metodología de Yourdon realizada aquí está basada en la recogida en [Alonso et al., 2002].

Bajo esta metodología el problema se puede contemplar como un conjunto diferente de datos de entrada (fuentes) que sufren una serie de transformaciones como consecuencia de diferentes procesos que actúan sobre los datos, obteniéndose como resultado unos flujos de salida que reciben los receptores de la información (sumideros). A esta metodología se la denomina como "metodología estructurada orientada al flujo de datos", o simplemente "metodología estructurada".

El ciclo de vida de un proyecto orientado al flujo de datos consta de nueve etapas o actividades que se describen en la Figura 2.1 [Yourdon, 1989].

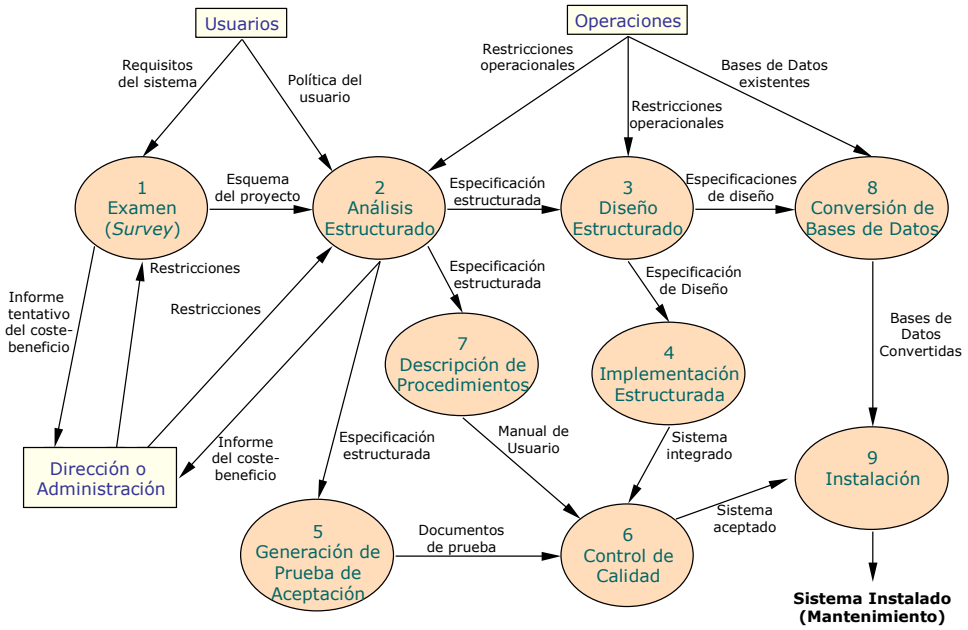


Figura 2.1 Etapas de la Metodología Estructurada de Yourdon

En líneas generales, la función que tiene cada una de estas actividades es:

1. *Examen* (en inglés, *Survey*).- Se inicia cuando el usuario solicita automatizar una parte o todo su sistema de negocio. Esta actividad se conoce como estudio de viabilidad o estudio inicial del sistema.

2. *Análisis Estructurado*.- El fin de esta actividad es transformar las políticas del usuario y el esquema del proyecto, en una especificación estructurada. Esto implica modelar el sistema desde tres puntos de vista: analizando las características externas del sistema, las características internas y las peculiaridades de instalación del usuario. Ello proporciona un modelo del sistema que describe los requerimientos del usuario. Adicionalmente, se prepara un presupuesto y se calculan costes y beneficios de un modo preciso y detallado.

3. *Diseño Estructurado*.- Esta actividad se dedica a la creación de una jerarquía apropiada de módulos de programas y de interfaces entre ellos para implantar las especificaciones producidas en la fase de análisis. Se ocupa también de transformar

los modelos de datos de entidad-relación en unas estructuras de datos o de bases de datos, según proceda. Previamente a todo ello, es necesario asignar las especificaciones estructuradas entre los diferentes procesadores que componen el sistema, así como las tareas que va a realizar cada procesador.

4. *Implementación Estructurada*.- Esta actividad incluye la codificación de cada módulo y su integración en un esqueleto progresivamente más completo del sistema final. En esta metodología la programación será estructurada y la integración descendente. Una etapa interesante de la implementación son las pruebas de cada módulo, y del sistema integrado.

5. *Generación de las Pruebas de Aceptación*.- El sistema ha de ser aceptado por el usuario. Por tal motivo, a partir de las especificaciones estructuradas del sistema, el analista produce un conjunto de casos de prueba que tendrá que pasar satisfactoriamente. Como las pruebas de aceptación se pueden desarrollar en paralelo con las actividades de diseño e implementación, es normal que esta actividad la inicie el analista nada más finalizar la actividad de "Análisis Estructurado".

6. *Garantía de Calidad o Control de Calidad*.- Se conoce esta actividad como prueba final o prueba de aceptación. Requiere como entradas los datos de las pruebas de aceptación y el sistema integrado producido en la actividad 4. La prueba la realizará algún miembro o departamento del usuario, o incluso un departamento independiente de control de calidad. Interesa señalar que es importante realizar actividades de control de calidad en cada una de las actividades anteriores de análisis, diseño e implementación para asegurar que se han realizado con un nivel apropiado de calidad. Así se asegura que el analista está desarrollando especificaciones de calidad, que el diseñador está produciendo diseños de calidad y que el programador está codificando programas de calidad. La actividad de Control de Calidad es simplemente la prueba final de la calidad del sistema.

7. *Descripción de Procedimientos*.- Esta actividad tiene como finalidad realizar una descripción formal de las partes del sistema que se desarrollarán manualmente, y una descripción de la interacción del usuario con el nuevo sistema informático. El resultado producido es el manual del usuario.

8. *Conversión de Bases de Datos*.- Esta actividad tiene lugar si existen bases de datos en el sistema de negocio actual del usuario, que sea necesario convertir para que puedan ser operadas por el nuevo sistema. Requiere como entrada las bases de datos existentes y las especificaciones de diseño producidas en la actividad 3.

9. *Instalación*.- Es la actividad final del desarrollo del software. Consiste en instalar el sistema, ya aceptado por el usuario durante la actividad 6, en la máquina del usuario para su utilización. El proceso de instalación puede ser, en algunos casos, una actividad inmediata de corta duración, mientras que en otros, exige un proceso incremental de forma que los usuarios se vayan incorporando y adaptando poco a poco al nuevo sistema. Es usual que durante unos meses funcionen en paralelo ambos sistemas, el viejo y el nuevo, para contrastar la bondad de este último. Las entradas de la actividad de instalación son el manual de usuario producido en la

actividad 7, las bases de datos convertidas en la actividad 8 y el sistema aceptado en la actividad 6.

Dada esta descripción de actividades, queda claro que la etapa que interesa dentro del ámbito de esta Tesis es la segunda: el análisis estructurado.

El análisis estructurado tiene por objeto desarrollar tres modelos: ambiental, de comportamiento y de implementación del usuario.

- *Modelo Ambiental*, que modela el exterior del sistema a partir de tres componentes:
  - La *declaración de objetivos*. Consiste en una declaración textual, breve y concisa del propósito del sistema a nivel usuario; es decir, planteada desde el punto de vista de usuario y no del desarrollador informático.
  - El *diagrama de contexto*. Es un diagrama de flujo de datos (DFD) en el que se reflejan:
    - Las entidades externas con las que se comunica el sistema.
    - Los datos que el sistema recibe del exterior para su proceso y los datos que produce y envía al exterior.
    - Los almacenes de datos externos al sistema, que se crean fuera del sistema y él los utiliza o que los crea el propio sistema y son usados fuera.
    - Las fronteras entre el sistema a desarrollar y el resto del mundo.
  - La *lista de sucesos*. Es una lista de los eventos externos al sistema a los cuales debe responder.
- *Modelo de Comportamiento*. Tiene por objeto describir el comportamiento que debe tener el sistema para dar respuesta al "modelo ambiental", para lo cual se definen tres componentes:
  - Un *modelo de proceso* que consta de un conjunto de DFD desarrollados por niveles, un diccionario de datos y las especificaciones de los procesos del nivel inferior.
  - Un *modelo de datos* definido mediante un diagrama Entidad-Relación (E-R) [Chen, 1976].
  - Un *diagrama de transición de estados*, para reflejar la reacción ante los eventos.
- *Modelo de Implantación del Usuario*. Recoge detalles importantes de implantación que el usuario debe aprobar antes de que se inicie el desarrollo del sistema y que deben ser definidos en la fase de análisis. Estos detalles comprenden los siguientes puntos:
  - Establecimiento de los límites hombre-máquina.
  - Definición de la interfaz de usuario.
  - Actividades adicionales manuales que requiere el sistema.
  - Restricciones Operativas del sistema.

Para el desarrollo de los tres modelos que componen el análisis estructurado se utilizan las siguientes técnicas básicas:

- *Diagramas de Flujo de Datos* (DFD) [Stevens et al., 1974], [Yourdon, 1975]. El DFD es una técnica gráfica que representa el flujo de información y las transformaciones que se aplican a los datos al moverse desde la entrada a la salida. También es conocido como "grafo de flujo de datos" o "diagrama de burbujas". Los diagramas básicos que se utilizan para crear un DFD son [Yourdon, 1989]: (Figura 2.2):
  - *El proceso*. Se representa gráficamente con un círculo y muestra una parte del sistema que transforma unos datos de entrada en salidas. Se nombra o describe con una palabra o frase situada en el interior del círculo que indica lo que hace.
  - *El flujo*. Se representa gráficamente por medio de una flecha que entra o sale de un proceso. Se usa para describir el movimiento de información de una parte del sistema a otra; es decir, el flujo representa datos en movimiento. Lleva un nombre que especifica el tipo de información que se mueve a lo largo del flujo.
  - *El almacén*. Se representa con dos líneas paralelas y sirve para modelar un conjunto de datos en reposo (ficheros o registros en memoria) que es utilizado por uno o varios procesos. Lleva un nombre entre las dos líneas que especifica la información almacenada.
  - La *entidad externa*. Se representa con un rectángulo y muestran entidades externas con las cuales se comunica el sistema. Suele ser una persona, un grupo, una organización u otro sistema con el cual se relaciona.

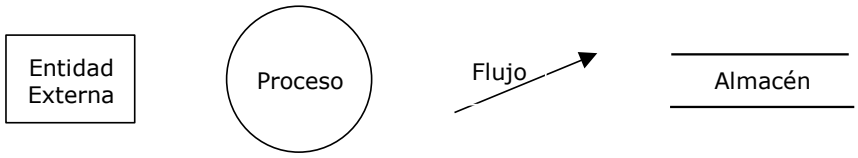


Figura 2.2 Elementos básicos de un DFD

Normalmente, los DFD se van refinando en niveles que representan un mayor flujo de información y un mayor detalle funcional. Inicialmente, todo el sistema se representa como una única burbuja reflejando todo el proceso donde figuran todas las unidades externas que participan en el mismo. A este DFD de nivel 0 se le denomina "modelo fundamental del sistema" o "diagrama de contexto".

El DFD de contexto consta de una sola burbuja que representa el sistema completo. Se pasa al siguiente nivel expandiendo la burbuja en procesos de tipo inferior que definen el sistema. Estos procesos se enumeran y el DFD resultante constituye el nivel 1. Las burbujas del nivel 1 se vuelven a expandir en procesos más pequeños obteniéndose los DFD de nivel 2. El refinamiento de los DFD continúa hasta que cada burbuja representa una función sencilla que se pueda implementar fácilmente en un lenguaje de programación (Figura 2.3).

- *Especificaciones de Proceso*. Sirven para describir cada uno de los procesos (burbujas) del DFD en el modelo de proceso (dentro del modelo de comportamiento) y deben expresarse de forma tal que puedan ser verificadas tanto

por el usuario como por el analista. Estas especificaciones se pueden describir de diversas formas: mediante una narrativa textual, con ecuaciones matemáticas, con tablas de decisión, etc. No obstante, es usual emplear un lenguaje estructurado, con sentencias en castellano y utilizando las estructuras de control que emplean los lenguajes de programación (if-then-else, do, while, case, ...).

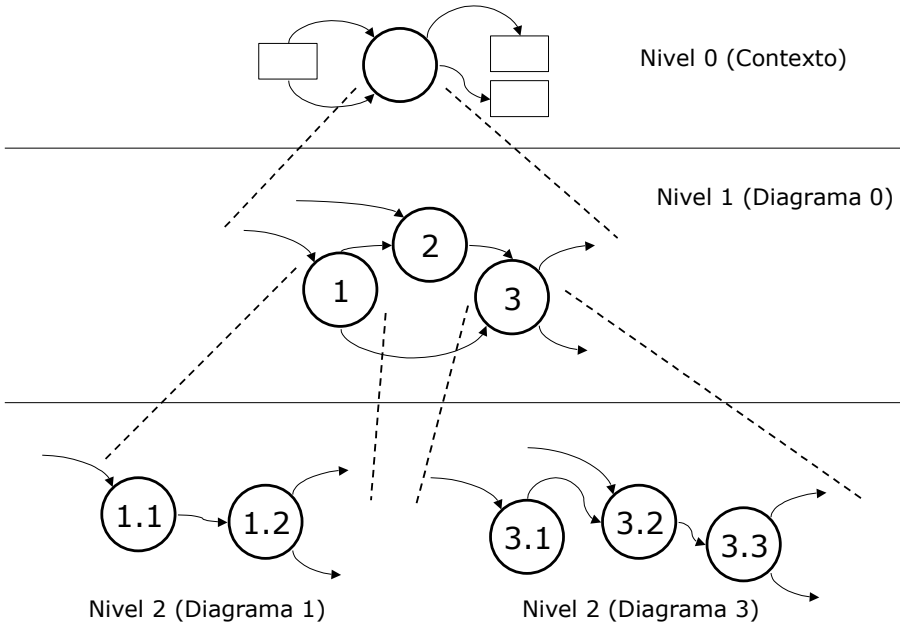


Figura 2.3. Refinado por niveles de los DFD

- *Diccionario de Datos*. Es un listado organizado de todos los elementos de datos que son pertinentes para el sistema, con definiciones precisas y rigurosas que permiten que el usuario y el analista del software tengan una misma comprensión de las entradas, de las salidas, de los componentes de los almacenes y de los cálculos intermedios [Yourdon, 1989]. Normalmente, cada entrada al diccionario contiene para cada dato la siguiente información [Pressman, 1997]:
  - *Nombre*. Nombre principal del dato, fichero o entidad externa.
  - *Alias*. Otros nombres usados para ese dato.
  - *Dónde se usa/cómo se usa*.- Listado de los procesos que utilizan ese dato y cómo lo usan (por ejemplo, como entrada o salida del proceso, como almacén, como entidad externa).
  - *Descripción del contenido*. Los datos compuestos se representan mediante la notación recogida en la Tabla 2.2.
- *Diagrama Entidad – Relación* [Chen, 1976]. El diagrama de entidad-relación, también conocido como diagrama E-R, es un modelo en red que describe la distribución de datos almacenados en el sistema. Los componentes fundamentales que participan en este tipo de diagramas son dos (Figura 2.4):
  - *Entidad o Tipo de Objeto*. Se representan por un rectángulo y especifican un conjunto de objetos cuyos miembros individuales (instancias) se identifican de

una manera única, juegan un papel necesario en el sistema y se describen por una serie de atributos.

Tabla 2.2 Notación para el diccionario de datos [Alonso et al., 2002]

Notación	Significado
=	Está compuesto de varios datos
+	Secuencia de datos. Conjunción (y)
[|]	Selección entre un conjunto de datos. Disyunción (o)
$\{\}^n$	Agrupación repetida n veces. Iteración.
( )	Datos opcionales
* ... *	Comentarios
@	Identificador (campo clave) para un almacén

- *Relaciones*. Los objetos se conectan entre sí mediante relaciones y se representan por medio de un rombo. La relación representa un conjunto de conexiones y cada instancia de la relación representa una asociación entre una ocurrencia de un objeto y una ocurrencia del otro.

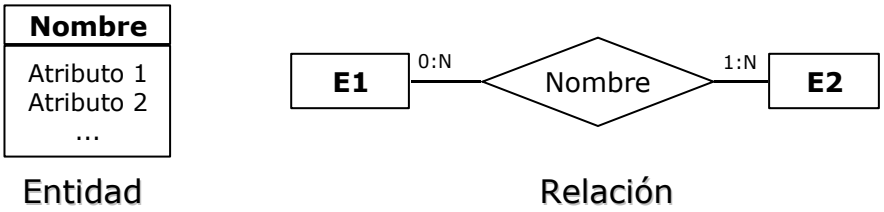


Figura 2.4 Notación de los diagramas E-R

- *Diagramas de Transición de Estados*, que relacionan sucesos y estados. Cuando ocurre un suceso, el sistema pasa a un nuevo estado dependiente del estado actual y del suceso producido. Un cambio de estado causado por un proceso se denomina transición. Un diagrama de transición de estados es un grafo cuyos nodos son estados y cuyos arcos son transiciones etiquetadas con los nombres de los eventos y de las acciones originadas por esos eventos. La Figura 2.5 recoge la notación de estos diagramas.

Figura 2.5 Notación de los Diagramas de Transición de Estados

En resumen se puede constatar que las metodologías estructuradas se basan en la combinación de una serie de técnicas diversas que permiten representar la dualidad procedimientos – datos:

- Los *procedimientos* se reflejan como procesos en los DFD, con su correspondiente especificación. Estos procedimientos deben realizarse como reacción a los eventos del diagrama de transición de estados.
- Los *datos* se reflejan como flujos y almacenes en los DFD, como entradas en el diccionario de datos y, finalmente, como un modelo E-R completo.

Por otro lado, hay una parte del análisis que queda reflejada usando únicamente el lenguaje natural, fundamentalmente en lo que respecta al modelo de implantación del usuario.

2.3.1.2 Metodologías Estructuradas Avanzadas

Existen muchas variantes de metodologías estructuradas, que han ido evolucionando a lo largo del tiempo. Entre ellas destacan las metodologías definidas de forma normativa dentro de las administraciones públicas de varios países, como:

- SSADM, y su evolución llamada BSD (del inglés *Business System Development*, Desarrollo de Sistemas para el Negocio) [OGC, 2000] [OGC, 2001]. Esta metodología es la utilizada por la administración pública del Reino Unido.
- MERISE (del francés *Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise*, Método de Estudio y de Desarrollo Informático para los Sistemas de Empresa), metodología definida en la administración pública de Francia [Gabay, 1991].
- MÉTRICA, la metodología definida por la administración pública de España [MAP, 2001].

Todas ellas comparten los mismos principios básicos y técnicas fundamentales ya descritos para la metodología de Yourdon, y por tanto poseen sus mismas características básicas.

La mayoría han visto ampliadas sus raíces estructuradas y ahora admiten, como opción dentro del desarrollo, la utilización de técnicas orientadas a objetos, que están basadas en las metodologías descritas en el apartado siguiente.

2.3.2 Metodologías Orientadas a Objetos

Las metodologías orientadas a objetos fueron desarrolladas como consecuencia de la aplicación del paradigma de programación orientado a objetos [Meyer, 1988]. Este paradigma tiene sus orígenes a finales de los años setenta con el lenguaje de programación Simula 67 y la definición posterior de Smalltalk [Lalonde et al., 1990] [Morales et al., 1993].

Los elementos fundamentales que configuran el Paradigma Orientado a Objetos, algunos autores [Khoshafian et al., 1990] los centran en tres: Los Tipos Abstractos de Datos, La Herencia y la Identidad de los Objetos. Mientras que otros [Meyer, 1988] señalan siete aspectos básicos a considerar para una verdadera orientación al objeto:

- *Estructura modular basada en objetos*, dado que los sistemas en esta metodología son modularizados sobre la base de sus estructuras de datos.
- *Abstracción de Datos*, porque los objetos se describen como implementaciones de tipos de datos abstractos.
- *Gestión automática de memoria*, de forma que los objetos no utilizados sean liberados por el propio sistema sin intervención del programador.
- *Clases*, en las que cada tipo abstracto no simple sea un módulo.

- *Herencia*, que permita que una clase sea definida como una extensión o restricción de otra.
- *Polimorfismo y enlace dinámico*, de forma que las entidades del programa puedan referenciar en tiempo de ejecución a objetos de diferentes clases.
- *Herencia múltiple* para que se pueda declarar una clase como herencia de varias.

A partir de las ideas de la programación orientada a objetos, fueron surgiendo metodologías que aprovecharan sus principios. En primer lugar se trataba de métodos de diseño orientado a objetos, como [Abbot, 1983] [Booch, 1986] [Coad et al., 1991] [EVB, 1986] [Lorensen, 1986] [Meyer, 1988], para pasar con posterioridad a definirse el análisis orientado a objetos [EVB, 1989] [Cashman, 1989] [Coad et al., 1990] [Booch, 1994] [Martin et al., 1992].

Hay una gran variedad de metodologías orientadas a objetos. De hecho, ya en 1996 Henderson-Sellers llegó a identificar más de 80 [Henderson-Sellers, 1996a] y este número no ha dejado de crecer desde entonces.

Aun así ha habido un gran esfuerzo de convergencia de este tipo de metodologías, destacándose dos resultados como más relevantes: el Proceso Unificado [Jacobson et al., 1999] y OPEN [Graham et al., 1997a]. Éstas serán las metodologías descritas en los apartados siguientes.

2.3.2.1 El Proceso Unificado y UML

El Proceso Unificado [Jacobson et al., 1999] [Kruchten, 2000] [Rational, 2002] es una metodología de desarrollo de software que abarca el proceso completo, incluyendo la gestión del mismo. Esta metodología utiliza como notación fundamental UML (del inglés *Unified Modeling Language*, que significa Lenguaje Unificado de Modelado) [Booch et al., 1999] [OMG, 2003]. La descripción que aquí se recoge está basada en la realizada en [Alonso et al., 2003].

En esta metodología de desarrollo del ciclo de vida de un proyecto software, cada iteración (o ciclo) trata seis Flujos de Trabajo Fundamentales o Disciplinas de Ingeniería: “modelado de negocio, captura de requisitos, análisis y diseño, implementación, pruebas y despliegue” y tres de Soporte: "configuración y administración de cambios, gestión de proyectos, y ambiente"; y concluye con una versión del producto para presentar al cliente en un estado cada vez más elaborado.

Una nueva iteración produce una nueva versión, que incorpora en el software más funcionalidad y es más refinada. Este desarrollo incremental de la aplicación se presenta en cuatro fases: “Comienzo (del inglés *Inception*), Elaboración, Construcción y Transición”. Cada fase termina con un hito que plantea una serie de objetivos que se han tenido que alcanzar. Por lo que al finalizar cada hito la dirección del proyecto tiene que decidir si el trabajo puede continuar con la siguiente fase.

La Figura 2.6 presenta el Proceso Unificado a través de sus Flujos de Trabajo, Iteraciones y Fases de Desarrollo.

Seguidamente se describen las disciplinas fundamentales, es decir, aquellas dedicadas a labores de desarrollo:

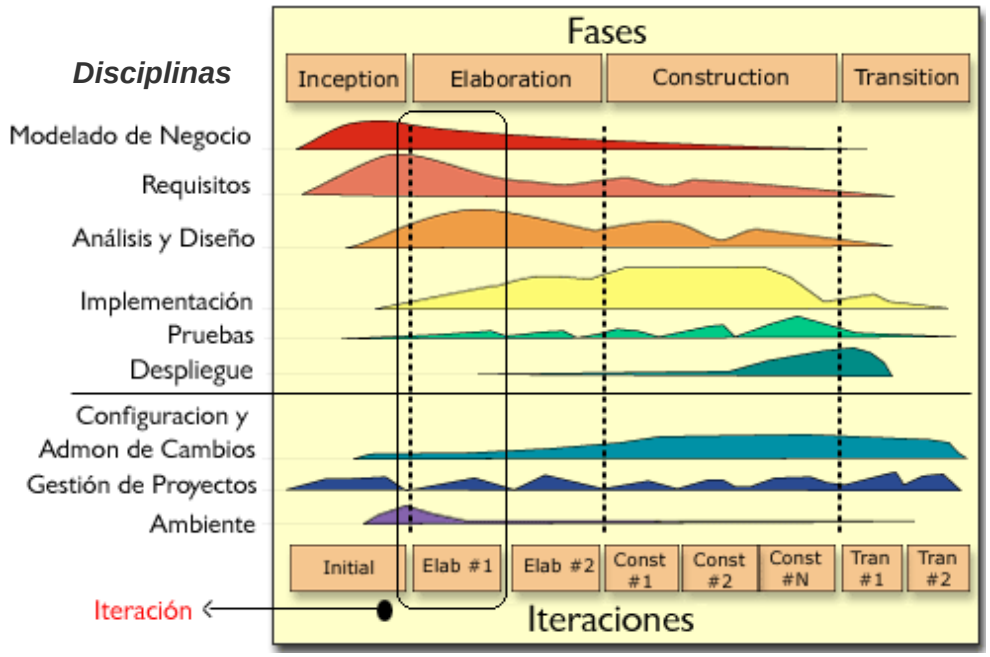


Figura 2.6 Esquema general del Proceso Unificado [Rational, 2002]

1. *Modelado de negocio*. En el caso de que el cliente no presente una descripción detallada su organización y del sistema (o sistemas) que desea construir, que pueda servir de punto de partida del desarrollador del sistema, es necesario proceder a desarrollar un modelo del negocio para comprender los procesos de negocio de la organización. El modelado del negocio está soportado por dos tipos de modelos: modelo de casos de uso del negocio y modelo de objetos.

   a. *Modelo de casos de uso del negocio*. Describe los procesos de negocio y los clientes en términos de casos de uso y actores del negocio. Este modelo de casos de uso permite a los desarrolladores comprender mejor el valor que proporciona el negocio a sus actores.

   b. *Modelo de objetos del negocio.* Es un modelo interno del negocio. Describe cómo se lleva a cabo cada caso de uso del negocio por el conjunto de trabajadores del negocio. Cada realización de un caso de uso se puede mostrar en diagramas de interacciones y diagramas de actividades.

2. *Captura de requisitos*. Tiene por objeto averiguar qué se debe construir. Ello implica que el cliente y los desarrolladores tienen que tener claro lo que debe y no debe hacer el sistema. El Proceso Unificado plantea los siguientes pasos para la captura de requisitos:

   a. *Enumerar los requisitos candidatos*, mediante una "lista de características" que incorpore por cada requisito: su estado, coste estimado de implementación, prioridad y nivel de riesgo en su implementación.

   b. *Comprender el contexto del sistema*, modelando el dominio del problema. Los objetos del modelo del dominio son una parte de los del negocio. Se puede pensar en el modelo del dominio como una variante simplificada del modelo del negocio, en la que uno se centra sólo en las cosas que se van a modelar, es decir, las entidades del negocio que necesitan usar los trabajadores.

c. *Capturar los requisitos funcionales*, mediante casos de uso. El caso de uso es una funcionalidad que proporciona el sistema. Para el usuario el caso de uso representa una forma de usar el sistema. También se especificará la forma de la interfaz de usuario para la ejecución de los casos de uso.

d. *Capturar los requisitos no funcionales*. Se especifican propiedades del sistema como restricciones, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, fiabilidad, etc.

3. *Análisis*. Mientras que los requisitos proporcionan una vista externa del sistema descrita en lenguaje del usuario, el análisis proporciona una vista interna descrita en el lenguaje de los desarrolladores. En el análisis, se analizan los requisitos capturados anteriormente con el objeto de tener una comprensión y descripción más precisa de los mismos, se refinan y se estructuran. Se plantean las siguientes actividades de análisis:

   a. *Analizar la arquitectura*. Consiste en esbozar el modelo de análisis y la arquitectura mediante la identificación de: paquetes de análisis, clases de análisis (de interfaz, de control y de entidad) y requisitos especiales sobre los casos de uso).

   b. *Analizar los casos de uso*. Consiste en analizar cada caso de uso: identificando sus clases de análisis, distribuyendo el comportamiento del caso de uso entre las clases, y capturando los requisitos especiales (no funcionales) sobre la realización del caso de uso.

   c. *Analizar las clases.* Consiste en analizar cada clase del caso de uso: identificando y definiendo las responsabilidades, atributos y relaciones de la clase; y capturando los requisitos especiales (no funcionales) sobre la realización de la clase.

4. *Diseño*. El diseño produce un modelo físico del sistema que se va a implementar (modelo de diseño) que es específico para una implementación, más formal que el del análisis y que debe ser mantenido durante todo el ciclo de vida del software. Se plantean las siguientes actividades de diseño.

   a. *Diseñar la arquitectura*. Consiste en esbozar los modelos de diseño y de despliegue.

   b. *Diseñar los casos de uso*. Consiste en identificar las clases de diseño y los subsistemas, distribuir el comportamiento del caso de uso entre las clases y subsistemas participantes, definir los requisitos sobre las operaciones de las clases de diseño y los subsistemas, y capturar los requisitos especiales (no funcionales) del caso de uso.

   c. *Diseñar las clases*. Consiste en definir para cada clase de diseño: sus operaciones, atributos, relaciones en las que participa, sus métodos, los estados de los objetos de la clase y los requisitos especiales relevantes para su implementación.

d. *Diseñar los subsistemas*: Consiste en definir las dependencias que se producen entre subsistemas cuando una clase pertenece a varios, las interfaces proporcionadas por los subsistemas y los contenidos de los subsistemas.

5. *Implementación*. Consiste en implementar el sistema en términos de componentes, es decir, ficheros de código fuente, de cabecera, ejecutables, etc.
6. *Pruebas*. Los objetivos de las pruebas son: planificar las pruebas de cada iteración, incluyendo las pruebas de integración y del sistema; crear casos de prueba especificando qué probar, cómo realizar la prueba; realizar las pruebas y evaluar sus resultados.
7. *Despliegue*. El objetivo del despliegue consiste en asegurarse que el producto está preparado para suministrarlo al cliente, ajustar el producto software a las necesidades del usuario, y proceder a su entrega y recepción por parte del usuario.

Dentro de todas las disciplinas fundamentales, hay que identificar cuáles son las relevantes para esta Tesis, es decir, cuáles son las que se encargan de realizar el análisis del problema. Hay tres disciplinas candidatas: el modelado del negocio, la captura de requisitos y el análisis.

El modelado del negocio se puede descartar, ya que su objetivo es realizar un modelo de la entidad donde se va a desarrollar uno o varios sistemas. Los propios autores indican que no es necesario realizar el modelo del negocio si ya hay una definición de los objetivos del sistema que se desea.

Por tanto quedan las disciplinas de captura de requisitos (el modelo de casos de uso) y de análisis. Para determinar sus características se va a acudir a los propios autores del Proceso Unificado, que realizan la comparación recogida en la Tabla 2.3.

Tabla 2.3 Comparación de Captura de Requisitos y Análisis en Proceso Unificado

Modelo de Casos de Uso	Modelo de Análisis
• Se describe con el lenguaje del cliente. • Presenta una vista externa del sistema. • Está estructurado por los casos de uso. • Se utiliza como contrato entre el cliente y los desarrolladores sobre lo que debería o no hacer el sistema. • Captura la funcionalidad del sistema. • Define casos de uso.	• Se describe con el lenguaje del desarrollador. • Presenta una vista interna del sistema. • Está estructurado por las clases de análisis. • Señala cómo incorporar esa funcionalidad en el sistema • Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso.

Según esta comparación, puede parecer que el resultado de la captura de requisitos (el modelo de casos de uso) es realmente el análisis del problema, ya que está descrito en el lenguaje del cliente y sirve para definir lo que hay que hacer. Sin embargo, los propios autores de la metodología consideran que en sistemas complejos no es suficiente con el modelo de casos de uso y se hace necesario realizar el modelo de análisis para que los desarrolladores comprendan realmente lo que hay que hacer:

> *"Aunque consigamos llegar a un acuerdo con el cliente acerca de lo que debería aparecer en el sistema, es probable que queden asuntos sin resolver relativos a los requisitos del sistema. Este es el precio que hay que pagar por usar el intuitivo pero impreciso lenguaje del cliente durante la captura de requisitos. ... En el análisis se analizan los requisitos obtenidos en la*

> *captura de requisitos mediante su refinamiento y estructuración. El objetivo de hacer esto es lograr una comprensión más precisa de los requisitos y obtener una descripción de los requisitos que sea fácil de mantener y que ayude a dar estructura al sistema completo" [Jacobson et al., 1999].*

Por lo tanto se hace necesario tratar la combinación de ambas disciplinas como el análisis del problema del Proceso Unificado. A continuación se describen brevemente las técnicas que se utilizan para concretar los resultados de estas dos disciplinas, todas ellas definidas dentro de UML. En la terminología de UML estas técnicas reciben el nombre de artefactos.

- *Captura de Requisitos*. Se utilizan los siguientes artefactos:
  - *Actor*. Representa es un conjunto coherente de roles que desempeñan los usuarios de los casos de uso cuando interactúan con estos. Un Actor representa a un tipo de usuario cuando éste interacciona con el sistema en un caso de uso o bien a un sistemas externo que en algún momento interacciona con el sistema bajo estudio, como por ejemplo una base de datos o el sistema operativo. Cada actor tiene un nombre y se especifica mediante una descripción informal en lenguaje natural.
  - *Caso de Uso.* Un Caso de Uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede realizar y que ofrece un resultado observable o tangible para un determinado usuario. Cuando termina el proceso de captura de requisitos cada caso de uso tiene asignado un nombre, una descripción textual informal, con los campos descritos en la Tabla 2.4) y un diagrama de transición de estados (representado según la notación recogida en la Figura 2.7).

Tabla 2.4 Elementos de la descripción textual de un Caso de Uso

Campo	Descripción
Nombre	Nombre del caso de uso.
Precondición	Descripción del estado inicial del que parte, es decir, cuál es el estado de sus actores y qué pretenden.
Flujo de eventos	Descripción de los pasos del caso de uso, indicándose en cada paso la interacción con los actores y la información que se intercambian.
Atributos	Elementos utilizados para describir los estados: objetos, valores o recursos del sistema.
Postcondición	Estados finales que se pueden alcanzar, describiendo qué han obtenido los actores.
Caminos alternativos	Alternativas al flujo de eventos principal.

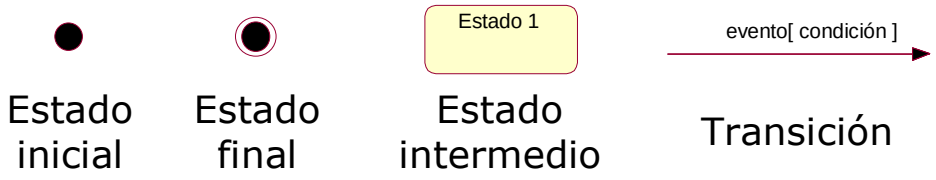


Figura 2.7 Notación de Diagramas de Transición de Estados de UML

- *Modelo de Casos de Uso*. Es un modelo del sistema que compendia los casos de uso, sus actores y sus relaciones. El sistema usual de describir este modelo es mediante un diagrama de casos de uso, cuya notación aparece en la Figura 2.8.

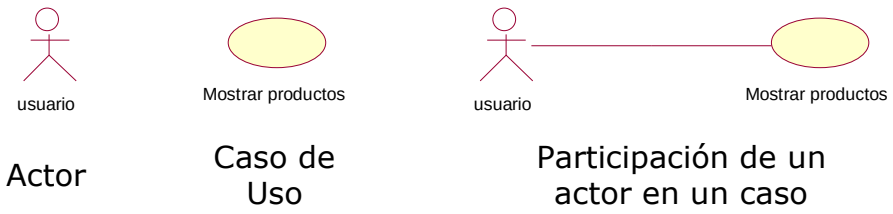


Figura 2.8 Notación de Diagramas de Casos de Uso en UML

- *Análisis*. Se utilizan los siguientes artefactos:
  - *Clase de análisis*. Existen tres tipos básicos de clases de análisis:
    - *Interfaz o Frontera* (del inglés *boundary*). Son usadas como intermediarios entre el sistema y sus actores.
    - *Entidad*. Son usadas para modelar información que persiste en el tiempo o tiene una larga vida.
    - *Control*. Son clases que realizan la coordinación, secuenciado de transacciones y, en definitiva, el control sobre otros objetos del sistema. Se suelen usar para encapsular el control asociado a los casos de uso. Por tanto, toda la dinámica del sistema es modelada por clases de control.

    Al terminar el proceso de análisis, cada una de las clases de análisis tendrá una descripción en la que se recogen sus responsabilidades (acciones que se le solicitan) y sus atributos más relevantes.
  - *Diagrama de clases de análisis*. Se realiza un diagrama para cada caso de uso, donde se especifica qué clases de análisis toman parte del caso de uso y sus relaciones. La notación con la que se representan las clases de análisis y sus relaciones se muestra en la Figura 2.9.

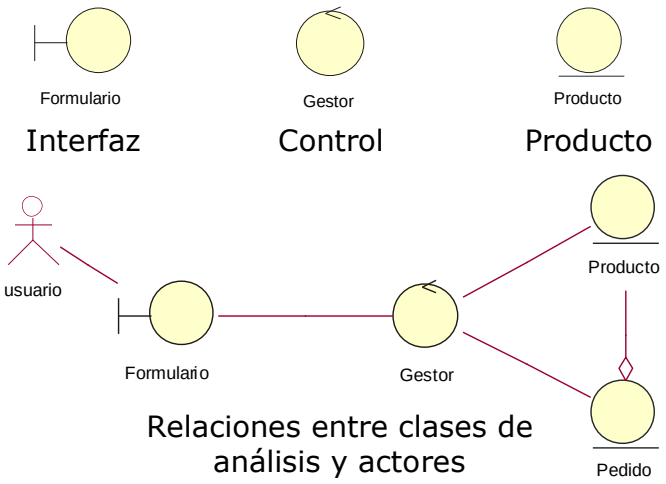


Figura 2.9 Notación de Diagramas de Clases de Análisis en UML

- *Diagrama de colaboración*. Para cada caso de uso se desarrollan uno o más diagrama de interacción entre instancias de clases de análisis (objetos) que representa los objetos, sus enlaces y los mensajes que se envían de una manera secuencial o paralela (Figura 2.10).

  Los autores del Proceso Unificado recomiendan que cada diagrama de colaboración venga acompañado de una descripción de su flujo de eventos, que describa de forma textual lo que sucede.

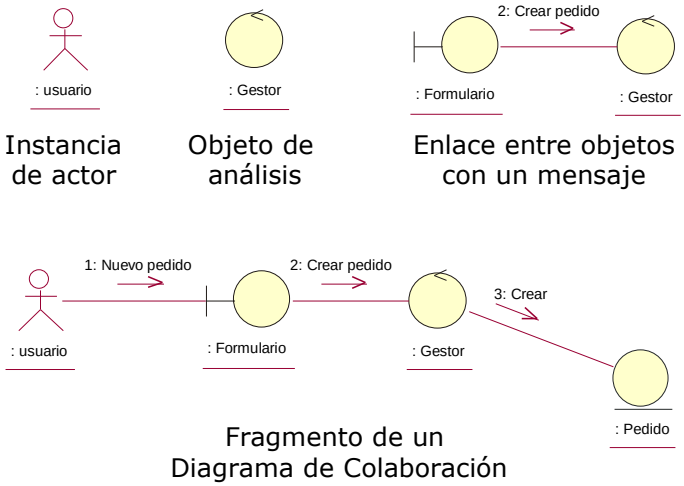


Figura 2.10 Notación de Diagrama de Colaboración de Análisis en UML

Para terminar la descripción del Proceso Unificado conviene resaltar que la transición de la captura de requisitos (con su modelo de casos de uso) al análisis (en el que aparecen las clases de análisis) es especialmente complicada ya que supone la transición del problema hacia el principio de la solución.

La principal dificultad radica en identificar las clases de análisis, para lo cual los autores realizan las siguientes sugerencias:

- *Identificación de clases de tipo entidad*: se refieren a información que se manipula en la realización del casos de uso. Suelen aparecer como atributos de los casos de uso.
- *Identificación de clases de tipo interfaz*. Para cada caso de uso se identifican las siguientes:
  - Una para por cada actor humano, siendo esa clase la que representa la ventana o lo que sea su interfaz de usuario.
  - Una por cada clase entidad encontrada que represente información con la que el usuario humano va a interactuar.
  - Una por cada actor que sea un sistema externo, de forma que esa clase interfaz sea el canal de comunicación con él.
- *Identificación de clases de tipo control*: una para cada caso de uso.
- *Refinado*: se intentan unificar clases de control y de interfaz que aparezcan en casos de uso diferentes, con el fin de reducir el número total de clases.

2.3.2.2 OPEN

OPEN (del inglés *Object Oriented Process, Environment and Notation*, es decir, Notación, Entorno y Proceso Orientados a Objetos) [Firesmith et al., 2001] [Graham et al., 1997a] [Henderson-Sellers et al., 1999] es una metodología de desarrollo orientada a objetos, que pretende unificar los esfuerzos de un grupo numeroso de autores que previamente habían desarrollado sus propias metodologías.

El desarrollo de OPEN coincidió en el tiempo con el esfuerzo del consorcio que diseñó UML y el Proceso Unificado. De hecho inicialmente el consorcio de OPEN estaba frontalmente opuesto a los trabajos de UML y de hecho diseñaron su propia notación, llamada OML [Henderson-Sellers, 1996b][Firesmith et al., 1997]. Este rechazo se ha suavizado en gran medida y las últimas publicaciones de OPEN han aparecido utilizando UML como notación de modelado [Henderson-Sellers et al., 2000].

La estructura del proceso de OPEN está organizada en pasos de alto nivel, llamados actividades. Cada actividad se descompone en una serie de tareas y éstas se realizan aplicando una serie de técnicas.

En el modelo de proceso de OPEN se distingue entre actividades asociadas a un único proyecto y actividades que transcienden el proyecto y que hacen referencia a un grupo de proyectos, llamado programa. También se distinguen entre actividades La Figura 2.11 recoge gráficamente todas estas actividades. En esa figura las actividades pueden ser sin límite de tiempo (las que tienen esquinas redondeadas) o estrictamente ligadas al tiempo (las que tienen esquinas sin redondear).

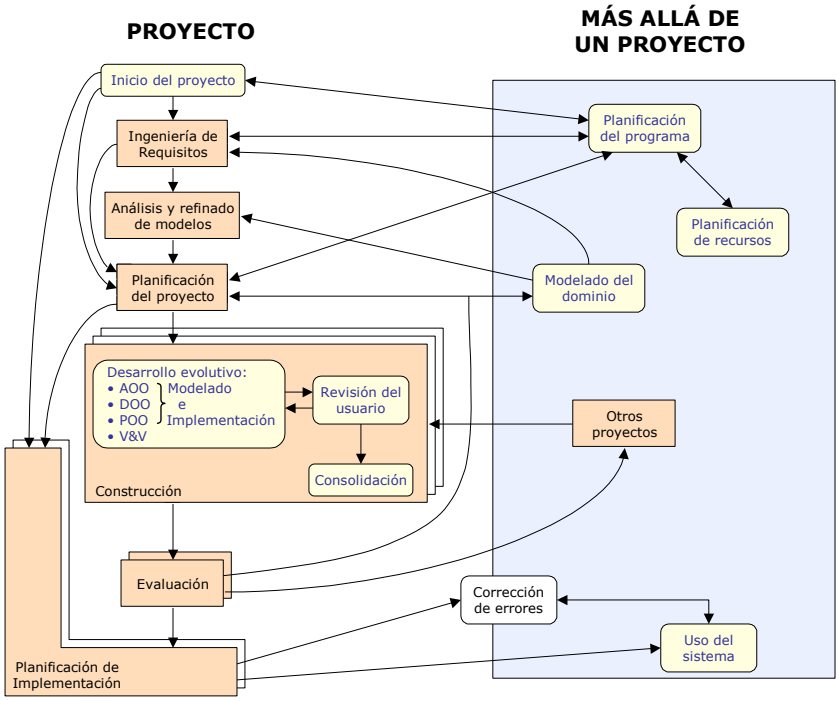


Figura 2.11 El ciclo de vida de OPEN [Graham et al., 1997a]

Seguidamente se resumen las actividades relacionadas con el desarrollo de un proyecto:

- *Inicio del proyecto*. A partir de una necesidad en el negocio se realiza un documento de propuesta de proyecto, que debe especificar su misión global y las limitaciones

de coste y de otro tipo. También se debe incluir un análisis de coste – beneficios y de viabilidad.

- *Ingeniería de requisitos*. Esta actividad está descrita en detalle en [Swatman et al., 1998]. Los autores proponen aplicar el análisis de tareas durante el proceso de generación de requisitos. El resultado es la construcción un modelo de tareas (TOM, del inglés *Task Object Model*), en el cual se describen los procesos del negocio en los que participará el sistema que se va a desarrollar. La terminología usada en el TOM debe ser la del problema.
- *Análisis y refinado de modelos*. En esta actividad se parte del TOM, y se identifican objetos del negocio y sus responsabilidades, con lo que se genera un modelo de objetos del negocio (BOM, del inglés *Business Object Model*). Este modelo está a medio camino entre la definición del problema y su resolución. Es un modelo orientado a objetos, pero mantiene su foco de atención en el mundo real. Por ello puede servir como un contrato entre los usuarios y los desarrolladores. El resultado final es un informe de análisis que contiene el TOM refinado y el BOM.
- *Planificación del proyecto*. En esta actividad, que puede empezar una vez se aprueba la propuesta del proyecto, se realiza una planificación de las tareas de desarrollo y una preparación de todos los recursos necesarios para comenzar a desarrollar.
- *Construir*. Esta actividad integra de forma iterativa, tres sub-actividades que se corresponden con los aspectos técnicos del desarrollo de software:
    - *Desarrollo evolutivo.* En esta actividad se va construyendo el modelo de diseño (SOM, del inglés *System Object Model*) y se procede a su implementación utilizando un lenguaje de programación orientado a objetos. Los autores distinguen los pasos de análisis, diseño, programación orientadas a objetos, pero justifican que no hay una transición real entre ellas (sobre todo en lo que respecta a las dos primeras), ya que es muy difícil distinguir cuándo se están descubriendo objetos a partir del problema (análisis) y cuándo se están inventado nuevos objetos para la resolución del problema (diseño). Durante el desarrollo evolutivo se incluyen tareas de validación y verificación de los prototipos desarrollados.
    - *Revisión del usuario.* En esta actividad, cada prototipo generado durante el desarrollo evolutivo es evaluado por los usuarios para comprobar su aceptación.
    - *Consolidación*. En esta actividad se consolidan los resultados de varios ciclos de desarrollo evolutivo. Por un lado se identifican sistemas que pueden implantarse en la organización y, por otro, se identifican los componentes de esos sistemas que puedan ser candidatos para reutilizarlos en proyectos futuros.
- *Evaluación*. Esta actividad consiste en una revisión formal del sistema y su documentación, con el fin de decidir si se procede a su implantación en la organización.
- *Planificación de implantación*. En esta actividad se prepara todo lo necesario para la implantación del sistema, incluyendo entrenamiento, formación, recursos hardware

y software, soporte técnico, etc. El objetivo es asegurar que todo está preparado para la puesta en marcha del sistema.

Tras esta breve descripción de las tareas de OPEN, es complicado decidir cuáles se corresponden con la fase de análisis del problema tal y como se ha definido en esta Tesis. Para facilitar esta elección se puede utilizar la explicación de los propios autores sobre la evolución de los diferentes modelos a lo largo del tiempo (Figura 2.12): TOM, BOM, SOM y el modelo de implementación (IOM, del inglés *Implementation Object Model*).

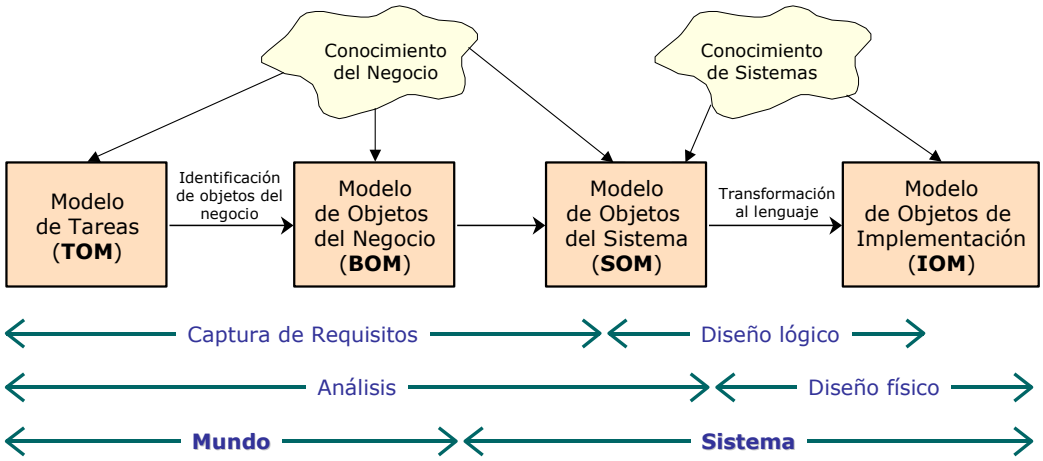


Figura 2.12 Evolución de TOM, BOM, SOM e IOM en OPEN [Graham et al., 1997a]

Es especialmente interesante en la figura la última línea, en la que los autores indican qué modelos hacen referencia al mundo (al problema) y cuáles al sistema (la solución). Así, los modelos de tareas (TOM) y de objetos del negocio (BOM) forman parte del estudio del problema, mientras que los otros dos modelos (SOM e IOM) forman parte de la solución.

Por lo tanto, a efectos de esta Tesis interesa describir los elementos de los dos primeros modelos:

- *Modelo de Tareas (TOM)*. En este modelo se describen las tareas del negocio de forma jerárquica.
  - En el nivel más alto está la *misión* de la empresa (o de un área de negocio de la misma) dentro de la cual se va a realizar un nuevo sistema.
  - Esta misión se divide en una serie de *objetivos* que puedan evaluarse y priorizarse.
  - Para el cumplimiento de cada uno de los objetivos, la empresa deberá mantener alguna conversación con personas u organizaciones externas o internas, que en OPEN se denominan *mensajes*. Todo mensaje deberá servir para un objetivo y todo objetivo se realizará mediante uno o más mensajes. El conjunto de todos los mensajes relevantes para un sistema se refleja mediante diagramas de contexto, con la notación mostrada en la Figura 2.13. Hay dos tipos de diagramas de contexto:

Figura 2.13 Notación de diagramas de contexto en OPEN

- Externos: representan la comunicación de la empresa con el exterior. Estos diagramas sólo muestran objetos externos (personas o empresas), mensajes y la empresa. La Figura 2.14 muestra un ejemplo.

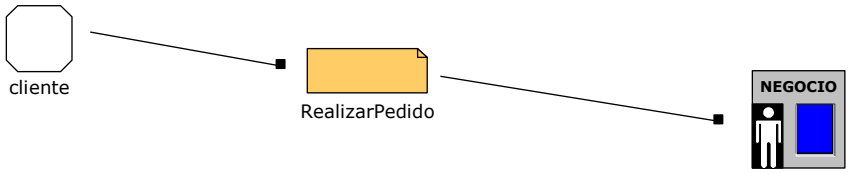


Figura 2.14 Ejemplo de diagrama de contexto externo en OPEN

- Internos: desaparece la empresa y en cambio se muestran el sistema y los actores (personas que manejan el sistema). La Figura 2.15 muestra un ejemplo.

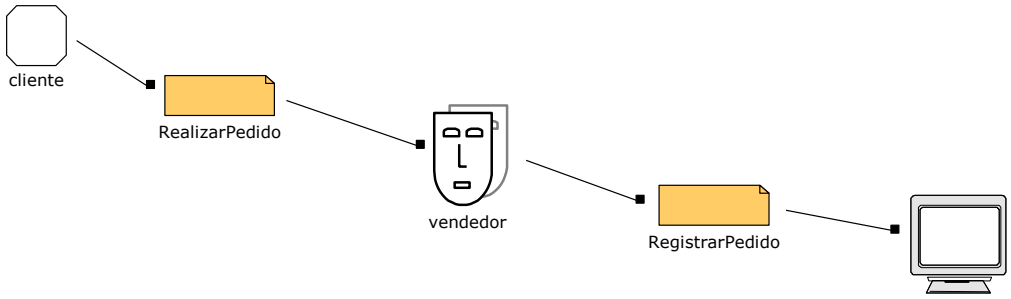


Figura 2.15 Ejemplo de diagrama de contexto interno en OPEN

Es interesante destacar que en los diagramas de contexto se reflejan todas las conversaciones (mensajes), impliquen o no al sistema informático bajo estudio.

- Dentro de un mensaje se identifican la *meta* que se logra cuando se completa y la *tarea* que se realiza durante la conversación correspondiente. Cada una de las tareas puede descomponerse en *subtareas* de varios niveles y se describen mediante *guiones de tarea* (del inglés, *task scripts*) que es una técnica de representación de tareas basada en la técnica de guiones de Inteligencia Artificial y que, según los autores, es superior a los casos de uso [Graham, 1997]. La descomposición de tareas sigue hasta llegar a tareas que no se descomponen más, que se llaman *tareas atómicas* y que se representan mediante frases, preferiblemente en la forma sujeto + verbo + objeto. La especificación de cada tarea se realiza mediante una tarjeta de tarea según el formato mostrado en la Tabla 2.5.

Tabla 2.5 Especificación de tareas en OPEN

Nombre de Tarea Abstracta / Concreta	
Cuerpo de Tarea (guión)	
Supertareas	
Tareas hijas (componentes)	
Tareas asociadas o análogas	
Atributos de la Tarea	
Tiempo que lleva realizarla	
Complejidad	
Excepciones	Guiones alternativos
Reglas	

- *Modelo de Objetos del Negocio (BOM)*. El modelo de objetos del negocio se obtiene mediante un análisis textual de los guiones de las tareas. Todas las clases identificadas se especifican mediante una tarjeta de clase según el formato mostrado por la Tabla 2.6.

Tabla 2.6 Especificación de clases de negocio en OPEN

Nombre de Clase Abstracta / Concreta	
Descripción	
Clases padre	
Clases hija (componentes)	
Atributos y asociaciones	
Operaciones	Usada en
Reglas	

Para concluir la descripción de la metodología OPEN hay que indicar que sus autores también reconocen la dificultad de la transición entre el TOM y el BOM. De hecho ellos lo llaman "pasar el Rubicón" (por alusión al momento en que Julio César, sin autorización del senado, cruza este río con sus legiones y penetra en Italia) [Graham, 1996a], [Graham et al., 1997a], tal y como muestra la Figura 2.16.

En esta figura se aprecia una barrera (el "Rubicón") entre el modelo de las tareas (que está definido en términos del problema y sin ninguna referencia a aspectos de la solución) y el modelo de objetos (que está estructurado en clases, como punto de partida para la arquitectura de la solución). Antes de pasar el Rubicón las transiciones misión – objetivo – mensaje – tarea – tarea atómica son naturales y lo mismo ocurre después de pasarlo con las transiciones objeto de negocio – objeto de sistema – código. La dificultad está en el paso de la barrera.

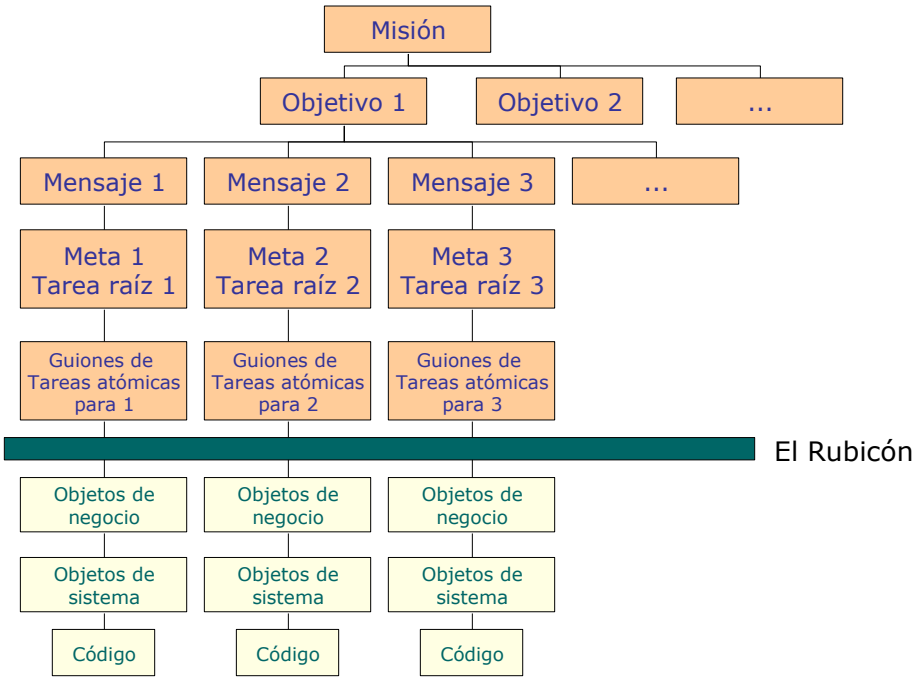


Figura 2.16 El paso del Rubicón: de las tareas a los objetos [Graham, 1996a]

2.3.2.3 Otras Metodologías Orientadas a Objetos

Tal y como se comentó al empezar esta sección dedicada a las metodologías orientadas a objetos, existe una gran cantidad de ellas, entre las que se pueden destacar las siguientes:

- En primer lugar, las que dieron origen a UML y el Proceso Unificado: la metodología de Booch [Booch, 1994], OMT (del ingles *Object Modeling Technique*) [Rumbaugh et al., 1991] y OOSE (*Object Oriented Software Engineering*) [Jacobson et al., 1992].
- En segundo lugar, las que dieron origen a OPEM: MOSES [Henderson-Sellers et al., 1994], SOMA [Graham, 1991] [Graham, 1995] y Firesmith [Firesmith, 1993].
- Por último, a pesar de la poca atención que están recibiendo otras metodologías actuales distintas del Proceso Unificado y OPEN, se pueden destacar algunas por su originalidad en el proceso y en alguno de sus modelos. Por ejemplo, Catalysis [D'Souza et al., 1998] [D'Souza et al., 1995] y el método basado en especificación formal FOOM (del inglés, *Formal Object Oriented Method*) [Fowler et al., 1998] [Nguyen et al., 2000a] [Swatman et al., 1998].

2.3.3 Metodologías Orientadas a Agentes

Dentro de los paradigmas de programación y diseño, existe un paradigma creado recientemente y que está alcanzando una rápida difusión: el paradigma orientado a agentes.

El término agente proviene de los trabajos de Inteligencia Artificial Distribuida a finales de los años ochenta, que culminaron en los Sistemas Multi-Agente (MAS) [Guilfoyle et al., 1994].

El concepto clave que se maneja en esta aproximación es el de agente, cuya definición no ha sido aún consensuada entre los investigadores del área. Pueden encontrarse algunas definiciones de agente en [FIPA, 1996] [Maes, 1995] y [Nwana et al, 1996].

Habitualmente se usa el término agente para referirse a un sistema software que presenta varias de las siguientes características: inteligencia, autonomía, percepción, actuación independiente. Muchos autores ven los agentes como una extensión del concepto de objeto al que se añaden características como metas, motivaciones, creencias, negociación con otros agentes, etc.

Dado el avance sufrido por la tecnología de agentes en los últimos años, han surgido metodologías orientadas a agentes que guían el ciclo de vida del desarrollo de un MAS. Estas metodologías siguen alguno de los siguientes enfoques:

- *Metodologías basadas en metodologías orientadas a objeto*: partiendo de metodologías orientadas a objeto, se añaden características propias de los agentes: creencias, objetivos, planes, interacciones entre agentes, etc.
- *Metodologías basadas en metodologías de ingeniería del conocimiento*: partiendo de metodologías de ingeniería del conocimiento, como KADS o su sucesora CommonKADS, se añaden el modelado de interacciones y la conducta proactiva de los agentes.
- *Metodologías orientadas a sociedades de agentes*. Estas metodologías se basan en la tecnología de agente y multi-agente, centrándose en abstracciones de nivel social, como son el agente, el grupo o la organización.

En los apartados siguientes se van a describir, de forma muy breve, las características más relevantes de algunas metodologías de cada tipo. Para una descripción más detallada puede consultarse [Frutos, 2003].

2.3.3.1 Metodologías basadas en la orientación a objetos

Debido a la similitud entre el paradigma de objetos y de agentes y a la popularidad de las metodologías orientadas a objetos en el desarrollo de software, surgen las primeras metodologías orientadas a agentes basadas en estas metodologías.

Dentro del grupo de las metodologías basadas en metodologías orientadas a objeto caben destacar: la Metodología de Agentes Styx [Bush et al., 2001], Metodología MaSE [DeLoach et al., 2001a], Método MASSIVE [Lind, 2001], Metodología AAII [Kinny et al., 1997], Metodología Orientada a Agentes para Modelado de Empresas [Kendall et al., 1997a], Técnica de Análisis y Diseño Orientado a Agentes [Burmeister, 1996] y Método MASB [Moulin et al., 1996].

2.3.3.1.1 Metodología Styx

La metodología de agentes Styx [Bush et al., 2001] está orientada al desarrollo de agentes colaborativos. Propone los siguientes modelos: modelo de conceptos del dominio, modelo responsabilidad de roles, modelo de relaciones de roles y modelo de utilización (Figura 2.17). Esta metodología consta de las siguientes fases:

1. Fase de análisis:
    a. Identificación de los roles de los agentes y de los conceptos del dominio.

b. Generación del Mapa de Casos de Uso de alto nivel.

c. Generación del Modelo de Conceptos de Dominio.

2. Fase de diseño:

   a. Generación del Modelo de Responsabilidad de Roles para cada componente del Mapa de Casos de Uso.

   b. Generación del Modelo de Relaciones de Roles que especifica como se relacionan los roles entre ellos y que conceptos utilizan para comunicarse.

   c. Generación del Modelo de Utilización que asocia los roles identificados con agentes.

3. Fase de implementación:

   - Soportada por un esqueleto de agente y código específico de la aplicación

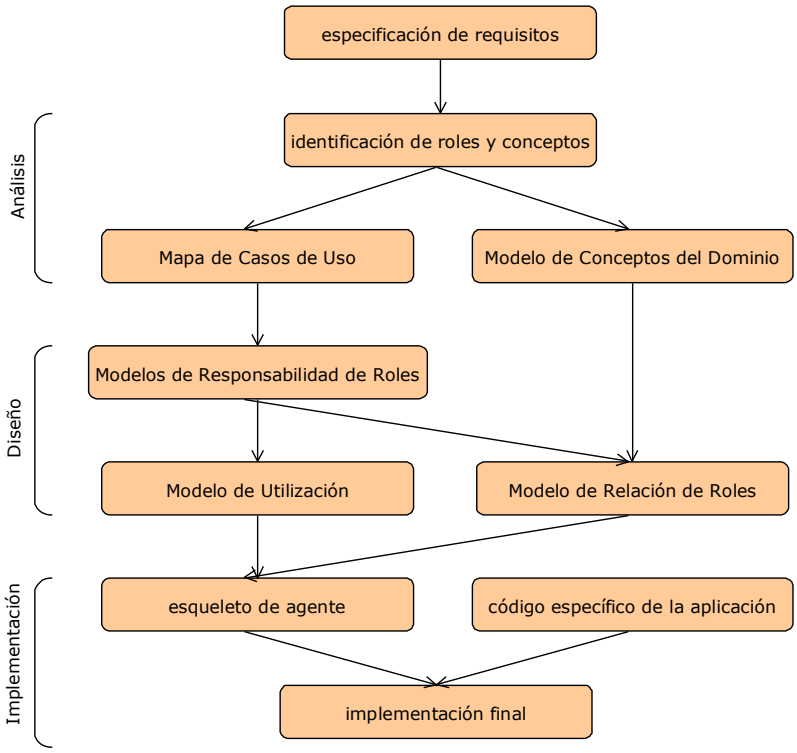


Figura 2.17 Perspectiva general de la metodología de agentes Styx

Puede observarse cómo la fase de análisis comienza con la presuposición de que va a realizarse un sistema multi-agente y a partir de esa hipótesis utiliza elementos de las metodologías orientadas a objetos como los casos de uso, pero sesgadas hacia la construcción de agentes.

2.3.3.1.2 Metodología MaSe

La metodología MaSE (del inglés *Multiagent Systems Engineering*) [DeLoach, 1999] [DeLoach et al., 2001a] [DeLoach et al., 2001b] [Wood, 2000] [Wood et al., 2000] divide el diseño en varios niveles de generalidad: nivel de dominio, nivel de agente y nivel de componente y diseño del sistema.

La metodología aporta dos lenguajes para describir agentes y sistemas multi-agente: AgML (del inglés, *Agent Modeling Language*), que es el lenguaje del nivel de sistema; y AgDL (del inglés *Agent Definition Language*), que describe el funcionamiento interno del agente.

MaSE realiza un proceso en cascada de cuatro etapas (Figura 2.18):

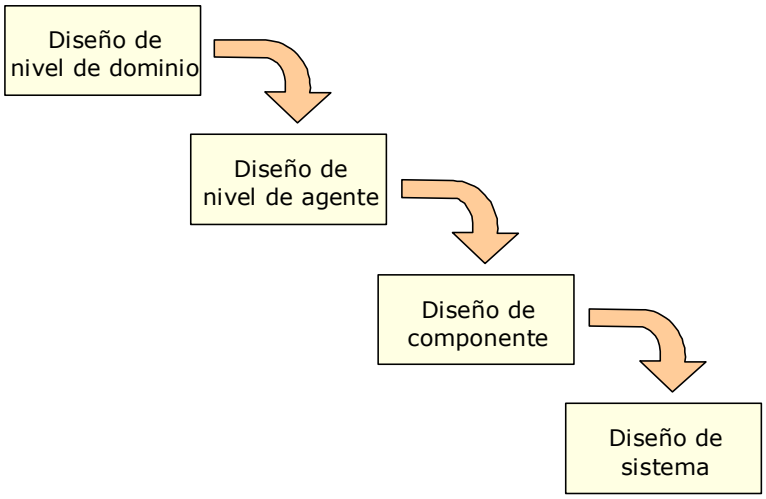


Figura 2.18 Etapas de la metodología MaSe

- **Diseño de nivel de dominio**: utiliza AgML para capturar los tipos básicos de agentes en el sistema y las interacciones entre ellos. AgML utiliza diagramas para definir las características externas de un MAS: el diagrama de agente, el diagrama de jerarquías de comunicación y el diagrama de clases de comunicación. Existen tres etapas en esta fase de diseño:
  - Identificación de tipos de agentes.
  - Identificación de posibles interacciones entre tipos de agentes.
  - Definición de protocolos de coordinación para cada tipo de interacción
- **Diseño de nivel de agente**: se describe cada tipo de agente utilizando AgDL. Existen tres etapas en esta fase de diseño:
  - Asociación de acciones identificadas en las conversaciones de agentes con componentes internos
  - Definición de estructuras de datos identificadas en conversaciones de agentes
  - Definición de estructuras de datos adicionales, internas al agente.
- **Diseño de nivel de componente**: se detallan los componentes introducidos en las etapas anteriores.
- **Diseño del sistema**: se unen todos los agentes diseñados y otros componentes.
  - Seleccionar los tipos de agentes necesarios
  - Determinar el número de agentes requeridos de cada tipo y definir la localización física o dirección del agente, los tipos de conversaciones que los

agentes son capaces de mantener y cualquier otro parámetro definido en el dominio.

2.3.3.1.3 Método MASSIVE

El método MASSIVE (del inglés, *MultiAgent SystemS Iterative View Engineering*) [Lind, 2001] [Lind, 1999] combina técnicas de ingeniería del software tradicionales como el modelado en múltiples vistas (Figura 2.19), la re-ingeniería y el procesado iterativo.

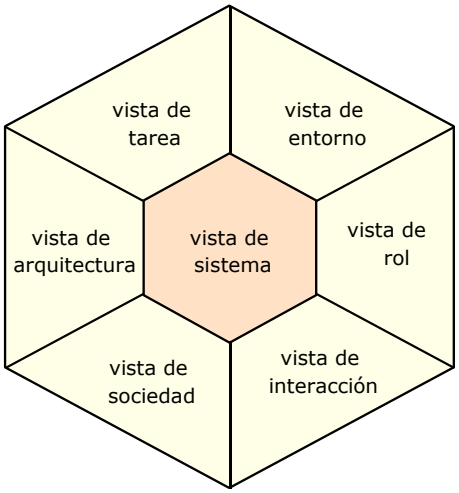


Figura 2.19 Vistas utilizadas por MASSIVE

La parte central del método la constituye el sistema de **vistas**:

- *Vista de entorno*. Se analiza el entorno del sistema a partir de las perspectivas de los desarrolladores y de la perspectiva del sistema.
- *Vista de tarea*. Se analizan los aspectos funcionales del sistema y se genera una jerarquía de tareas.
- *Vista de rol*. Se determina la agregación funcional de las capacidades de resolución de problemas básicos según las restricciones físicas del sistema.
- *Vista de interacción***.** La interacción dentro del sistema se ve como una resolución de conflictos, no limitada a un tipo concreto, como es la comunicación,
- *Vista de sociedad*. Se clasifica la sociedad de manera que pre-exista dentro del contexto organizativo del sistema o que sea interesante desde el punto de vista del desarrollador
- *Vista de arquitectura*. Se transforma la especificación de características en otras vistas en una arquitectura de sistema.
- *Vista de sistema*. Se estructuran las características del sistema según sus enlaces conceptuales o lógicos.

2.3.3.1.4 Metodología AAII

La metodología AAII [Kinny et al., 1996] [Kinny et al., 1997] se centra en agentes con arquitectura BDI (es decir, agentes con creencias, deseos e intenciones – en inglés

Beliefs, Desires, Intentions). Extiende la metodología OMT [Rumbaugh et al., 1991], considerando dos niveles en el modelado de agentes: vista externa y vista interna.

En la **vista externa** se descompone el sistema en agentes a partir de los roles clave de la aplicación. Los agentes son instanciaciones de clases dentro de una jerarquía de clases de agentes. Las clases de agentes se describen en dos modelos: el *modelo de agente*, que describe los agentes y las relaciones jerárquicas entre clases de agentes; y el *modelo de interacción*, que describe las comunicaciones y las relaciones de control entre clases de agentes.

Las etapas para la obtención de los modelos de la vista externa son las siguientes:

1. Identificación de los roles: permite identificar los agentes y organizarlos en una jerarquía de clases de agentes.
2. Para cada rol, identificar las responsabilidades y servicios.
3. Para cada servicio, identificar las interacciones (comunicaciones).
4. Refinar la jerarquía, introduciendo nuevas clases de agentes, componiendo clases y creando nuevas instancias de agentes.

Los roles son conjuntos de responsabilidades y las responsabilidades son conjuntos de servicios.

Las etapas y la vista externa son independientes de la arquitectura de agente elegida. La **vista interna** está fuertemente condicionada por el paradigma BDI. En la vista interna, se modela cada agente BDI utilizando el *modelo de creencias*, que mantiene el conjunto de creencias del agente que es información sobre el entorno y el estado actual del agente y el posible cambio en el tiempo; el *modelo de objetivos*, que contiene los posibles objetivos que puede adoptar un agente y los eventos a los que puede responder; y el *modelo de planes*, que mantiene el conjunto de posibles planes que puede adoptar un agente para conseguir sus objetivos. El conjunto de objetivos contiene los deseos de un agente BDI y el conjunto de planes contiene las intenciones de un agente BDI.

Las etapas para modelar internamente un agente son las siguientes:

1. Análisis de los medios para alcanzar los objetivos.
2. Construir las creencias del sistema.

2.3.3.1.5 Metodología Orientada a Agentes para Modelado de Empresas

La Metodología Orientada a Agentes para Modelado de Empresas [Kendall et al., 1996] [Kendall et al., 1997a] combina la metodología orientada a objetos OOSE [Jacobson et al., 1992], la metodología de modelado de empresas IDEF (del inglés *Integration Definition for Function modelling*) y CIMOSA (del inglés *Computer Integrated Manufacturing Open System Architecture*). Propone los siguientes modelos:

- *Modelo de funciones*: describe las funciones incluyendo la selección de métodos.
- *Modelo de casos de uso*: describe los actores involucrados en cada función
- *Modelo dinámico*: se analizan las interacciones entre objetos.

- *Sistema orientado a agentes*: La construcción de este sistema está compuesto de las siguientes fases:
  - Identificación de los agentes: se identifican como agentes los actores de los casos de uso.
  - Protocolos de coordinación: se describen en diagramas de estado.
  - Invocación de planes: se definen diagramas de secuencia que incluyen las condiciones de invocación de cada plan.
  - Creencias, sensores y actuadores: las entradas de las funciones se modelan como creencias u objetos obtenidos de los sensores y los objetivos como cambios en las creencias o modificaciones a través de los actuadores.

2.3.3.1.6 Técnica de Análisis y Diseño Orientado a Agentes

La Técnica de Análisis y Diseño Orientado a Agentes [Burmeister, 1996] propone tres modelos para analizar un sistema de agentes: el modelo de agente, formado por los agentes y su estructura interna; el modelo de organización, que contiene las relaciones entre los agentes; y el modelo de cooperación, que describe las interacciones entre los agentes.

Para la construcción de estos modelos, se tienen que realizar las siguientes tareas:

- Modelo de Agente:
  - Identificación de los agentes y su entorno.
- Modelo de Organización:
  - Identificación de los roles de cada agente.
  - Elaboración de una jerarquía de herencia y relaciones de agentes.
- Modelo de Cooperación:
  - Identificación de cooperaciones y participantes en las mismas.
  - Analizar los tipos de mensajes intercambiados.
  - Analizar los protocolos empleados.

2.3.3.1.7 Método MASB

El Método Basado en Escenarios Multi-Agente (MASB) [Moulin et al., 1994] [Moulin et al., 1996] propone una metodología para el desarrollo de MAS que soporten trabajo colaborativo.

Las actividades propuestas en la fase de análisis y en la fase de diseño son las siguientes:

- Fase de análisis:
  - Descripción de escenarios: se identifican en lenguaje natural los roles de los agentes software, humanos y objetos del entorno.

  - Descripción funcional de los roles: se describen los roles de los agentes mediante diagramas de conducta, que describen el proceso realizado, la información empleada y la interacción con el resto de agentes.
  - Modelado conceptual de los datos y del mundo: se modela el conocimiento, los datos empleados por cada agente y objetos del mundo, utilizando diagramas entidad-relación y diagramas de ciclo de vida de las entidades.
  - Modelado de la interacción usuario–sistema: se simula y define las interfaces de interacción hombre-máquina para cada escenario.
- Fase de diseño:
  - Descripción de los escenarios y de la arquitectura MAS: se seleccionan los escenarios a implementar y los roles desempeñados por cada agente.
  - Modelado de los objetos: se refinan el modelo de análisis, añadiendo jerarquías de herencia, atributos y métodos.
  - Modelado de los agentes: se especifican las estructuras de creencias y el proceso de decisión de los agentes basándose en las creencias, planes, objetivos e interacciones.
  - Modelado de las conversaciones.
  - Validación global del sistema.

2.3.3.2 Metodologías basadas en Ingeniería del Conocimiento

Las metodologías de Ingeniería del Conocimiento, como CommonKADS [Schreiber et al., 1999] e IDEAL [Gómez et al., 1997], se utilizan en el desarrollo de Sistemas Basados en Conocimientos. Su aplicación exitosa en la industria, aunque no tan extendida como las metodologías orientadas a objetos, y su enfoque a conocimientos y métodos de resolución de problemas, hacen de estas metodologías buenas candidatas para su utilización como base en el desarrollo de MAS.

Dentro del grupo de las metodologías basadas en metodologías de Ingeniería del Conocimiento caben destacar: CoMoMAS [Glaser, 2002] y MAS-CommonKADS [Iglesias et al., 1999].

2.3.3.2.1 CoMoMAS

CoMoMAS (del inglés *Contribution to Knowledge Acquisition and Modelling in a Multi-Agent Framework*) [Glaser, 2002] [Glaser, 1997] [Glaser, 1996] propone una extensión de la metodología CommonKADS [Schreiber et al., 1999] para el modelado de MAS basándose en los modelos de la Figura 2.20.

- *Modelo de Agente*: define la arquitectura del agente y su conocimiento. Los agentes se clasifican en sociales, cooperativos, cognitivos, reactivos y de control.
- *Modelo de la Experiencia*: describe las competencias cognitivas y reactivas del agente. Incluye el conocimiento de las tareas (descomposición de tareas), el conocimiento de resolución de problemas (métodos de resolución de problemas y estrategias de selección) y el conocimiento reactivo (procedimientos para responder a estímulos).

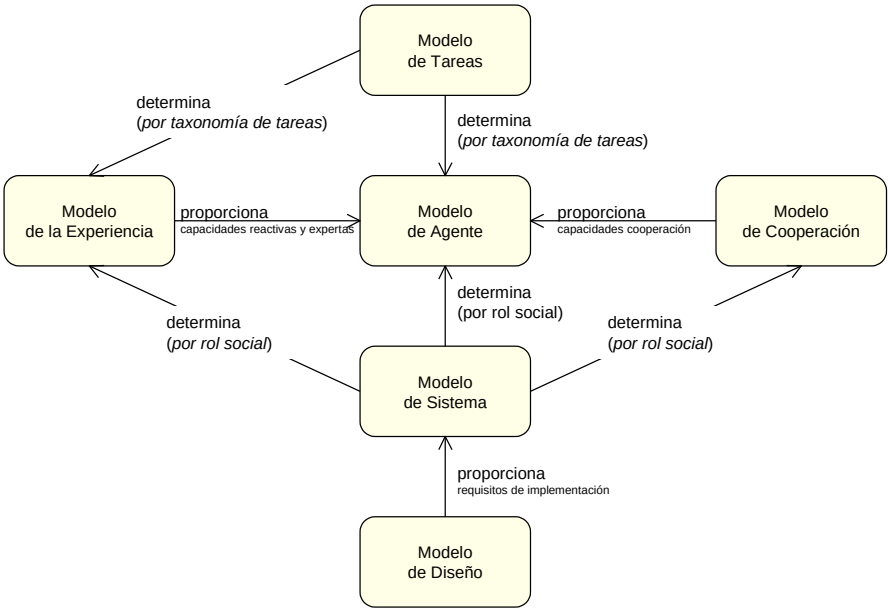


Figura 2.20 Los modelos de CoMoMAS

- *Modelo de Tareas*: describe la descomposición de las tareas indicando si son resolubles por el usuario o por un agente.
- *Modelo de Cooperación*: describe la cooperación entre los agentes. Incluye métodos de resolución de conflictos (negociación y cooperación) y conocimiento de cooperación (primitivas, protocolos y terminología de interacción).
- *Modelo del Sistema*: representa la organización de los agentes y la arquitectura del sistema multi-agente y de sus agentes.
- *Modelo de Diseño*: describe el paso de los modelos anteriores a código ejecutable.

2.3.3.2.2 MAS-CommonKADS

MAS-CommonKADS [Iglesias et al., 1999] [Iglesias, 1998], extensión de la metodología CommonKADS, propone los siguientes modelos para el desarrollo de MAS (Figura 2.21):

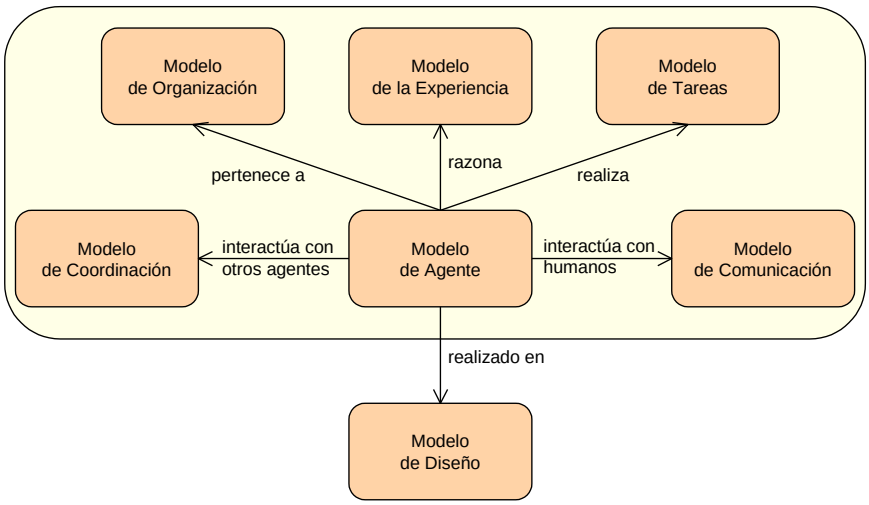


Figura 2.21 Modelos de MAS-CommonKADS

- *Modelo de Agente*: especifica las características de un agente: sus capacidades de razonamiento, habilidades, servicios, sensores, efectores, grupos de agentes y clases de agentes.
- *Modelo de Organización*: analizar la organización humana en la que se va a introducir el MAS y describe la organización de agentes software y su relación con el entorno.
- *Modelo de Tareas*: describe las tareas que los agentes pueden realizar: sus objetivos, su descomposición, los ingredientes y los métodos de resolución de problemas para cada objetivo.
- *Modelo de la Experiencia*: describe el conocimiento necesitado por los agentes para alcanzar sus objetivos.
- *Modelo de Comunicación*: describe las interacciones entre un agente humano y un agente software.
- *Modelo de Coordinación*: describe las interacciones entre agentes software.
- *Modelo de Diseño*: mientras que los modelos anteriores tratan del análisis, este modelo describe la arquitectura y el diseño del sistema multi-agente.

Las fases que propone esta metodología son las siguientes:

1. *Conceptualización*: obtener una primera descripción del problema y la determinación de los casos.
2. *Análisis*: determinar los requisitos del sistema partiendo del enunciado del problema. Durante esta fase se desarrollan los siguientes modelos: organización, tareas, agente, comunicación, coordinación y experiencia.
3. *Diseño*: Determinar cómo los requisitos de la fase de análisis pueden ser logrados mediante el desarrollo del modelo de diseño.
4. *Codificación y pruebas de cada agente*.
5. *Integración*. Pruebas del sistema completo.
6. *Operación y mantenimiento*.

2.3.3.3 Metodologías orientadas a sociedades de agentes

Estas metodologías se basan en la tecnología de agente y multi-agente, centrándose en abstracciones de nivel social, como son el agente, el grupo o la organización.

Dentro de este tipo de metodologías destacan SODA [Omicini, 2001], Gaia [Wooldridge et al., 2000] y Cassiopea [Collinot et al., 1996].

2.3.3.3.1 Metodología SODA

SODA (del inglés, *Societies in Open and Distributed Agent spaces*) [Omicini, 2001] es una metodología orientada a agentes para el análisis y diseño de sistemas basados en Internet. A continuación se muestran los modelos que generan durante la fase de análisis y la de síntesis.

- Fase de Análisis.
  - *Modelo de Rol*: los objetivos de la aplicación se modelan en términos de las *tareas* a realizar, que están asociadas con los *roles* y los *grupos*.
  - *Modelo de Recurso*: el entorno de la aplicación se modela en términos de *servicios* disponibles, que están asociados con *recursos* abstractos.
  - *Modelo de Interacción*: la interacción entre roles, grupos y recursos se modela en términos de *protocolos de interacción* expresados como la *información* requerida y proporcionada por roles y recursos, y de *reglas de interacción* que gobiernan la interacción dentro de los grupos.
- Fase de Diseño.
  - *Modelo de Agente*: los roles sociales e individuales se transforman en clases de *agentes*.
  - *Modelo de Sociedad*: los grupos se transforman en *sociedades* de agentes, diseñadas y organizadas por *abstracciones de coordinación*.
  - *Modelo de Entorno*: los recursos se transforman en clases de *infraestructura* y se asocian a *abstracciones topológicas*.

2.3.3.3.2 Metodología Gaia

Metodología Gaia [Wooldridge et al., 1999] [Wooldridge et al., 2000] permite diseñar un MAS partiendo de conceptos abstractos (roles, permisos, responsabilidades, protocolos, actividades, propiedades de vida, propiedades de seguridad) para llegar a conceptos más concretos (tipos de agente, servicios, conocimientos) (véase la Figura 2.22).

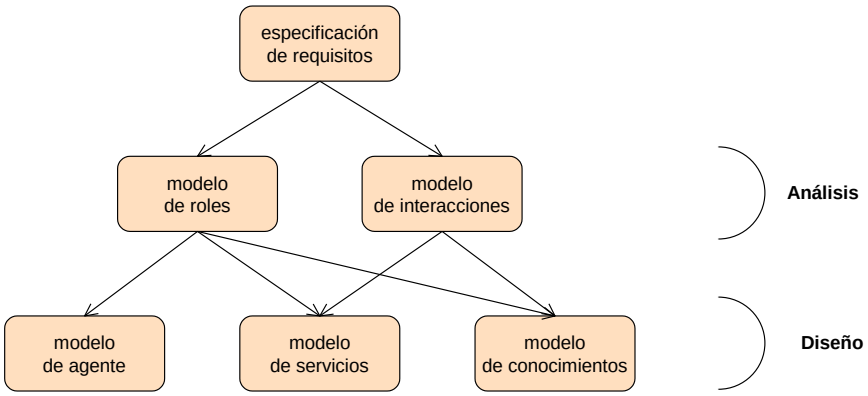


Figura 2.22 Relaciones entre los modelos de la metodología Gaia

Esta metodología consta de las siguientes fases:

- Proceso de análisis:
  - Identificación de los roles del sistema.
  - Identificar y documentación de los protocolos asociados para cada rol.
  - Elaboración en detalle del modelo de roles, empleando el modelo de protocolos como base.

- Proceso de diseño:
  - Creación de un *modelo de agente*, agregando roles a los tipos de agentes y documentando las instancias de cada tipo de agente.
  - Desarrollo de un *modelo de servicios*, examinando protocolos y propiedades de vida y seguridad.
  - Desarrollo de un *modelo de conocimiento*, por medio del modelo de interacción y el modelo de agente.

2.3.3.3.3 Metodología Cassiopea

La metodología Cassiopeia [Collinot et al., 1995] [Collinot et al., 1996] es una metodología orientada a agentes y basada en roles. Los conceptos básicos que maneja son: roles, agentes, dependencias y grupos.

Propone cinco etapas para el diseño de un MAS, organizadas en capas según puede verse en la Figura 2.23.

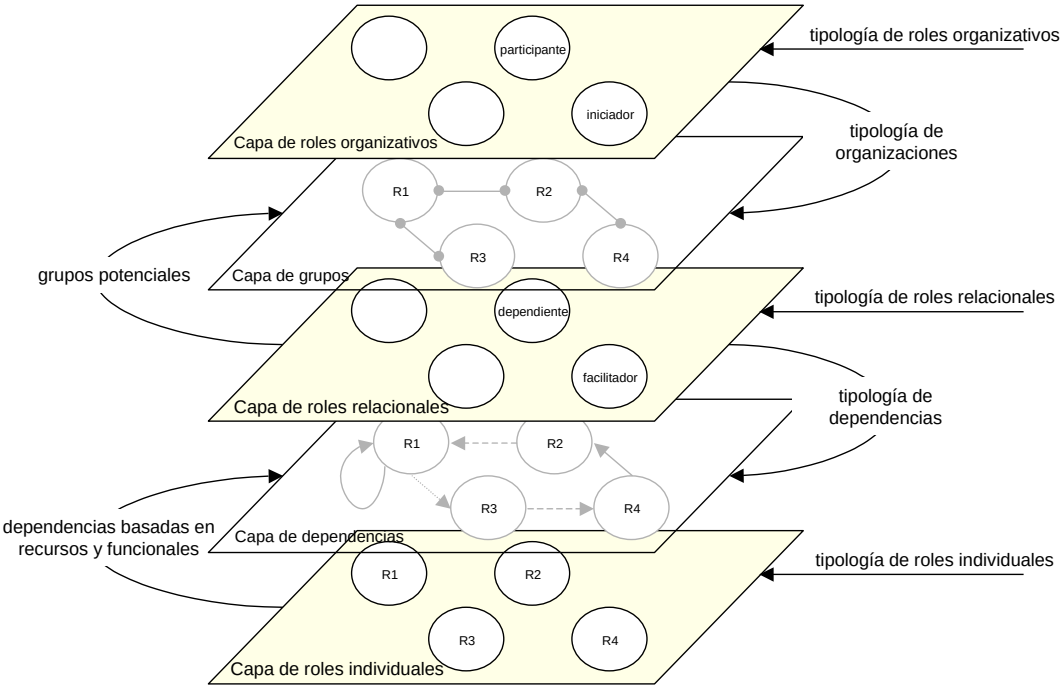


Figura 2.23 Capas en la metodología Cassiopea

1. Definición de los roles individuales para definir tipos de agentes (Capa de Roles Individuales).
2. Definición de dependencias entre roles (Capa de Dependencias).
3. Definición de la forma en que los agentes manejan esos roles (Capa de Roles Relacionales).
4. Definición de los grupos potenciales (Capa de Grupo).
5. Descripción de las dinámicas de estos grupos (Capa de Roles Organizativos).

2.4 ANÁLISIS BASADO EN FORMALISMOS MATEMÁTICOS

En este apartado se van a describir los método formales, que se basan en la especificación formal de los requisitos del sistema, lo que permite realizar una comprobación automática de la corrección de los mismos. Muchos métodos formales incorporan además la transformación automática de programas, lo que permite generar código automáticamente a partir de las especificaciones, garantizándose la corrección formal del proceso [NASA, 1997] [NASA, 1998]:

> *"Los métodos formales consisten en un conjunto de técnicas y herramientas basadas en modelado matemático y lógica formal que se utilizan para especificar y verificar requisitos y diseños de sistemas informáticos y software" [NASA, 1998].*

Estos métodos tienen gran interés en el campo de las aplicaciones de alto riesgo, como los sistemas de control de centrales nucleares o los sistemas automáticos de aviónica, dado que garantizan la corrección formal del programa resultante.

El formalismo matemático tiene dos grandes ventajas:

- Por un lado se puede verificar formalmente que una especificación es completa y consistente.
- Por otro lado, se pueden aplicar técnicas de demostración de teoremas para transformar la especificación en una implementación asegurando la corrección formal del proceso.

La gran limitación que tienen los métodos formales radica en la dificultad de la validación de la especificación, es decir, la comprobación de que la especificación se atiene al problema modelado. Esto sólo es posible si el problema permite realizar desde el principio una especificación completa de los requisitos del sistema. En palabras de Blum:

> *"En problemas con requisitos cerrados, habrá una alta confianza en la definición del problema y los mayores riesgos estarán asociados con verificar que el producto es correcto. Aquí los métodos formales son efectivos" [Blum, 1996].*

En la actualidad existe un gran número de métodos formales. Así, [Bowen, 2003] recoge un total de 91 notaciones, métodos o metodologías. En este estado de la cuestión se van a describir brevemente algunos de los más conocidos y referenciados: Z [ISO, 2002] [Spivey, 1992], VDM [Dawes, 1991] [ISO, 1996] [IFAD, 1999] y PVS [Owre et al., 2001a] [Rushby et al., 1996].

2.4.1 Z

La notación Z [Spivey, 1992] está basada en la teoría de conjuntos axiomática de Zermelo-Fraenkel [van Dalen et al., 1978] [Malitz, 1979] y en la lógica matemática. El desarrollo de Z comenzó en la Universidad de Oxford a finales de los años 70 [Hayes et al., 1993a] y ha culminado con su publicación como estándar ISO en el año 2002 [ISO, 2002].

Z es una notación matemática y no existe un método de desarrollo oficialmente asociado, aunque sí existen convenciones y prácticas habituales que facilitan su uso para especificar requisitos.

Lo que sigue es un breve resumen de los elementos fundamentales de la notación Z, basado fundamentalmente en [Spivey, 1992]. Los elementos fundamentales de Z son los siguientes:

- *Tipos de datos matemáticos* que permiten modelar de forma matemática los datos de un sistema. Estos tipos de datos no están orientados al computador, sino que obedecen las leyes de matemáticas, lo que permite razonar formalmente sobre el comportamiento de un sistema.
- *Lógica de predicados* como notación para representar de forma abstracta y declarativa el efecto de cada operación que pueda realizarse sobre el sistema.
- *Los esquemas*, como unidad mínima de especificación en Z, que permite dividir un documento en pequeños fragmentos que pueden presentarse progresivamente. Los esquemas permiten representar tanto aspectos estáticos como dinámicos:
  - *Aspectos estáticos*: los elementos que definen el estado de un sistema y las invariantes que deben cumplirse durante la evolución del sistema.
  - *Aspectos dinámicos*: las operaciones posibles, la relación entre sus entradas y salidas y los cambios de estado que ocurren al ejecutar operaciones.

  Además, Z define un lenguaje de esquemas, que permite relacionar y combinar varios esquemas para formar especificaciones más complejas.

Para describir estos elementos se va a seguir el ejemplo expuesto en [Spivey, 1992]: se trata de una agenda de cumpleaños, que almacena las fechas de cumpleaños de varias personas, y genera un aviso cuando llega uno de esos días.

Para este ejemplo no interesa conocer cómo se representan los nombres de las personas y las fechas, por lo que se definen como **tipos básicos**. Esto permite nombrar a estos conjuntos sin necesidad de saber qué tipos de objetos contienen:

[*NOMBRE*, *FECHA*]

El primer aspecto que hay que describir el sistema es su **espacio de estados**, para lo cual se define el siguiente esquema, de nombre AgendaCumpleaños:

∪ *AgendaCumpleaños* _____
→ *conocidos* : Π *NOMBRE*
→ *cumpleaños* : *NOMBRE* ♣ *FECHA*
∩_____
→ *conocidos* = dom *cumpleaños*
∠_____

En la mitad superior del esquema se definen las **variables** que definen el estado:

- *conocidos* es un conjunto de nombres de personas.
- *cumpleaños* es una función parcial que, para algunos nombres, devuelve su fecha de nacimiento.

En la mitad inferior del esquema se definen **relaciones** entre las variables. En este caso se trata de la **invariante** del estado, que especifica que el conjunto *conocidos* es exactamente igual al dominio de la función *cumpleaños* (es decir, que es el conjunto de aquellos nombres que tienen almacenada su fecha de cumpleaños).

Para definir **operaciones** sobre el estado también se escriben esquemas. En primer lugar se especificará la operación correspondiente a añadir un nuevo cumpleaños al estado del sistema (esquema *AñadirCumpleaños*).

En este esquema, la declaración *ΔAgendaCumpleaños* como parte de las variables indica que se está especificando un **cambio de estado** sobre el esquema *AgendaCumpleaños*, y equivale a la declaración de cuatro variables: dos para el estado inicial (*conocidos*, *cumpleaños*) y dos para el estado final, que se marcan con un apóstrofe (*conocidos'*, *cumpleaños'*).

∪ *AñadirCumpleaños* _____
→ Δ *AgendaCumpleaños*
→ *nombre?* : *NOMBRE*
→ *fecha?* : *FECHA*
∩_____
→ *nombre?* ™ *conocidos*
→ *cumpleaños'* = *cumpleaños* Y { *nombre?* □ *fecha?*}
∠_____

Después de la declaración de cambio de estado aparecen las **entradas**, que se indican como variables con una interrogación:

- *nombre?* representa un nombre de una persona nueva.
- *fecha?* representa la fecha de cumpleaños de esa persona.

En la segunda parte del esquema se muestra:

- Una **precondición** que especifica que el nombre que se va añadir no está almacenado ya en el sistema (es decir, no forma parte de los conocidos). Si no se cumple la precondición no se ejecutará la operación.
- Una **postcondición** que especifica que se inserta un nuevo par (*nombre?*, *fecha?*) en la función *cumpleaños*. Dada la invariante definida en el esquema *AgendaCumpleaños*, esto implicará que también se añade el nombre al conjunto *conocidos*.

Obsérvese que en Z no se hace una separación explícita entre la precondición y la postcondición, simplemente aparecen como partes de la especificación de las relaciones de un esquema.

Otra operación del sistema puede ser la obtención del cumpleaños de una persona (*EncontrarCumpleaños*). En este esquema se introducen dos nuevas notaciones:

- La declaración *Ξ AgendaCumpleaños* indica que el estado no cambia después de ejecutar la operación o, dicho de otro modo, que los estados inicial y final son iguales.
- La declaración *fecha!* indica que se trata de una variable de salida.

$\cup$ *EncontrarCumpleaños* _____
$\rightarrow \Xi$ *AgendaCumpleaños*
$\rightarrow$ *nombre?* : *NOMBRE*
$\rightarrow$ *fecha!* : *FECHA*
$\cap$_____
$\rightarrow$ *nombre?* ε *conocidos*
$\rightarrow$ *fecha!* = *cumpleaños*(*nombre?*)
$\angle$_____

En el esquema *EncontrarCumpleaños* se ha especificado como precondición que el nombre de entrada debe ser conocido y como postcondición que la fecha de salida es precisamente la fecha asignada a ese nombre.

La operación más interesante del sistema es aquella que devuelve las personas que cumplen los años un determinado día, para que se les pueda enviar tarjetas de felicitación. Esta operación se define mediante el siguiente esquema (*Recordar*).

$\cup$ *Recordar* _____
$\rightarrow \Xi$ *AgendaCumpleaños*
$\rightarrow$ *hoy?* : *FECHA*
$\rightarrow$ *tarjetas!* : Π *FECHA*
$\cap$_____
$\rightarrow$ *tarjetas!* = { *n* : *conocidos* | *cumpleaños*(*n*) = *hoy*? }
$\angle$_____

En este esquema se especifica que el conjunto de salida (*tarjetas!*) es un subconjunto de las personas conocidas formado por aquellas personas que cumplen los años el día pasado como entrada (*hoy?*).

Para completar la especificación debe indicarse cuál es el **estado inicial** del sistema, para lo que es necesario definir un nuevo esquema:

$\cup$ *IniciarAgendaCumpleaños* _____
$\rightarrow$ *AgendaCumpleaños*
$\cap$_____
$\rightarrow$ *conocidos* = 0
$\angle$_____

Este esquema (*IniciarAgendaCumpleaños*) indica que el estado inicial es aquél en el que no se conoce ninguna persona (*conocidos* está vacío) y como consecuencia de ello, tampoco se conoce el cumpleaños de nadie (la función *cumpleaños* está vacía, aplicando la invariante de *AgendaCumpleaños*).

La especificación anterior puede ampliarse para indicar qué ocurre cuando no se cumplen las precondiciones de las operaciones. Para ello se deberían crear nuevos esquemas con esas excepciones y luego se pueden combinar los esquemas normales y los esquemas de error para formar la especificación completa.

A partir de una especificación como la anterior, Z permite dos acciones [Spivey, 1992]:

- En primer lugar, se puede *verificar formalmente* que la especificación es correcta y que no se incumple ninguna de las invariantes.

- En segundo lugar, se puede ir *refinando progresivamente* esta especificación aumentando el grado de detalle de los datos (refinado de datos) y de las operaciones (refinado de operaciones o desarrollo de algoritmos), con el fin de ir acercándose poco a poco a un modelo más cercano al diseño y la implementación del sistema final. Este refinado progresivo garantiza la corrección del producto final puesto que en todo momento se mantienen las definiciones formales.

2.4.2 VDM

VDM (del inglés *Vienna Development Method*, es decir, método de desarrollo de Viena) es un método formal de desarrollo de software, dentro del cual se utiliza una notación de especificación formal llamada VDM-SL [Dawes, 1991]. VDM tiene su origen en el laboratorio de IBM de Viena, que, durante los años 60 y primeros de los 70 dedicó grandes esfuerzos a la formalización de la semántica del lenguaje de programación PL/I utilizando semántica denotativa. Estos esfuerzos culminaron con la definición del método VMD a mediados de los años 70 [Hayes et al., 1993a]. El trabajo sobre la notación VDM-SL ha continuado desde entonces, culminando con su publicación como estándar ISO a mediados de los años 90 [ISO, 1996].

En VDM-SL se admiten dos variantes de la notación. La primera usa símbolos matemáticos (de la misma forma que Z) mientras que la segunda sólo utiliza caracteres ASCII. Esta segunda variante facilita la creación de especificaciones con cualquier editor de texto [ISO, 1996] [IFAD, 1999]. En cualquiera de las dos variantes los distintos elementos de una especificación van identificados con palabras clave (en vez de las "cajas" que usa Z).

Para explicar los elementos fundamentales de una especificación realizada en VDM-SL se va a especificar el ejemplo de la agenda de cumpleaños visto en el apartado anterior.

Lo primero que hay que destacar es que existen dos formas de construir especificaciones en VDM-SL. En la primera se crean especificaciones completas en un único documento, que se denominan *especificaciones planas*. La segunda forma consiste en dividir una especificación compleja en varios documentos, cada uno de los cuales se denomina módulo. Las especificaciones con este formato se denominan *especificaciones modulares*, y tienen la ventaja de permitir la reutilización de módulos en las especificaciones de problemas distintos [ISO, 1996].

En el ejemplo de la agenda de cumpleaños y dado su reducido tamaño, basta con definir una especificación plana. Las especificaciones de este tipo comienzan con una primera sección (*types*) en la que se definen los tipos y sus invariantes.

Así, en el ejemplo de la agenda de cumpleaños, lo primero sería definir los tipos genéricos con los que se representan los nombres y las fechas.

```
types
   NOMBRE : token
   FECHA: token
```

Los tipos *NOMBRE* y *FECHA* se han definido como tipos de datos sin restricciones, (es decir, si definir), mediante el uso del tipo *token* de VDM-SL.

El siguiente paso consiste en definir cómo es el estado del sistema especificado. Para ello se define una sección específica (*state*). Los elementos definidos en esta sección representan el estado global de la especificación y pueden actuar como variables globales en las definiciones de operaciones. La definición de estados permite especificar también su invariante e inicialización.

```
state AgendaCumpleaños of
   conocidos :  NOMBRE-set
   cumpleaños :  NOMBRE ──m──→ FECHA
   inv  mk-AgendaCumpleaños(cn, cmp)  Δ
      cn = dom cmp
   init a  Δ
      a = mk-AgendaCumpleaños({}, {})
end
```

Esta definición de estado tiene los siguientes elementos:

- Las dos variables que representan el estado: el conjunto *conocidos* (conjunto de elementos del tipo *NOMBRE*) y la función parcial *cumpleaños*.
- La invariante del estado, que dice que para cualquier agenda formada por los elementos *cn* (de conocidos) y *cmp* (de cumpleaños) debe cumplirse que *cn* sea igual al dominio de la función *cmp*.
- La inicialización del estado con el conjunto y la función vacíos.

Las operaciones se definen dentro de una sección *operations* que se sitúa a continuación de los tipos y estados. VDM-SL admite dos tipos de especificación de operaciones: implícitas (especificando precondición y postcondición) o explícitas (con instrucciones de programa). En el ejemplo se va a definir las operaciones de forma implícita.

Así, la operación que permite añadir una nueva fecha de cumpleaños podría definirse de la siguiente manera:

```
operations
   AñadirCumpleaños(n:NOMBRE, f:FECHA)
   pre n ∉ conocidos
   post cumpleaños = cumpleaños~ Y { n □ f } ∧
         conocidos = conocidos~ Y { n }
...
```

En la especificación de esta operación se recogen los siguientes elementos:

- La cabecera, en la que aparece el nombre de la operación (*AñadirCumpleaños*) y sus parámetros de entrada (n y f).
- La precondición, que indica que el parámetro *n* no debe pertenecer al conjunto de nombres conocidos del estado.
- La postcondición, que indica que la función *cumpleaños* contiene un nuevo par y que el conjunto *conocidos* contiene un nuevo elemento. En las postcondiciones se distingue el estado inicial de las variables mediante el carácter tilde (~) después del nombre de la variable (por ejemplo, *conocidos*~).

La segunda operación del sistema es la que permite obtener la fecha de cumpleaños de una persona:

EncontrarCumpleaños (*n:NOMBRE) f:FECHA*
pre *n* ε *conocidos*
post *f* = *cumpleaños*(*n*)

Puede observarse que las variables que definen la salida de la operación se representan a continuación de los parámetros (entrada de la operación).

Para terminar este ejemplo queda por especificar la operación *Recordar*, devuelve el conjunto de personas que cumplen los años un determinado día.

Recordar (*hoy:FECHA) tarjetas:NOMBRE*-set
post *tarjetas* = { *n* : *conocidos* | *cumpleaños*(*n*) = *hoy?* }

Además de los mostrados en esta breve presentación, la notación VDM-SL contiene muchos más elementos. Entre ellos destaca la posibilidad de definir de forma explícita excepciones y gestión de errores en las operaciones.

2.4.3 PVS

PVS viene del inglés *Prototype Verification System* (sistema de verificación de prototipos) [Butler, 1995] [Butler, 1996] [Crow et al., 1995] [Owre et al., 2001a] [Owre et al., 2001b] [Rushby et al., 1996] [Vitt et al., 1996]. Se trata de un sistema de verificación que proporciona un lenguaje de especificación integrado con un conjunto de herramientas de soporte y un demostrador de problemas.

PVS es un producto de la empresa "SRI International" y es el último de una serie de sistemas dentro de una línea de trabajo en lenguajes de especificación, demostradores de teoremas y sistemas de verificación que empezó en los años 70 [Owre et al., 2001a]. Dentro de esta línea se incluyen, entre otros, el Sistema de Verificación "Jovial" [Elspas et al., 1979], la Metodología de Desarrollo Jerárquico (HDM) [Robinson et al., 1979] [Spitzen et al., 1978], STP [Shostak et al., 1982] y Ehdm [Melliar-Smith et al., 1985] [Rushby et al., 1991].

El lenguaje de especificación de PVS está basado en lógica clásica de orden superior con tipos [Owre et al., 2001a]. Se admiten tipos no interpretados (sin definir), tipos básicos que incluyen los booleanos, enteros, reales, ordinales, etc., y construcción de nuevos tipos a partir de funciones, conjuntos, tuplas, predicados, etc. Las especificaciones de PVS se descomponen en teorías (similares a los módulos de VDM-SL) que pueden contener suposiciones, definiciones, axiomas y teoremas.

Al igual que ocurre con VDM-SL, en PVS hay dos notaciones del lenguaje. La primera sólo incluye caracteres ASCII y es la que se escribe para ser tratada por las herramientas de PVS. La segunda incluye caracteres matemáticos y se genera de forma automática a partir de la notación ASCII.

Para describir de forma resumida las características del lenguaje de PVS se va a volver a utilizar el ejemplo de la agenda de cumpleaños, usando como base un ejemplo similar (una agenda telefónica) utilizado en [Crow et al., 1995]. Dado que la notación

matemática nunca se utiliza para escribir las especificaciones, se va a usar la notación ASCII.

Lo primero es definir los tipos que representan nombres y fechas, que se especificarán como tipos sin definir:

```
NOMBRE : TYPE
FECHA : TYPE
```

Lo siguiente es definir la representación de la agenda de cumpleaños, como una función del conjunto de los nombres al conjunto de las fechas:

```
Cumpleaños : TYPE [NOMBRE -> FECHA]
```

Al describir Z y VDM-SL, *cumpleaños* se definió como una función parcial. En PVS, sin embargo, no existen las funciones parciales, por lo que es la función anterior es total. Para poder representar el hecho de que haya nombres sin fechas se hace necesario realizar una serie de definiciones adicionales:

- En primer lugar se debe definir una constante que represente una fecha nula. Esta fecha se asignará a todos los nombres que no tengan fecha de cumpleaños.

```
fnula : FECHA
```

- En segundo lugar se define un subtipo de *FECHA*, que contiene aquellas fechas distintas de la fecha nula.

```
FECHABUENA : TYPE = {f: FECHA / f /= fnula}
```

A partir de este punto se pueden definir las distintas condiciones que permitirán reflejar el estado inicial, invariantes y operaciones de la especificación. Con el fin de hacer más breves estas declaraciones se empieza por definir las variables que se usarán como parámetros:

```
no : VAR NOMBRE
fe : VAR FECHA
ag : VAR Cumpleaños
fb : VAR FECHABUENA
```

La primera función va a indicar cómo es una agenda vacía: aquella en la que todos los nombres tienen asignada la fecha nula:

```
agendavacía(no): FECHA = fnula
```

En esta declaración se ha definido la función *agendavacía*, que toma como parámetro un nombre (*no*) y devuelve un valor de tipo *FECHA*, que en este caso siempre es igual a la fecha nula (*fnula*).

La siguiente definición es la operación de consulta del cumpleaños de una persona que es muy sencilla, dado que la función es total y, por lo tanto, todos los nombres tienen alguna fecha asociada (aunque sea la fecha nula).

```
EncontrarCumpleaños(ag, no): FECHA = ag(no)
```

En esta definición la función recibe una agenda de cumpleaños y un nombre, y devuelve la fecha asociada a ese nombre en la agenda. En PVS no se usan variables globales para

definir el estado del sistema, por lo que todas las operaciones de consulta o de actualización deberán recibir la agenda como parámetro.

Para definir la operación de inserción de nuevos cumpleaños es conveniente definir previamente una función que diga si un nombre es conocido en una agenda (en otras notaciones formales esto se representaba como un tipo, pero en PVS es más adecuado hacerlo como función):

```
Conocido?(ag, no): BOOL = ag(no) /= fnula
```

La función *Conocido?* recibe devuelve un valor booleano que es cierto si la fecha de la persona no es nula.

A partir de la función anterior se puede definir la operación de inserción de cumpleaños:

```
AñadirCumpleaños(ag, no, fb): Cumpleaños =
   IF Conocido?(ab, no)
     THEN ag
     ELSE ag WITH [(no) := fb]
   ENDIF
```

La función *AñadirCumpleaños* recibe como parámetros una agenda, un nombre y una fecha no nula y devuelve la agenda modificada. En esta definición se indica que, si el nombre ya es conocido en la agenda entonces no se hace nada (se devuelve la misma agenda). En otro caso se devuelve la agenda modificada, con la fecha asignada al nombre. Obsérvese cómo en PVS no se utilizan precondiciones ni poscondiciones, sino que se simplemente se dice cómo es el resultado de la operación mediante las sentencias de control del propio lenguaje.

La última operación es la consulta de todos los nombres de personas que cumplen los años en un día concreto:

```
Recordar(ag, fb): setof[NOMBRE] =
   {n: NOMBRE / ag(n) = fb}
```

Con esto terminaría la especificación propiamente dicha de la agenda de cumpleaños. Sin embargo en PVS, dado que incluye un demostrador de teoremas, se pueden añadir más elementos cuya veracidad se desea comprobar, como parte de la verificación de la especificación.

Como un ejemplo se muestra a continuación una conjetura que dice que si se añade un nuevo nombre a la agenda, entonces su fecha será la que se insertó:

```
EncontrarAñadir: CONJECTURE
 NOT Conocido?(ag, no) =>
  EncontrarCumpleaños(AñadirCumpleaños(ag,no,fb),no) = fb
```

El usuario de PVS puede decirle al sistema que demuestre la veracidad de esta conjetura, lo cual realizará de forma automática, diciendo si es o no cierto que se cumple esta condición a partir de todos los elementos de la teoría. De esta forma se podrá comprobar de forma automática que la especificación es correcta.

Para concluir el ejemplo se mostrará la teoría completa, en el formato matemático generado automáticamente por el sistema PVS:

```
AgendaCumpleaños : THEORY
  BEGIN
  NOMBRE : TYPE
  FECHA : TYPE
  Cumpleaños : TYPE [NOMBRE ϕ FECHA]
  fnula : FECHA
  FECHABUENA : TYPE = {f: FECHA / f /= fnula}
  no : VAR NOMBRE
  fe : VAR FECHA
  ag : VAR Cumpleaños
  fb : VAR FECHABUENA

  agendavacía(no): FECHA = fnula

  EncontrarCumpleaños(ag, no): FECHA = ag(no)

  Conocido?(ag, no): bool = ag(no)⌊ fnula

  AñadirCumpleaños(ag, no, fb): Cumpleaños =
    IF Conocido?(ab, no) THEN ag ELSE ag WITH [(no) := fb] ENDIF

  Recordar(ag, fb): setof[NOMBRE] = {n: NOMBRE | ag(n) = fb}

  EncontrarAñadir: CONJECTURE
    ¬Conocido?(ag, no) ⇒ EncontrarCumpleaños(AñadirCumpleaños(ag, no, fb), no) = fb

  END AgendaCumpleaños
```

2.5 ANÁLISIS BASADO EN EL PARADIGMA DEL CONOCIMIENTO

Las metodologías que se recogen en este apartado provienen de la disciplina de la Ingeniería del Conocimiento [Gómez et al., 1997] [Studer et al., 1998], ya definida en la introducción, cuyo objetivo es la construcción de sistemas informáticos que utilicen conocimientos de forma explícita.

Desde los inicios de esta disciplina, el mayor problema afrontado fue la obtención de estos conocimientos a partir de diversas fuentes, entre las que destaca el experto humano. Por ese motivo los métodos de Ingeniería del Conocimiento ponen un énfasis especial en conocer (comprender) el problema antes de comenzar su desarrollo.

Existe una gran cantidad de métodos y metodologías de Ingeniería del Conocimiento [Studer et al., 1998], pero sólo un escaso número de ellas tiene alguna relevancia. Se van a describir dos de ellas: IDEAL [Gómez et al., 1997] y CommonKADS [Schreiber et al., 1999].

2.5.1 IDEAL

La metodología IDEAL ha sido desarrollada en la Facultad de Informática de la Universidad Politécnica de Madrid [Alonso et al., 1996] [Ares et al., 1998] [Gómez et al., 1997] [Gómez et al., 2000]. Su nombre viene de las iniciales de sus cinco fases, que se recogen en la Figura 2.24. En esta figura aparecen las fases en el centro, los factores que juegan en contra del desarrollo a la izquierda, y los factores que facilitan el desarrollo a la derecha.

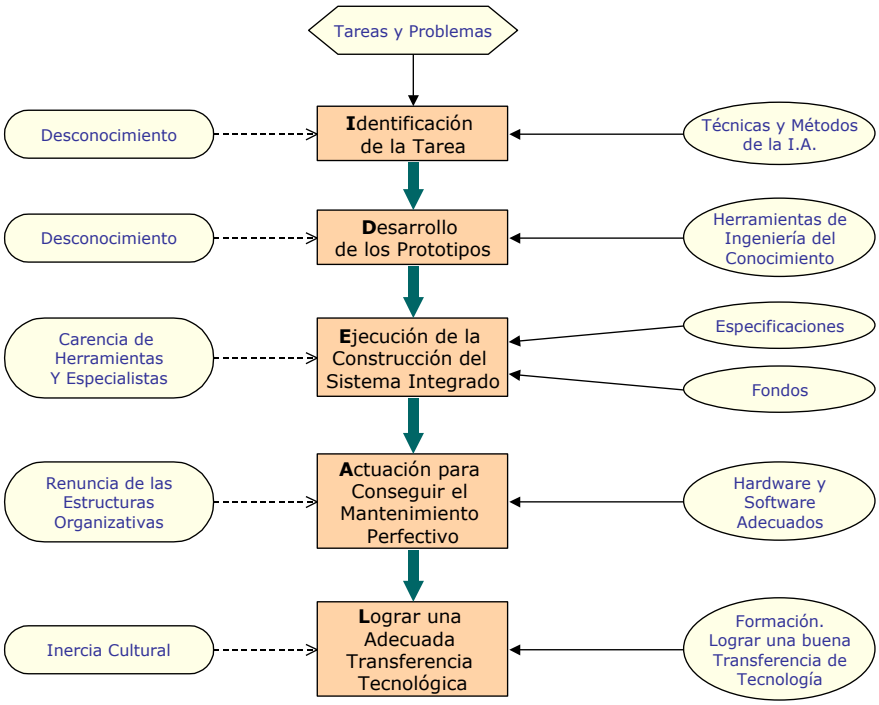


Figura 2.24 Fases de la metodología IDEAL [Gómez et al., 1997]

Seguidamente se describen de forma resumida las fases y etapas de la metodología IDEAL, según [Gómez et al., 1997]:

1. **Identificación de la tarea**. En esta fase se definen los objetivos generales de la aplicación y, a partir de los mismos, se determina si la tarea es susceptible o no de ser tratada con la tecnología de la Ingeniería del Conocimiento. Las etapas de esta fase son:

   a. *Plan de Requisitos y Adquisición de Conocimientos*. En esta etapa se determinan las necesidades del cliente mediante una especificación informal. Se debe comenzar con la adquisición de conocimientos, dado que generalmente es necesaria para entender los objetivos del sistema.

   b. *Evaluación y Selección de la Tarea*. En esta etapa se realiza un estudio de viabilidad en cuatro dimensiones: posibilidad, justificación, adecuación y éxito.

   c. *Definición de las Características de la Tarea*. Se establecen y definen las características más relevantes asociadas con el desarrollo de la aplicación, que incluye: una especificación completa de requisitos, criterios de éxito, casos de prueba, recursos ara el desarrollo, análisis de costes / beneficios, hitos y calendario.

2. **Desarrollo de los Prototipos**. Aquí se desarrollan los distintos prototipos que permiten ir definiendo y refinando las especificaciones del sistema. En esta metodología se identifican los siguientes prototipos: de demostración, de investigación, de campo y de operación. Para el desarrollo de cada prototipo se siguen las siguientes etapas:

a. *Concepción de la Solución*. En el primer prototipo se produce un diseño general del sistema. En los subsiguientes se refina la concepción de la solución presentada en el prototipo anterior.

b. *Adquisición de Conocimientos y Conceptualización de los Conocimientos*. Se adquieren y conceptualizan los conocimientos que utilizará el prototipo correspondiente.

c. *Formalización de los Conocimientos*. Se seleccionan los formalismos para representar en la máquina los conocimientos obtenidos en la etapa anterior y se realiza un diseño detallado del sistema.

d. *Implementación*. Se programa el sistema, bien usando una herramienta de Ingeniería del Conocimiento, bien usando un lenguaje de programación de propósito general.

e. *Validación y Evaluación*. Se evalúa el prototipo construido mediante casos de prueba y ensayos en paralelo.

f. *Definición de nuevos requisitos, especificaciones y diseño*. En esta etapa se definen los requisitos, especificaciones y diseño del siguiente prototipo, hasta que se obtenga el sistema final.

3. **Ejecución de la Construcción del Sistema Integrado**. Los Sistemas Basados en el Conocimiento suelen formar parte de sistemas computacionales más generales con los que interactúan. El objetivo de esta fase es realizar esta integración, mediante tres etapas:

   a. *Requisitos y Diseño de la Integración* con otros sistemas. Consiste en el estudio y diseño de interfaces y puentes con otros sistemas hardware y software.

   b. *Implementación y Evaluación de la Integración*. Se implementa la integración del Sistema Basado en el Conocimiento con el resto de sistemas.

   c. *Aceptación* por el usuario del sistema final.

4. **Actuación para Conseguir el Mantenimiento Perfectivo**. Esta fase trata el mantenimiento del sistema, con especial énfasis en el mantenimiento perfectivo, que incluye la incorporación de nuevas funcionalidades y nuevos conocimientos. Se tienen las siguientes etapas:

   a. *Definir el Mantenimiento del sistema global*. Se emplean técnicas de Ingeniería del Software, definiendo el mantenimiento que se llevará a cabo.

   b. *Definir el Mantenimiento de las bases de conocimientos*. Hay que dedicar una etapa especial a estudios del mantenimiento de los conocimientos.

   c. *Adquisición de nuevos conocimientos*. Se diseñan protocolos para captar y registrar los nuevos conocimientos que pudieran aparecer al usar el sistema.

5. **Lograr una Adecuada Transferencia Tecnológica**. En esta fase se trata la transferencia tecnológica para conseguir que el sistema sea manejado de forma adecuada por sus usuarios finales. Para ello son necesarias las siguientes tareas:

a. *Organizar la transferencia tecnológica*, mediante el entrenamiento en sesiones de tutoría que sirvan para explicar el manejo del sistema y facilitar la comprensión de su documentación.

b. *Completar la documentación del sistema*, desde el dossier técnico al manual de usuario, que deben incorporar todas las peculiaridades de su uso de una forma amigable para el usuario final a quien debe ir dirigido.

Las fases y etapas descritas siguen en la metodología IDEAL un ciclo de vida tronco-cónico en espiral (véase la Figura 2.25).

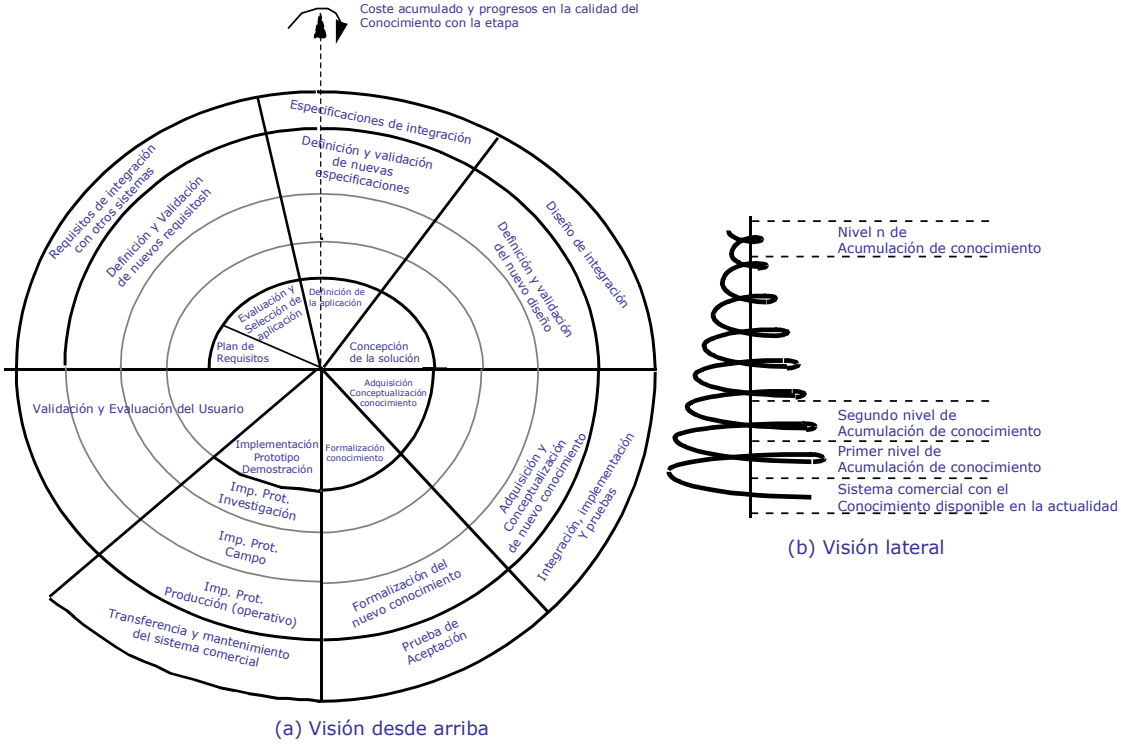


Figura 2.25 Modelo tronco-cónico del ciclo de vida de la metodología IDEAL

Dentro de todas las fases y etapas de IDEAL, la que más se ajusta al contexto de esta Tesis es la conceptualización, cuyos elementos fundamentales se describen a continuación.

En primer lugar, el proceso de conceptualización de IDEAL se realiza de forma coordinada con la adquisición de conocimientos y está basado en las reglas del método cartesiano (evidencia, análisis, síntesis, prueba), con dos tareas fundamentales (Figura 2.26):

1. **Análisis**: consiste en clasificar los conocimientos en cuatro categorías: estratégicos, tácticos, factuales y metaconocimientos.

   a. *Conocimientos estratégicos* o de control, que especifican qué hacer, dónde y por qué hacerlo. Estos conocimientos fijan la secuencia de pasos que deberá seguir el sistema para ejecutar su tarea. Dentro de los conocimientos estratégicos se identifican los siguientes elementos:

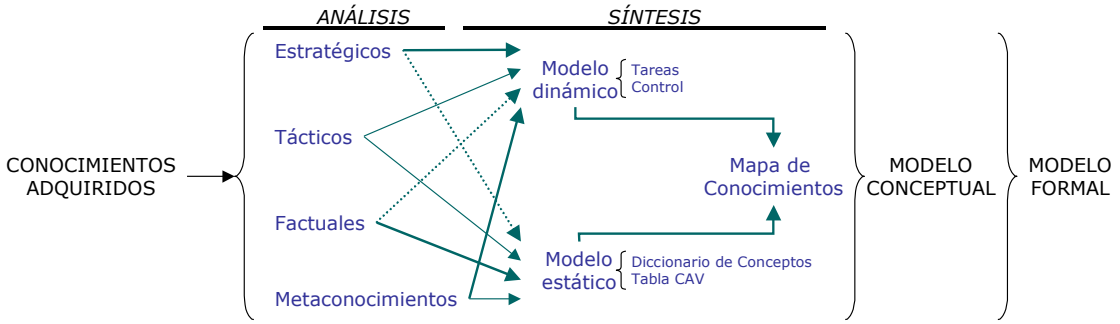


Figura 2.26 Análisis y síntesis en la conceptualización de IDEAL

1. *Pasos de alto nivel y su secuencia de control*
2. *Subpasos intermedios*, que a su vez se descomponen en pasos más sencillos.
3. *Subpasos de bajo nivel*, que ya no se descomponen más. Para cada uno se debe detallar la información recogida en la Tabla 2.7.

Tabla 2.7 Especificación de subpaso de bajo nivel en IDEAL

Campo	Descripción
Nombre	Nombre del subpaso.
Precondición	Condiciones que deben cumplirse para ejecutar el paso.
Entrada	Datos de entrada.
Acciones	Secuencia de acciones que deberían realizarse.
Razonamiento	Indicación de las inferencias realizadas.
Salida	Acciones de salida.

Para representar la división de pasos en subpasos, los autores de IDEAL recomiendan dos representaciones gráficas alternativas, según se muestra en Figura 2.27: un árbol de descomposición funcional (si la estructura es jerárquica), o un grafo dirigido (si es heterárquica).

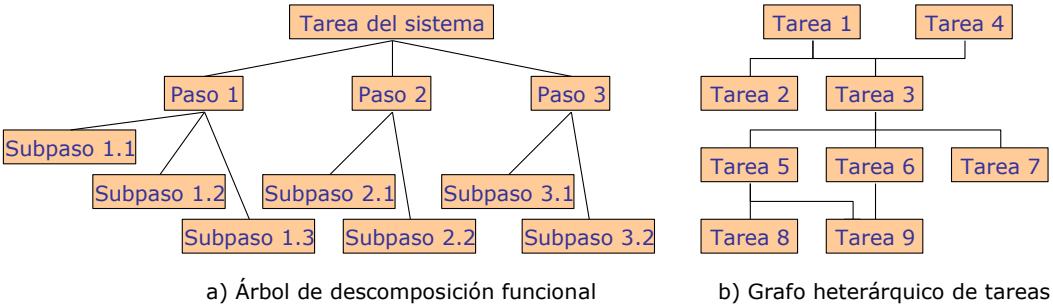


Figura 2.27 Diagramas para representar conocimientos estratégicos en IDEAL

b. *Conocimientos tácticos*, de acción u operativos, que especifican cómo y cuándo añade el sistema información a sus conocimientos acerca del caso. Se buscan fundamentalmente inferencias e incertidumbres y se especifican mediante alguna de las siguientes representaciones intermedias:

   1. *Tablas de decisión (Tabla 2.8)*, que se utilizan para representar inferencias en las que se conoce el resultado (atributo conclusión) para todas las posibles combinaciones de valores de la entrada (atributos básicos).

Tabla 2.8 Formato de tablas de decisión

Atributo básico 1	Atributo básico 2	...	Atributo básico n	**Atributo conclusión**
Valor 1	Valor 1	...	Valor 1	**Valor a**
Valor 2	Valor 1	...	Valor 2	**Valor b**
Valor 3	Valor 1	...	Valor 5	**Valor c**
...	...	...	...	**...**

1. *Árboles de decisión* (*Figura 2.28*), que representan una lógica de razonamiento en las que cada nodo del árbol es una pregunta que se plantea y los arcos (que llevan a otros nodos) son cada una de las respuestas a la pregunta. Los arcos pueden llevar a conclusiones (resultados) o bien a nuevas preguntas.

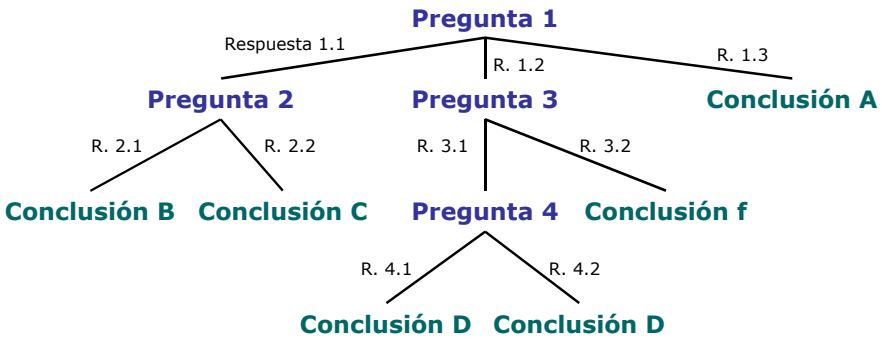


Figura 2.28 Formato de representación de árboles de decisión

2. *Seudorreglas*, representan reglas en la forma "si condición entonces acciones", escritas en términos del experto. Estas reglas se representan en el formato especificado en la Tabla 2.9.

Tabla 2.9 Formato de una Hoja de Reglas en IDEAL

Número de referencia del planteamiento de trabajo	**Adquirida de (X) Confirmada por (Y)**	**Fecha**
Número de entrevista o documento de tiempo	X Y	DD.MM.AAAA DD.MM.AAAA
Estado de la regla	**Texto de la regla**	
Palabras del experto		
Formulación externa de la regla		
Formulación en la herramienta		
Nombre de la regla		

3. *Fórmulas*, representan el cálculo de valores de salida a partir de los valores de entrada. Se especifican mediante la definición mostrada en la *Tabla 2.10*.

c. *Conocimientos factuales* o declarativos, que especifican lo que es verdad acerca del mundo en general y acerca del caso particular para el cual se está ejecutando la tarea. Estos conocimientos contienen, por un lado, información que el sistema conocerá a priori acerca del área de aplicación y, por otro, información que obtendrá el sistema cuando ejecute su tarea. Los pasos de la identificación de conocimientos factuales son los siguientes:

Tabla 2.10 Especificación de fórmulas en IDEAL

Información	Descripción
Atributo conclusión	Atributo que puede ser calculado con la fórmula.
Fórmula	Algoritmo o fórmula matemática.
Descripción	Se proporcionan explicaciones adicionales si la fórmula es compleja, experimental o si es una aproximación.
Atributos básicos	Lista los atributos básicos (entrada) de la fórmula.
Precisión	Precisión con la que debería calcularse.
Restricciones	Si la fórmula sólo es válida en ciertas situaciones hay que describirlas.
Referencias cruzadas	Referencias a documentos de adquisición donde aparece la fórmula, así como referencias a fórmulas relacionadas.
Estatus de implementación	Indicar si el uso de la fórmula ha sido comprobado y si la fórmula implementada trabaja como se desea.

1. *Crear definiciones de atributos.* Se recopilan todos los atributos usados como datos de entrada, conclusiones de inferencia o resultados de salida. Cada uno de los atributos se especifica usando el formato de la Tabla 2.11.

Tabla 2.11 Definición de atributo en IDEAL

Información	Descripción
Nombre	Nombre del atributo.
Descripción	Breve explicación del significado del atributo.
Tipo valor	Clase de valores para el atributo. Si son numéricos se muestran también las unidades de medida y precisión.
Rango de valores	Lista de valores, rango de valores o bien estructura de los valores correctos.
Número de valores	Si el número de valores simultáneos puede ser uno o muchos.
Fuente	Si es atributo de entrada, se indica de dónde procede el valor del atributo y qué hacer si no se puede obtener.
Detalles sobre método para obtenerlo	Si es de entrada, explicación de cómo se obtiene el valor y se chequea si es correcto.
Confiabilidad	Si la entrada no siempre es confiable, indicar cómo superar ese problema.
Uso	Indicación de cómo se usa en el sistema: entrada, conclusión, salida, etc.
Formato de salida	Si es de salida, indicación del formato con el que se presenta a usuarios o se envía a otros sistemas.
Material de soporte	Referencias a documentos de adquisición donde se proporciona información acerca del atributo.

1. *Clasificar atributos*. En este paso, los atributos identificados se agrupan, formando conceptos, que se recogen en el diccionario de conceptos.
2. *Organizar los hechos dependientes del caso*. Se trata de identificar qué hechos (valores de atributos) son propios de cada caso y cuáles son independientes de los casos. Los hechos independientes representan valores conocidos cuando comienza la tarea del sistema.
3. *Organizar relaciones independientes del caso*. Se identifican relaciones entre conceptos que no son hechos y que se utilizan en el proceso de razonamiento.

d. *Metaconocimientos*, que representan la forma en la que el sistema usará sus conocimientos para tomar una decisión. Se denominan metaconocimientos porque normalmente trabajan sobre otros conocimientos más explícitos, por ejemplo, definiendo bajo qué condiciones una tarea tiene más prioridad que otra. Estos metaconocimientos pueden representarse en forma de reglas (llamadas en ese caso metarreglas) o bien usando otra representación de control, como los organigramas.

2. *Síntesis*: consiste en transformar estos elementos en dos modelos, uno estático y otro dinámico que terminan integrándose en el mapa de conocimientos.

   a. *Modelo estático*. Este modelo recoge información sobre los conceptos, atributos y relaciones que conforman el conocimiento del dominio del problema. Para ello se consideran los siguientes documentos:

      1. *Diccionario de Conceptos*. En este diccionario, que se va creando a lo largo del proceso de conceptualización, se recogen todos los conceptos del sistema y, para cada uno, la información que muestra la *Tabla 2.12*.

Tabla 2.12 Campos de una entrada el Diccionario de Conceptos en IDEAL

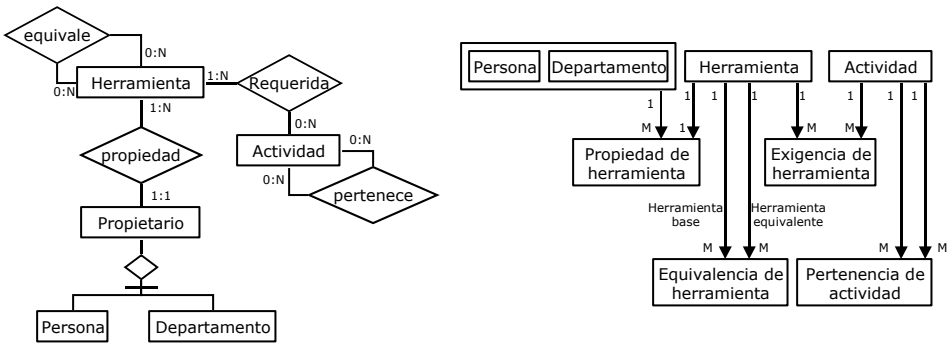
Información	Descripción
Concepto	Nombre del concepto.
Función	Indicación del uso dentro del sistema.
Sinónimos / Acrónimos	Otros términos usados para este concepto.
Atributos	Listado de los atributos del concepto.
Derivado de	Indicación de dónde se ha obtenido información sobre el concepto.

      2. *tabla Concepto / Atributo / Valor* (CAV). Esta tabla está muy relacionada con el diccionario de conceptos, pero se suele representar separada por cuestiones de claridad. Recoge, para cada concepto, sus atributos y los valores posibles para cada uno de los atributos, según el formato mostrado en la *Tabla 2.13*. Debe destacarse que en esta tabla se usan atributos para representar relaciones entre conceptos.

Tabla 2.13 Campos de una entrada de la Tabla CAV en IDEAL

Campo	Descripción
Concepto	Nombre del concepto.
Atributo	Lista de los atributos del concepto.
Posible valor	Especificación de los valores posibles de cada atributo. Puede ser la especificación de un tipo de valores, un rango de valores o una enumeración de valores.

      3. *Esquema o diagrama conceptual*. Una vez definidos todos los conceptos se procede a especificar las relaciones entre los mismos. Para ello los autores de IDEAL comentan que puede utilizarse el Diagrama Entidad Relación de Chen, pero proponen además su propia notación, basada en la idea de dependencia: un concepto (que representa una relación) depende de otros (los participantes en la relación). En la *Figura 2.29* se muestra un modelo de conceptos usando ambas notaciones.



a) Diagrama Entidad/Relación b) Diagrama basado en dependencias

Figura 2.29 Notaciones para diagramas conceptuales en IDEAL

b. *Modelo dinámico* o de proceso. En el modelo dinámico se recopilan de forma sintética los elementos estratégicos y tácticos en forma de tareas y procedimientos. Asimismo se relacionan los procedimientos con los elementos del modelo estático.

   1. *Especificación de tareas*. Se realiza un grafo jerárquico o heterárquico de las tareas del sistema (y sus subtareas). Para cada tarea se especifica su meta, entradas, salidas, llamadas a procedimientos externos y número de instancias.

   2. *Especificación de procedimientos*. Las tareas que no se descomponen deben describirse, recogiéndose su nombre, propósito, información necesaria, acciones, razón fundamental, notas de entrevistas (relacionándolo con documentos de adquisición) y estatus de implementación.

   3. *Modelo de proceso*. Para cada tarea se realiza un diagrama, basado en el diagrama conceptual, en el que se indica qué elementos participan en la tarea. Por ejemplo, la *Figura 2.30* muestra el modelo de proceso en el caso de una tarea "Determinar herramienta para trabajo": participan los conceptos herramienta, necesidad de herramienta, actividad y pertenencia de actividad.

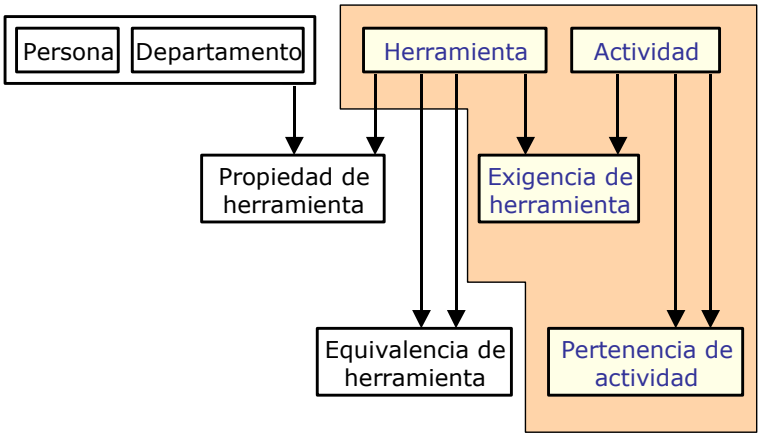


Figura 2.30 Ejemplo de modelo de proceso en IDEAL

c. *Mapa de conocimientos*. Representa la síntesis de los modelos dinámico y estático, mostrando gráficamente la relación dinámica que se establece entre los distintos atributos durante el proceso de razonamiento. Este mapa es

fundamental para validar la conceptualización con el experto. En el diagrama se muestra una caja para cada atributo y se representan con flechas las relaciones entre los atributos básicos y conclusión de cada inferencia del proceso (véase la Figura 2.31). Los pasos que deben seguirse para construir el mapa de conocimientos son los siguientes:

1. *Identificar las decisiones* de la tarea hechas por el sistema (metas).
2. *Diseñar el bloque de la decisión meta*, en el que se representa el atributo y sus posibles valores.
3. *Añadir atributos para inferir la decisión meta*, que se colocan alrededor del bloque que representa la meta.
4. *Ampliar el Mapa*, añadiendo los atributos necesarios para inferir los que van apareciendo. Se sigue así hasta que se han recopilado todos los atributos necesarios para llegar a la meta final.
5. *Registrar usos múltiples de un atributo*.
6. *Usar el estado externo de los elementos tácticos*, especialmente las seudorreglas, para comprobar que se usan adecuadamente los nombres.
7. *Evitar duplicar inferencias.*
8. *Completar el MC*, comprobando que está toda la información necesaria.

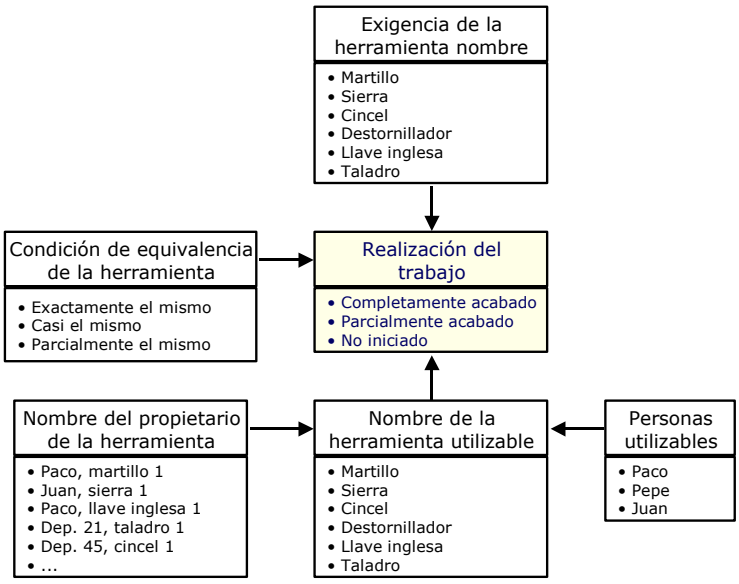


Figura 2.31 Fragmento de un Mapa de conocimientos en IDEAL

Un último aspecto de la conceptualización de IDEAL es que, a pesar de utilizarse representaciones informales o semi-formales para los distintos componentes de la conceptualización, los autores de la metodología han tenido especial cuidado en establecer criterios de comprobación (verificación) para que el Ingeniero del Conocimiento pueda comprobar la corrección interna de los modelos construidos.

2.5.2 CommonKADS

La metodología CommonKADS [Schreiber et al., 1999] [Schreiber et al., 1994] es el resultado de un esfuerzo continuado que comenzó en 1983 con el proyecto europeo KADS (ESPRIT P12, P314, P1098) [Wielinga et al., 1992] y siguió con los proyectos KADS-II (P5248), KACTUS (P8145) y TRACKS (P25087).

CommonKADS está basada en cuatro principios fundamentales que son los siguientes, en palabras de sus autores:

> *"La ingeniería del conocimiento no es una especie de 'extracción de la cabeza del experto', sino que consiste en construir modelos de aspectos diferentes del conocimiento humano.*
>
> *El principio del nivel de conocimiento: en ingeniería del conocimiento primero hay que concentrarse en la estructura conceptual del conocimiento, dejando los detalles de programación para más adelante.*
>
> *El conocimiento tiene una estructura interna estable que puede ser analizada distinguiendo roles y tipos específicos de conocimiento.*
>
> *Un proyecto de conocimiento debe ser gestionado aprendiendo de la experiencia siguiendo una espiral controlada" [Schreiber et al., 1999].*

Partiendo de estos principios, la aplicación de la metodología CommonKADS implica el desarrollo de seis modelos, que pueden agruparse en tres niveles (Figura 2.32), en función de las preguntas que se pretende contestar:

- *¿Por qué?* Se trata de averiguar por qué se desea construir un sistema basado en conocimientos y qué puede aportar. Lo fundamental es estudiar el contexto y entorno de la organización.
- *¿Qué?* En este nivel se pretende averiguar la naturaleza y estructura del conocimiento y comunicaciones implicadas en el sistema. Para ello se realiza una descripción conceptual del conocimiento utilizado en la tarea.
- *¿Cómo?* Se define cómo se implementa el conocimiento en un computador, es decir, se trabaja sobre los aspectos técnicos de la implementación.

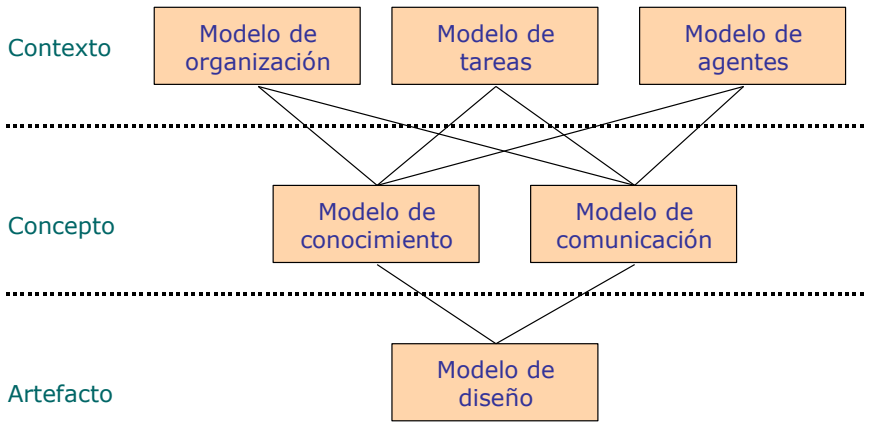


Figura 2.32 El conjunto de modelos de CommonKADS

Según sus autores, cada modelo se centra en un aspecto limitado del sistema, pero al unirlos todos se obtiene una visión completa. Seguidamente se describe brevemente cada uno de los modelos:

1. *Modelo de organización*. Este modelo permite realizar el análisis de las características principales de una organización, con el fin de descubrir problemas y oportunidades para el desarrollo de sistemas basados en el conocimiento. Se debe establecer la viabilidad de cada sistema candidato y evaluar el impacto que su desarrollo puede tener en la organización.
2. *Modelo de tareas*. Las tareas son los componentes relevantes de un proceso de negocio. El modelo de tareas analiza la estructura global de tareas, sus entradas y salidas, precondiciones y criterios de rendimiento, así como los recursos y competencias necesarios. El objetivo es identificar tareas con fuerte carga de conocimientos (*knowledge intensive*, en inglés) que puedan convertirse en sistemas basados en el conocimiento.
3. *Modelo de agentes*. Los agentes llevan a cabo las tareas. Pueden ser personas, sistemas de información o cualquier entidad capaz de realizar una tarea. Este modelo define las características de los agentes, en concreto sus competencias, autoridad para actuar y posibles restricciones. Adicionalmente enumera los enlaces de comunicación entre agentes para realizar una tarea.
4. *Modelo de conocimiento*. El objetivo de este modelo es explicar en detalle los tipos y estructura de los conocimientos utilizados en el desarrollo de una tarea. Proporciona una descripción comprensible para las personas e independiente de implementación del papel que desempeñan distintos componentes de conocimiento en la resolución de problemas. El modelo de conocimiento es, por tanto, una vehículo importante para comunicarse con los expertos y usuarios sobre las características de resolución del problema, tanto durante el desarrollo como durante el uso del sistema.
5. *Modelo de comunicación*. Se modela la comunicación establecida entre agentes implicados en la resolución de una tarea, también de forma independiente de la implementación.
6. *Modelo de diseño*. Los cinco modelos anteriores tomados en conjunto pueden verse como la especificación de requisitos del sistema. A partir de estos requisitos el modelo de diseño define los aspectos técnicos del sistema final: arquitectura, plataforma de implementación, módulos software, técnicas de representación, etc.

Dada la figura anterior y la breve descripción de cada modelo, queda claro que los modelos que interesan desde el punto de vista de esta Tesis son los dos modelos del nivel conceptual: el modelo de conocimiento y el modelo conceptual.

Sin embargo, antes de dar más detalles sobre estos modelos conviene indicar la situación de partida cuando comienza su desarrollo, para así comprender mejor el origen de algunos de sus elementos.

Los tres primeros modelos (organización, tareas, agentes) forman lo que los autores llaman el modelado del contexto organizativo, que se muestra con más detalle en la Figura 2.33. Durante su desarrollo se crean una serie de fichas u hojas de trabajo (del

inglés *worksheet*) que van recogiendo información cada vez más detallada sobre la organización, las tareas y los agentes. Estas hojas tienen un identificador formado por la abreviatura del modelo (OM para modelo de organización, TM para modelo de tareas, AM para modelo de agentes) y un número.

Desde el punto de vista del nivel conceptual las fichas más relevantes son TM-1 (análisis y descripción de tareas), TM-2 (análisis preliminar de los conocimientos necesarios) y AM-1 (descripción de los agentes que realizan tareas).

Una vez vista la situación de partida cuando se afronta el modelo conceptual de CommonKADS es el momento de pasar a describir brevemente sus dos modelos: el de conocimiento y el de comunicación.

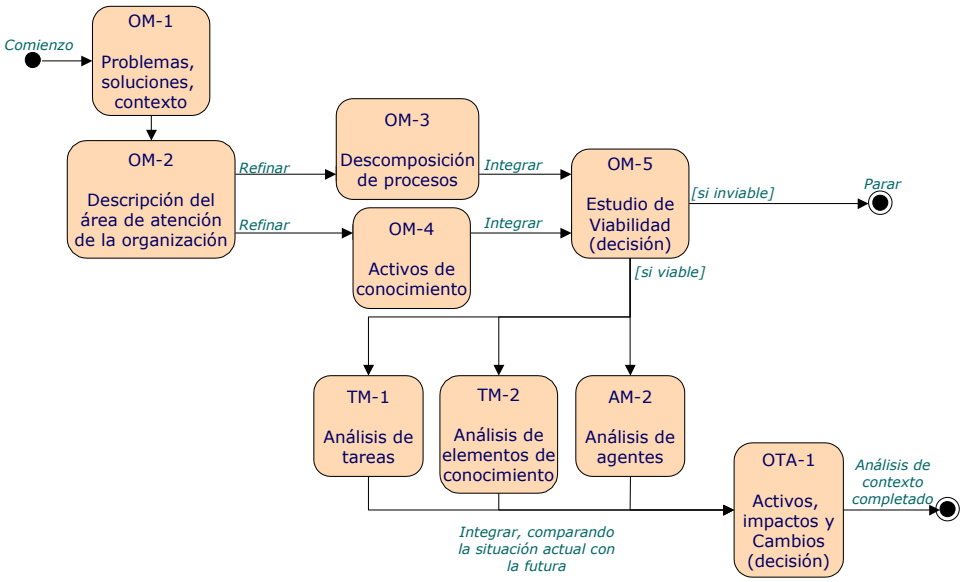


Figura 2.33 Proceso de desarrollo del modelo de la organización en CommonKADS

1. *Modelo de conocimiento*. Este modelo se divide en tres partes (Figura 2.34), en función del tipo de conocimiento considerado: conocimiento de dominio, conocimiento de inferencia y conocimiento de tarea (obsérvese la similitud con la división en fáctico, táctico y estratégico de IDEAL).

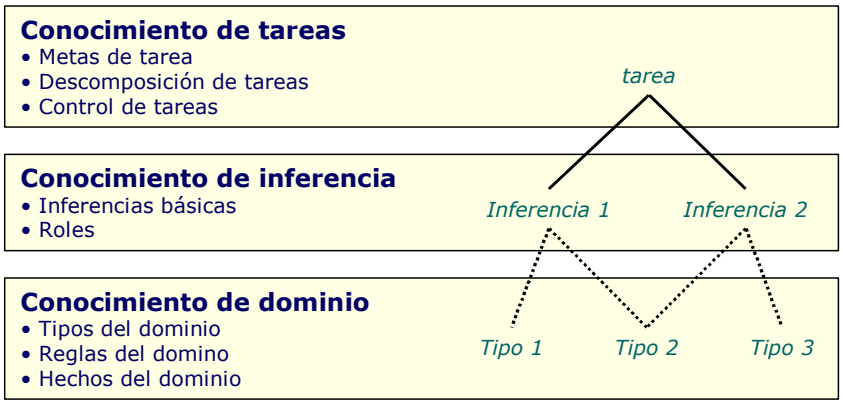


Figura 2.34 Categorías dentro del modelo de conocimiento en CommonKADS

a. *Conocimiento de Dominio*. Representa el conocimiento específico del dominio y los tipos de información que se utilizarán en la aplicación. Este nivel se divide, a su vez, en dos componentes:

   1. *Esquema de dominio*, que es una descripción esquemática de la estructura de información y conocimientos del dominio de la aplicación. Los elementos fundamentales de un esquema de dominio son los *conceptos* (conjunto de objetos que ocurren en el dominio de la aplicación y que comparten características comunes – los atributos), las *relaciones* (que se establecen entre conceptos y pueden tener también atributos) las generalizaciones entre conceptos y los tipos de reglas (que representan dependencias entre expresiones sobre elementos del dominio). Para representar todos estos elementos en CommonKADS se utilizan de forma conjunta un lenguaje de modelado, llamado CML [Schreiber et al., 1999] y diagramas con una notación basada en UML. La Figura 2.35 muestra los elementos fundamentales de la notación gráfica, mientras que la Figura 2.36 muestra un fragmento de un esquema de dominio con su correspondiente especificación en CML.

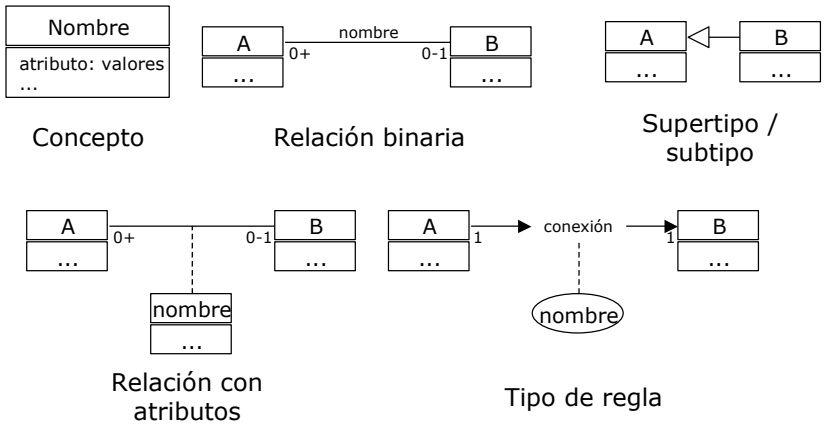


Figura 2.35 Notación gráfica básica para esquemas de dominio en CommonKADS

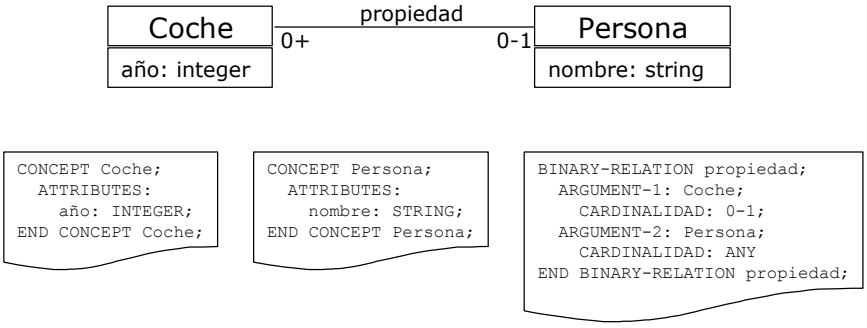


Figura 2.36 Ejemplo de esquema de dominio en notación gráfica y CML

   2. *Base de conocimientos*, que contiene instancias de los tipos especificados en un esquema de dominio. La necesidad de representar instancias en la fase de análisis es, según los autores de CommonKADS una característica que distingue a los sistemas basados en conocimientos. Las bases de

conocimiento no tienen representación gráfica y se especifican utilizando únicamente el lenguaje CML.

b. *Conocimiento de Inferencia*. Representa cómo se usa el conocimiento del dominio para razonar. En este nivel se distinguen tres elementos principales:

   1. *Inferencias*. Representan el nivel más bajo de descomposición funcional. Una inferencia lleva a cabo un paso elemental de razonamiento, utilizando el conocimiento de una base de conocimientos para obtener nueva información a partir del valor de sus entradas. Las inferencias no se detallan, sino que se utilizan como los elementos básicos para construir el resto del modelo de razonamiento.

   2. *Roles de conocimiento*. Se utilizan para representar de forma abstracta (es decir, independiente del dominio) la entrada / salida de las inferencias. Para ello se utilizan nombres que representan el papel de un elemento de conocimiento en una inferencia. Hay dos tipos de roles: dinámicos – que cambian en cada llamada a la inferencia y representan su entrada y resultados – y estáticos – que no suelen cambiar y representan el conocimiento manejado por la inferencia. Para poder utilizar inferencias en un dominio concreto es necesario asignar elementos concretos del dominio a los roles de las inferencias.

   3. *Funciones de transferencia*. Son funciones que representan la transferencia de información entre el agente que resuelve la tarea y el mundo exterior. Hay cuatro tipos de funciones de transferencia en CommonKADS: *obtener* (el agente pide información al exterior), *recibir* (el agente recibe información por iniciativa de otros), *presentar* (el agente envía información al exterior por iniciativa propia) y *proporcionar* (el agente envía información como respuesta a una solicitud del exterior).

   En CommonKADS existe un diagrama que permite representar la relación entre las inferencias, sus entradas, sus salidas y las funciones de transferencia. Este diagrama se denomina *estructura de inferencia* y se representa utilizando la notación mostrada en la Figura 2.37.

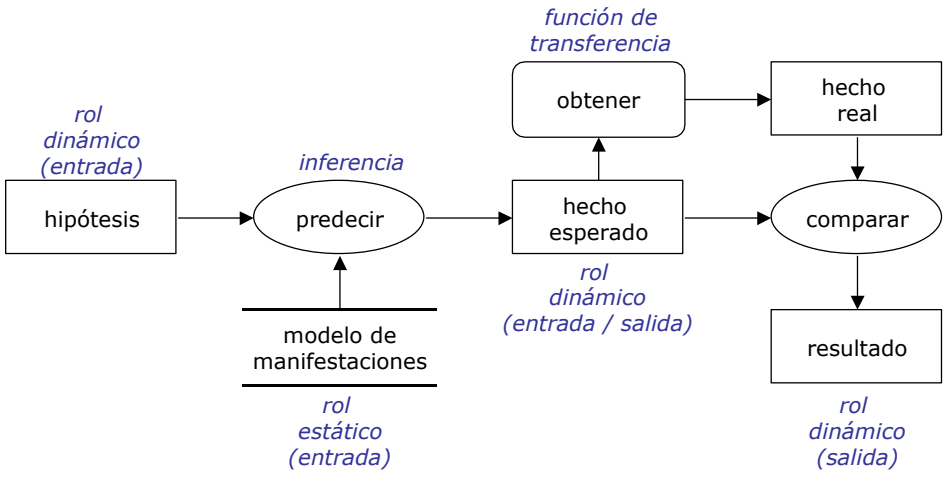


Figura 2.37 Notación de estructuras de inferencia en CommonKADS

c. *Conocimiento de Tareas*. Representa las metas y estrategias utilizadas para llegar a esas metas. Para ello utiliza dos elementos fundamentales:

1. *Tareas*: definen funciones de razonamiento y se especifican en el lenguaje CML indicando su nombre, metas, entradas, salidas y descripción.

2. *Métodos de tarea*: definen cómo se lleva a cabo una tarea mediante su descomposición en pasos más elementales (que pueden ser tareas, inferencias o funciones de transferencia) y la definición del control ejercido sobre esos elementos. La descomposición se puede representar gráficamente mediante un diagrama de descomposición de tareas (Figura 2.38), mientras que el control sólo aparece en la especificación completa del método usando el lenguaje CML.

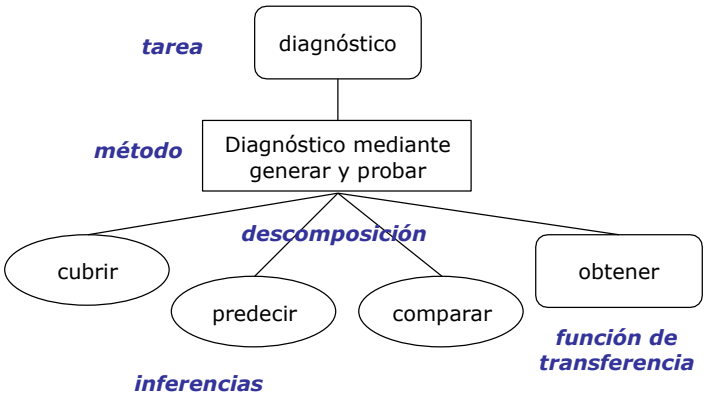


Figura 2.38 Ejemplo de diagrama de descomposición de tareas en CommonKADS

2. *Modelo de comunicación*. Este modelo permite especificar la comunicación entre agentes cuando resuelven tareas, para lo cual se definen tres elementos principales:

a. *Plan de Comunicación*, que pretende dar una visión general de la comunicación establecida entre varios agentes que colaboran para realizar una tarea global (para ello cada uno de los agentes realiza su propia tarea). Este plan de comunicación tiene dos componentes fundamentales:

1. *Diagrama de diálogo*, que representa en un único dibujo las tareas que realiza cada agente y las transacciones de información establecidas entre los mismos (Figura 2.39).

2. *Control sobre las transacciones*, que especifica cómo se controla la comunicación entre los agentes y qué debe hacerse en cada momento. Para ello se utiliza pseudocódigo o diagramas de transición de estados.

b. *Transacción*, que representa un intercambio de información entre tareas de los agentes. Debe especificarse el nombre de la transacción, la información intercambiada, los agentes que emiten y reciben la información y restricciones sobre la transacción.

c. *Especificación de Intercambio de Información*, mediante la que se detallan todos los aspectos internos de la transacción, fundamentalmente los que tienen que ver con los mensajes involucrados (formato y número de mensajes intercambiados, más el control del protocolo si fuera necesario).

Para concluir el resumen de CommonKADS es de destacar la gran importancia que dan sus autores a la reutilización en el modelo de conocimiento. Para ello proponen reutilizar no sólo fragmentos de esquemas de dominio, sino fundamentalmente catálogos de tareas, métodos e inferencias de propósito general. Su objetivo es que el Ingeniero del Conocimiento, una vez comprendido el problema, pueda elegir elementos de esos catálogos para luego ajustarlos a sus necesidades y construir el modelo completo.

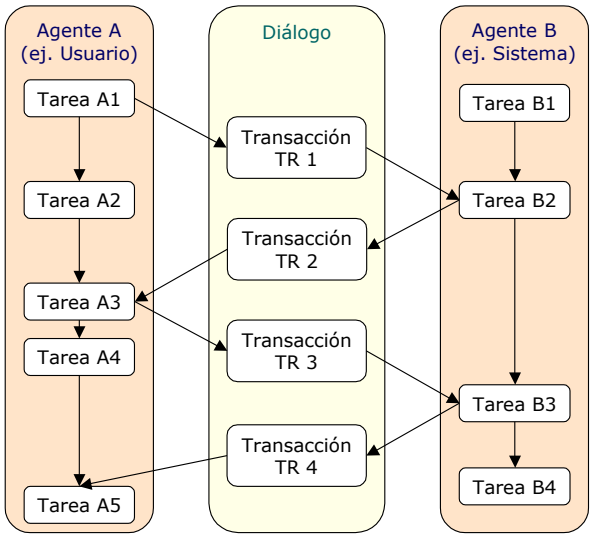


Figura 2.39 Ejemplo de diagrama de diálogo en CommonKADS

2.5.3 Otras Metodologías de Ingeniería del Conocimiento

Además de las dos metodologías de Ingeniería del Conocimiento descritas anteriormente, pueden destacarse dos más:

1. *MIKE*, del inglés *Model-based and Incremental Knowledge Engineering* (es decir, Ingeniería del Conocimiento Basada en Modelos e Incremental) es un método de desarrollo que propone integrar técnicas de especificación semiformales y formales, junto con el prototipado para construir sistemas de forma incremental [Angele et al., 1998] [Angele et al., 1996] [Angele 1996] [Fensel et al., 1996] [Landes et al., 1995] [Neubert, 1993].
2. *PROTÉGÉ-II* plantea el desarrollo de Sistemas Basados en el Conocimiento a partir de la reutilización de métodos de resolución de problemas y de ontologías. Además pone un gran énfasis en la generación de herramientas de adquisición de conocimientos ajustadas a las necesidades de cada caso [Eriksson et al., 1995] [Puerta et al., 1992] [Tu et al., 1995].

2.6 Análisis Basado en el Problema

En los últimos años han surgido una serie de técnicas y métodos con mayor énfasis en el problema, que tratan de estar alejados de las técnicas de diseño e implementación. Para ello han utilizado tres estrategias, según [Andrade, 2002]:

1. Ampliación de los conceptos manejados por el método. Algunos ejemplos tradicionales de esta estrategia son la definición de control dentro de los DFD [Hatley et al., 1987] o la incorporación de la agregación y la especialización a la técnica de los diagramas entidad relación [Lazimy, 1989].
2. La definición de un metamodelo mediante el que se definen de forma explícita y no ambigua los conceptos utilizados por el método. Puede verse el metamodelo como una estructura genérica de modelo conceptual, sobre la que se construyen los modelos particulares. Dentro de esta estrategia pueden mencionarse EM [Bubenko et al., 1995] [Kirikova, 2000] [Kirikova et al., 1994a] y KAOS [Dardenne et al., 1993] [van Lamsweerde et al., 1991].
3. La definición de ontologías adaptables, de forma que la técnica permite definir nuevos elementos de modelado ajustados a las características del problema bajo estudio. Un ejemplo de esta filosofía es TELOS [Mylopoulos et al., 1990] [Nissen et al., 1996] [Plexousakis, 1993], que utiliza clases para modelar el problema y permite definir metaclases, que definen las características de las clases.

Un esfuerzo muy reciente en esta línea es el trabajo presentado por Andrade en su tesis doctoral [Andrade, 2002], cuyo objetivo es definir un marco de trabajo para el análisis del problema. Dado su interés se van a describir brevemente sus características.

2.6.1 Marco Metodológico para el Modelado Conceptual

La Tesis Doctoral de Andrade define un marco metodológico para realizar el modelado conceptual de problemas, tanto fuera como dentro de la Ingeniería del Software. Según su autor, este marco metodológico se caracteriza por [Andrade, 2002]:

- Ofrecer una sensibilidad al problema, abandonando los términos propios del desarrollo de software.
- Guiar la construcción del modelo conceptual, enumerando tanto las actividades que deben llevarse a cabo (enfoque prescriptivo) y la forma de hacerlo (descriptivo).
- Proporcionar actividades de verificación y validación que permiten probar que el modelo conceptual es correcto y completo antes de seguir con las siguientes actividades.
- Considerar los diferentes puntos de vista que coexisten entre los individuos afectados por el futuro sistema, así como las discrepancias que pudieran surgir.

Según este marco metodológico, el modelo conceptual puede dividirse en dos niveles principales (Figura 2.40):

1. *Estático*: conforma la información estructural del dominio en forma de conceptos, propiedades, relaciones y restricciones. Este nivel también se denomina *operativo*.
2. *Dinámico*: conforma el comportamiento que se produce en el dominio. Este nivel puede, a su vez, dividirse en otros dos:
   a. *Estratégico*: especifica el control del sistema, es decir, qué hacer, cuándo y en qué secuencia.

b. *Táctico*: Especifica cómo y cuándo se incorpora nueva información operativa acerca del problema.

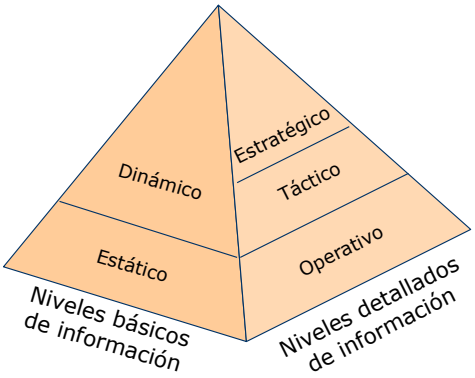


Figura 2.40 Niveles de información en el modelado conceptual de [Andrade, 2002]

En cuanto al proceso de conceptualización, el autor sitúa la construcción del modelo conceptual dentro de un ciclo que integra la adquisición de información con la conceptualización del problema y lo divide en tres grandes pasos (Figura 2.41):

1. *Análisis*: se identifica la información relevante de toda la adquirida, clasificándola en conceptos, relaciones, propiedades, pasos, etc.
2. *Síntesis*: se organiza y representa la información para formar el modelo conceptual.
3. *Prueba*: se evalúa el modelo, comprobando que es correcto y completo (con ayuda de los usuarios o expertos).

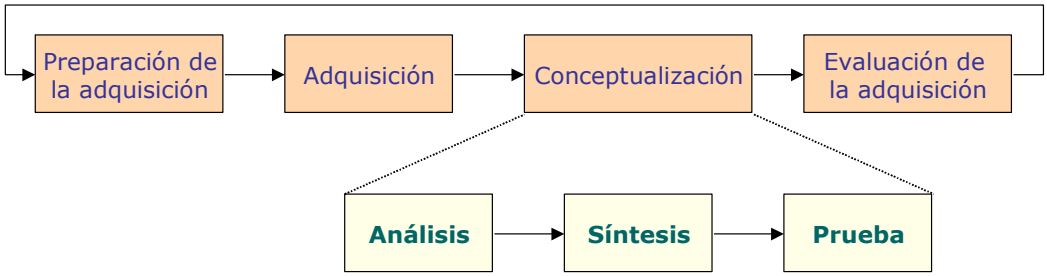


Figura 2.41 Actividades principales de la conceptualización según [Andrade, 2002]

Para las actividades de análisis y síntesis, Andrade realiza una descomposición en pasos elementales tomando como base los tres niveles detallados de información, tal y como se recoge en la Tabla 2.14.

Tabla 2.14 Detalle de actividades de conceptualización según [Andrade, 2002]

Nivel	Pasos de análisis	Pasos de síntesis
Paso previo	Actualizar catálogo de términos	
Operativo	Identificar conceptos Identificar relaciones Identificar propiedades	Conformar catálogo de conceptos Conformar catálogo de relaciones Conformar catálogo de propiedades
Táctico	Identificar cálculos Identificar inferencias	Conformar catálogo de cálculos Conformar catálogo de inferencias
Estratégico	Identificar pasos Identificar secuencia de ejecución	Obtener descomposición de pasos Obtener secuencia de ejecución de pasos Obtener catálogo de pasos de último nivel

La tercera actividad, la prueba, queda especificada mediante unas listas de comprobación que deben aplicarse a los tres submodelos de la conceptualización: operativo, táctico y estratégico.

La presentación de los detalles del proceso de conceptualización de la Tesis de Andrade permite identificar una serie de catálogos, que permiten especificar todos los elementos de un modelo conceptual. Todos estos catálogos se describen con brevedad a continuación:

- *Catálogo de términos*. Este catálogo es un apoyo a la primera labor dentro de la conceptualización: familiarizarse con el problema que se desea modelar. Recoge los términos propios del dominio del problema junto con sus definiciones (Tabla 2.15).

Tabla 2.15 Descripción del catálogo de términos en [Andrade, 2002]

Descriptor	Descripción
Término	Nombre usado en el dominio.
Definición	Definición del término.
Observaciones	Cualquier aspecto que sea interesante reflejar como abreviaturas, sinónimos, etc.
Referencias	Indica de dónde se obtuvo el término.

- *Submodelo operativo*. Dentro de este submodelo se consideran catálogos para los conceptos, las relaciones entre conceptos y las propiedades:
  - *Catálogo de conceptos*. En este catálogo se recogen todos los conceptos identificados en el problema, según el formato de la Tabla 2.16.

Tabla 2.16 Descripción del catálogo de conceptos en [Andrade, 2002]

Descriptor	Descripción
Concepto	Nombre del concepto
Definición	Definición o función que desempeña en el problema.
Propiedades	Lista de propiedades del concepto.
Relaciones	Nombre de las relaciones que establece este concepto con otros, incluyendo generalizaciones / especializaciones.
Composición	Si es un concepto compuesto por otros, se indican los conceptos que lo componen.
Observaciones	Sinónimos, abreviaturas, etc.
Referencias	Indica de dónde se obtuvo el concepto.

  - *Catálogo de relaciones entre conceptos*, que recoge todos los detalles de las relaciones establecidas entre conceptos del modelo, siguiendo el formato mostrado en la Tabla 2.17.

Tabla 2.17 Descripción del catálogo de relaciones en [Andrade, 2002]

Descriptor	Descripción
Relación	Nombre de la relación.
Definición	Definición o función que desempeña en el problema.
Conceptos	Enumeración de los conceptos que participan en la relación.
Propiedades	Lista de propiedades de la relación.
Restricciones	Restricciones que afecten a la relación.
Observaciones	Sinónimos, abreviaturas, etc.
Referencias	Indica de dónde se obtuvo la relación.

- *Catálogo de relaciones entre conceptos*, que recoge todos los detalles de las propiedades identificadas en conceptos y relaciones (véase la Tabla 2.18).

Tabla 2.18 Descripción del catálogo de propiedades en [Andrade, 2002]

Descriptor	Descripción
Propiedad	Nombre de la propiedad.
Definición	Definición o función que desempeña en el problema.
Elemento	Concepto o relación al que describe
Tipo	Tipo de valor de la propiedad y, si es necesario, unidades y precisión requerida.
Rango	Posibles valores junto con sus restricciones
Restricciones	Restricciones que afecten a la propiedad.
Observaciones	Sinónimos, abreviaturas, etc.
Referencias	Indica de dónde se obtuvo la propiedad.

- *Submodelo táctico*. Dentro de este submodelo se consideran catálogos representar cálculos e inferencias:
  - *Catálogo de cálculos*. En este catálogo se recogen todos los cálculos identificados en el problema, es decir, fórmulas o algoritmos aplicados para obtener nuevos valores o ejemplares de relaciones (Tabla 2.19).

Tabla 2.19 Descripción del catálogo de cálculos en [Andrade, 2002]

Descriptor	Descripción
Cálculo	Nombre del cálculo.
Descripción	Descripción o función del cálculo en el problema.
Elementos básicos	Elementos (conceptos, relaciones, propiedades) utilizados en el cálculo para obtener la conclusión (resultado).
Elementos conclusión	Elementos cuyos valores se obtienen a través del cálculo.
Definición	Algoritmo o fórmula.
Precisión	Precisión exigida en el resultado.
Restricciones	Restricciones sobre cuándo es posible realizar el cálculo.
Referencias	Indica de dónde se obtuvo el cálculo.

  - *Catálogo de inferencias*. En este catálogo se recogen las inferencias identificadas en el problema (Tabla 2.20).

Tabla 2.20 Descripción del catálogo de inferencias en [Andrade, 2002]

Descriptor	Descripción
Inferencia	Nombre de la inferencia.
Descripción	Descripción de la inferencia en términos del problema bajo estudio.
Elementos básicos	Elementos empleados en la condición.
Elementos conclusión	Elementos inferidos en la conclusión.
Formulación	Formulación de la regla que permite realizar la inferencia. Se admite también utilizar árboles de decisión, tablas de decisión o redes de inferencia.
Referencias	Indica de dónde se obtuvo la inferencia.

- *Submodelo estratégico*. Dentro de este submodelo se consideran la descomposición de pasos, la secuencia de ejecución de pasos y un catálogo de pasos de último nivel:

- *Descomposición en pasos*. Es una representación gráfica en forma de árbol o grafo de la descomposición de los pasos que deben realizarse para resolver el problema planteado.
- *Secuencia de ejecución de pasos*. Consiste en la representación de la secuencia de ejecución de pasos que permite resolver el problema. El autor recomienda utilizar organigramas tradicionales, redes de Petri o metarreglas, según el tipo de control existente.
- *Catálogo de pasos de último nivel*. En este catálogo se recogen todos los pasos que no se descomponen en otros de menor nivel (véase la Tabla 2.21).

Tabla 2.21 Descripción del catálogo de pasos de último nivel en [Andrade, 2002]

Descriptor	Descripción
Paso	Nombre del paso de último nivel.
Descripción	Descripción de las funciones realizadas en el paso.
Criterios de inicio	Condiciones que se deben cumplir para poder ejecutar este paso.
Criterios de fin	Condiciones que se deben cumplir para establecer la finalización del paso.
Necesidades de inicio	Elementos necesarios para ejecutar el paso.
Resultados producidos	Elementos obtenidos como consecuencia de ejecutar el paso
Información táctica	Información táctica (cálculos, inferencias) que permite obtener los resultados.
Referencias	Indica de dónde se obtuvo el paso.

- *Integración de submodelos*. Una vez completados todos los submodelos, el método propuesto por Andrade recomienda realizar dos diagramas que integran estos modelos para su mejor validación: los mapas de proceso y los mapas de información, que son muy similares a los ya descritos en la metodología IDEAL.

Además de toda la información recogida en los catálogos, en el método de Andrade se propone una serie de notaciones gráficas para representar de forma visual los elementos principales de los modelos.

La más destacada es la notación para el *diagrama del submodelo operativo*, que permite mostrar gráficamente los conceptos, relaciones, propiedades y restricciones. Los elementos principales de esta notación se recogen en la Figura 2.42.

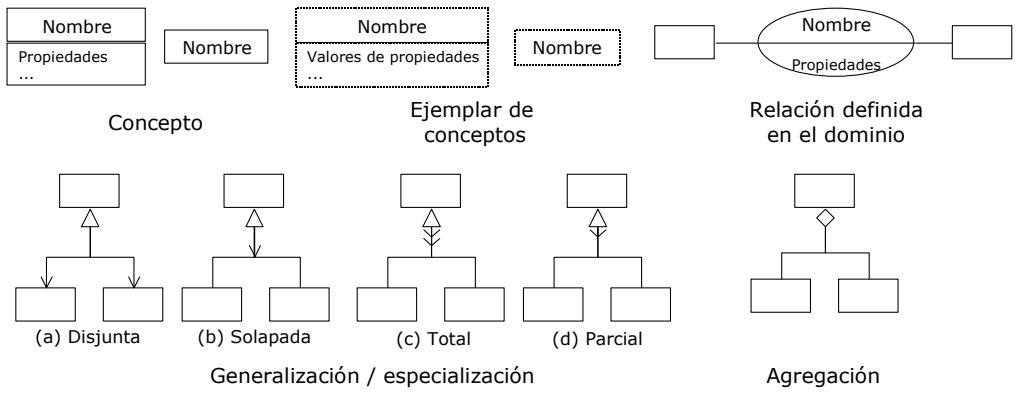


Figura 2.42 Notación gráfica del submodelo operativo de [Andrade 2002]

Después de toda esta descripción puede comprobarse que el método de modelado conceptual de Andrade es una ampliación de la propuesta realizada en IDEAL, que conserva su misma filosofía y aporta un mayor nivel de detalle en lo que respecta a los catálogos, notación gráfica y proceso de modelado.

2.6.2 Otros Métodos basados en el Problema

Además de las propuestas ya enumeradas en la introducción a este apartado (EM, KAOS y TELOS), pueden nombrarse algunos métodos más que pretenden ser genéricos e independientes de la implementación: OORAM [Reenskaug, 1992], i* [Yu et al., 1994], así como trabajos más recientes que representan la evolución de métodos ya existentes [Butler et al., 2000], [Gómez et al., 2000], [Kirikova, 2000], [Mineau et al., 2000].

2.7 DISCUSIÓN DEL ESTADO DE LA CUESTIÓN

Una vez descritos los métodos de análisis del problema más habituales en el desarrollo de software es el momento de comparar sus características e identificar los problemas pendientes en el estado de la cuestión.

Antes de nada debe destacarse que, aunque puedan tener inconvenientes, todos los métodos anteriores han sido utilizados con éxito en determinados tipos de problemas y bajo ciertas circunstancias. En otras palabras, todos estos métodos son y seguirán siendo útiles en alguna de las situaciones a las que deba enfrentarse un desarrollador de aplicaciones a lo largo de su carrera profesional.

La primera parte de este apartado va a estar dedicada a definir una serie de criterios que puedan utilizarse para comparar y clasificar los métodos de análisis.

2.7.1 Criterios de comparación

Existe un gran número de autores que han identificado las características que debería cumplir cualquier método de modelado conceptual para poder ser utilizado en la práctica. Así, por ejemplo, Loucopoulos y Karaostas enumeran las siguientes [Loucopoulos et a., 1995]:

1. *Abstracción*. Los elementos que conforman un modelo de análisis deben ser abstractos para evitar el tener que manejar en el modelo todos los detalles irrelevantes del mundo real.
2. *No ambigüedad*. La información recogida en el modelo conceptual debe estar expresada en términos no ambiguos, de forma que no admita distintas interpretaciones y pueda utilizarse como medio de comunicación efectivo entre los clientes, usuarios y desarrolladores.
3. *Facilidad para la comprensión*. Además de no ser ambigua, la información contenida en el modelo debe ser fácil de comprender tanto para los clientes y usuarios (que generalmente no son informáticos) como para los desarrolladores (que en muchos casos no conocen el dominio del problema). Esta característica es fundamental para poder validar el modelo, es decir, para poder comprobar si el modelo se corresponde con la realidad.

4. *Facilidad para la verificación*. El modelo debe estar especificado con el suficiente grado de detalle como para poder comprobar que es correcto, es decir, que ha sido construido de acuerdo con las reglas sintácticas impuestas por la técnica elegida. Además el formato de representación debería permitir que esta verificación fuera realizada de forma automática por un computador.
5. *Independencia respecto a técnicas de desarrollo*, de forma que pueda elegirse el paradigma o paradigmas de diseño más adecuados para desarrollar el sistema. Esta elección debe depender únicamente de las características del problema y no venir impuestas por la técnica de modelado utilizada para estudiar el problema.

En un sentido parecido se expresa Blum, que defiende la necesidad de técnicas de análisis que sean formales (y a la vez conceptuales), estén orientadas al problema, y tengan las siguientes propiedades [Blum, 1996]:

1. *Formalización de los conceptos*: los formalismos utilizados deben ser capaces de modelar los conceptos de mayor interés desde el principio del proceso de desarrollo del software, con el fin de aprovechar la capacidad de la notación para comprobar la corrección.
2. *Expresividad de los conceptos*: la notación debe ser efectiva para razonar sobre el tipo de problema que se está estudiando. Esto implica que la notación formal debe comunicar los conceptos del dominio de la aplicación de forma clara a todas las personas implicadas.
3. *Exhaustividad de los conceptos*: estos formalismos deben ser exhaustivos para poder establecer la corrección de cualquier aspecto del comportamiento del sistema a cualquier nivel de detalle.
4. *Automatización de los conceptos*: el entorno debe *manejar la especificación formal de forma automática* para que la especificación de bajo nivel pueda ser verificada, el comportamiento del sistema pueda ser inspeccionado y validado y la documentación pueda generarse con el mínimo de intervención humana posible.

Por otro lado, y aunque su trabajo está centrado en el tema de la Gestión de Conocimientos, Paradela define una serie de principios, de los cuales los tres primeros pueden aplicarse al modelado conceptual [Paradela, 2001]:

1. *Principio de la Sensibilidad al Problema* (enunciado en [Jackson, 1995a]): una metodología para resolver problemas debe expresarse, tanto como sea posible, en términos de los problemas que pretende ayudar a resolver.
2. *Principio del "Antiprocustismo"* o de flexibilidad. El nombre de este principio hace referencia al mito de Procusto, un hostelero de la antigua Grecia que tenía dos camas, una corta y otro larga. Si el cliente era alto, le ofrecía la cama corta y le cortaba lo que sobraba se sus miembros inferiores. Si el cliente era bajo, le ofrecía la cama larga y le estiraba los miembros hasta que el huésped expiraba. [Seeman, 1958]). Según este principio una metodología debe tener la flexibilidad suficiente como para que pueda ajustarse a las necesidades de cada caso.

3. *Principio de la Materia Prima*, o de Skater, según el cual debe elegirse un lenguaje apropiado para modelar el problema que además no influya sobre la elección de técnicas para resolverlo en un computador.

A partir de todas estas propiedades deseadas en las técnicas y notaciones de análisis del problema, pueden definirse los **criterios que se utilizarán para comparar los métodos** presentados en los apartados anteriores:

1. **Base formal**. La existencia de una base formal es imprescindible para evitar la ambigüedad y para poder verificar la corrección de los modelos generados. Según este criterio los métodos podrán clasificarse en tres categorías:
   a. *Métodos con base formal*, cuando todos los elementos de modelado están definidos formalmente y de forma no ambigua.
   b. *Métodos no formales*, cuando los elementos de modelado están definidos de manera informal.
   c. *Métodos semiformales*, cuando algunos elementos tienen base formal y otros no.
2. **Existencia de notación gráfica** para representar los modelos. Una notación gráfica es fundamental para que los elementos relevantes de los modelos puedan ser comprendidos y validados fácilmente por parte de los clientes y usuarios. Según este criterio habrá tres tipos de métodos:
   a. *Métodos sin notación gráfica*, que carezcan por completo de la posibilidad de realizar diagramas de partes del modelo.
   b. *Métodos exclusivamente gráficos*, que usen los diagramas como única forma de representar los modelos.
   c. *Métodos mixtos*, que incorporen una notación gráfica para representar los elementos importantes de los modelos y una notación textual que especifique todos los detalles.
3. **Tipo de notación textual** utilizada para especificar los detalles de los modelos. El tipo de notación tendrá gran influencia en la capacidad expresiva de los modelos construidos, lo cual afecta a la facilidad de comprensión y a la validación de los modelos por parte de los clientes y usuarios. Este criterio permite clasificar los métodos en las siguientes categorías:
   a. *Notación informal*: usan lenguaje natural para describir los detalles de los modelos, lo cual puede hacer que esos modelos sean ambiguos.
   b. *Notación tabular*: los elementos se especifican rellenando tablas con todos sus detalles.
   c. *Notación matemática*: usan notaciones basadas en el lenguaje matemático para especificar los modelos.
   d. *Notación formal no matemática*: se usa un lenguaje de especificación no ambiguo, pero sin base matemática, que suele tener cierto parecido con lenguajes de programación de muy alto nivel.

4. **Independencia de paradigmas de diseño**. Este criterio se refiere a si la técnica utilizada para analizar el problema utiliza únicamente términos del problema o si, por el contrario, impone cierto sesgo hacia términos de diseño. Según este criterio los métodos se pueden clasificar en:

   a. *Métodos dependientes de diseño*: utilizan paradigmas de diseño como filosofía a la hora de estudiar el problema, lo cual obliga a mantener esa filosofía cuando avanza el desarrollo, aunque no sea la más adecuada una vez se conoce el problema.

   b. *Métodos dependientes de arquitectura*: estos métodos, aunque en principio no utilizan terminología de diseño en el análisis, siguen dependiendo de una arquitectura típica de la solución que influye en el análisis realizado.

   c. *Métodos independientes de diseño*: utilizan términos que no tienen reflejo directo en el diseño, por lo que permiten centrarse en las características del problema.

5. **Adecuación a tipos de problemas**. Los métodos de modelado conceptual deberían permitir abordar el estudio de cualquier tipo de problemas. Para definir este criterio es necesario hacer una definición previa de los tipos de problemas que puede encontrarse un desarrollador de programas.

   Existen muchas clasificaciones diferentes de tipos de problemas. Así, por ejemplo, Blum recoge en su libro dos categorías de problemas [Blum, 1996]: cerrados (o bien estructurados), cuyos requisitos pueden conocerse antes de resolverse; y abiertos (o mal estructurados), cuyos requisitos son desconocidos y sólo se van comprendiendo según se están resolviendo. Siguiendo una línea similar, pero con más detalle, Ares y Pazos identifican tres tipos de problemas [Ares et al., 1998]: algorítmicos, de soporte a la decisión y basados en conocimientos (que son los afrontados en la actualidad en el desarrollo de software completo). Las características de estos problemas se recogen de forma resumida en la Tabla 2.22.

   Tomando como partida esta clasificación de tipos de problemas, este criterio permite definir tres categorías de métodos:

   a. *Métodos para problemas algorítmicos*: estos métodos son adecuados para resolver problemas con requisitos cerrados y solución algorítmica.

   b. *Métodos para problemas de soporte a la decisión*: estos métodos permiten gestionar de forma más o menos adecuada la falta de requisitos completos y el tratamiento de certezas probabilísticas.

   c. *Métodos para problemas basados en el conocimiento*: estos métodos han sido diseñados para abordar requisitos subjetivos, inconsistencia, incompletitud y falta de precisión. Además, los métodos de este tipo deben presentar dos características importantes, según [Schreiber et al., 1999]:

      1. Deben permitir *representar ejemplares* (o instancias) de los elementos de modelado estático, ya que esos ejemplares permiten representar conocimiento concreto que posee el sistema sobre el dominio del problema.

Tabla 2.22 Tipos de problemas según [Ares et al., 1998]

Problema Algorítmico	
Estructura:	**Estructurado**: ▪ Pueden describirse mediante valores numéricos. ▪ La solución viene dada por una función objetiva. ▪ Resuelto por un algoritmo en tiempo polinómico. ▪ Ejemplo: nómina
Tipo de decisión:	**Operativa**: ▪ Decisión reversible y a corto plazo (menos de 6 meses) que afecta a una pequeña parte de la organización. ▪ Resultados dados por objetivos cualitativos. ▪ Datos precisos y detallados de fuentes internas. ▪ Bajo riesgo.
Tipo de requisitos:	**Objetivos**: absolutamente independientes de los deseos, juicios o creencias de las personas que deben resolver el problema.
Contexto:	**Independiente** de contexto (ajedrez).
Certeza:	**Absolutamente cierta**: situación completamente determinista y determinada.
Tipo de información:	**Datos**: sintácticos, no estructurados, independientes de contexto (100º C).
Problema de soporte a la decisión	
Estructura:	**Semiestructurados**: ▪ Uno de los componentes es desconocido o está pobremente definido mientras que el resto están bien definidos. ▪ Ejemplos: problemas de investigación operativa.
Tipo de decisión:	**Táctica**: ▪ Decisión semirreversible y a medio plazo (de 6 a 24 meses) que afecta a una parte considerable de la organización. ▪ Resultados dados por objetivos cualitativos. ▪ Datos precisos y detallados de fuentes internas. ▪ Bajo riesgo.
Tipo de requisitos:	**Híbridos**: una parte significante del problema tiene al menos un componente subjetivo.
Contexto:	**Sensibles al contexto** (diagnóstico médico).
Certeza:	**Probabilística**: información probabilística con factores de certeza.
Tipo de información:	**Noticias**: semánticos, estructurados, dependientes de contexto (el agua hierve a 100º C).
Problema basado en conocimientos	
Estructura:	**Mal estructurado**: ▪ Casi nada es conocido o está bien definido. ▪ No se resuelve por un algoritmo en tiempo polinómico. ▪ Ejemplo: el computador simula la mente de un experto.
Tipo de decisión:	**Estratégica**: ▪ Decisión irreversible y a largo plazo (más de 24 meses) que afecta a la mayor parte de la organización. ▪ Resultados desconocidos. ▪ Objetivos filosóficos. ▪ Datos vagos provenientes de fuentes internas y externas. ▪ Alto riesgo.
Tipo de requisitos:	**Subjetivos**: muy influenciados por los deseos, juicios o creencias de las personas que resuelven el problema.
Contexto:	**Dependientes de contexto** (lenguaje natural).
Certeza:	**Incierta**. como consecuencia de inconsistencia, incompletitud o falta de precisión de la información procesada.
Tipo de información:	**Conocimiento**: pragmático, abstracto y tácito en cuanto a la conexión entre objetos (el agua se esteriliza a 100º C).

2. Deben permitir *representar el control interno* de las tareas, de forma que quede claramente reflejado el proceso de resolución del problema por parte

del experto. Los enfoques puramente declarativos no son suficientes para modelar ese comportamiento.

Un factor importante que hay que tener en cuenta cuando se evalúe este criterio es que cada método se ajusta mejor al tipo de problemas para el que fue diseñado y que los métodos diseñados para problemas complejos suelen ser poco eficientes al tratar problemas más sencillos.

Una vez definidos los cinco criterios clasificatorios y teniendo en cuenta las características que deberían cumplir los modelos conceptuales según los autores mencionados anteriormente, *los métodos más adecuados serían los siguientes*:

- Métodos con *base formal*, lo que facilita la verificación y tratamiento automático.
- Métodos con *mezcla de notación textual y gráfica*, en los que los gráficos sirvan de complemento a la especificación textual completa. Así se facilitará la comprensión del modelo y su validación.
- Métodos con *notación formal no matemática*, de forma que se facilite su validación y comprensión por personas sin conocimientos matemáticos pero al mismo tiempo sea lo suficientemente rigurosa para facilitar su verificación.
- Métodos *independientes de paradigmas de diseño*, para que su uso no imponga la utilización de técnicas de diseño antes de que se comprenda realmente el problema.
- Métodos para *problemas basados en el conocimiento*, que son los que realmente presentan un desafío para los desarrolladores de software.

Seguidamente se van a clasificar todos los métodos expuestos en este estado de la cuestión, con el fin de identificar los métodos que cumplan todas las características.

2.7.2 Clasificación de los métodos

Con el fin de que este apartado no sea demasiado extenso, se van a realizar las siguientes agrupaciones de métodos:

1. *Métodos estructurados*: no se van a considerar métodos individuales ya que todos tienen características muy similares.
2. *Métodos orientados a objetos*: según ha podido verse en su descripción, el Proceso Unificado y OPEN son muy similares ya que plantean el análisis en dos niveles: las tareas que deben realizarse y una primera versión de un modelo orientado a objetos. Lo que procede por tanto es dividirlo en dos partes, en función del tipo de análisis:
   a. *Análisis de tareas*, que se corresponde con los casos de uso en el Proceso Unificado y el modelo de tareas (TOM) en OPEN.
   b. *Análisis de clases*, que se corresponde con el modelo de análisis en el Proceso Unificado y el modelo de objetos del negocio (BOM) en OPEN.
3. *Métodos orientados a agentes*: todos son muy similares en cuanto a forma de representar el problema y sus diferencias son más accidentales que esenciales. Por tanto se considerarán como un único grupo.

4. *Métodos basados en el conocimiento*: en este grupo se estudiarán por separado las dos metodologías presentadas:
   a. *IDEAL*.
   b. *CommonKADS*.
5. *Métodos formales*: se dividirán en dos grupos.
   a. *Métodos formales matemáticos*, donde estarán Z y VDM.
   b. *Métodos formales no matemáticos*, que usan una notación similar a lenguaje de programación. A este grupo pertenece PVS.
6. *Métodos centrados en el problema*. Dentro de este grupo se estudiará únicamente el propuesto por Andrade.

2.7.2.1 Existencia de base formal

Respecto a la existencia de una base formal, los métodos pueden clasificarse de la siguiente forma:

1. *Métodos estructurados*: que parte de sus elementos de análisis están completamente especificados (como los diagramas de transición de estados), pero otras están definidas de manera informal. Además, estos métodos presentan dificultades de integración entre sus diferentes vistas. Por lo tanto se considerarán como **semiformales**.
2. *Métodos orientados a objetos*:
   a. *Análisis de tareas*: el análisis externo del comportamiento (los casos de uso y TOM) es muy informal, con la mayoría de las descripciones hechas en lenguaje natural. Se considerarán **no formales**.
   b. *Análisis de clases*: tiene una base formal pero los elementos que componen el análisis de clases no están completamente especificados. Por lo tanto se considera **semiformales**.
3. *Métodos orientados a agentes*: prácticamente ninguno de los métodos actuales basados en agentes utiliza una base formal completa. **Semiformales**.
4. *Métodos basados en el conocimiento*:
   a. *IDEAL*: sus autores consideran que en el modelado conceptual se usan representaciones intermedias, que no tienen restricciones formales, lo que permite centrarse en el problema. Por tanto se considera este método como **no formal**.
   b. *CommonKADS*: los elementos utilizados en esta metodología tienen un componente más formal, aunque se permiten algunas descripciones en lenguaje natural. **Semiformal**.
5. *Métodos formales*: ambos grupos (matemáticos y no matemáticos) son, por razones obvias, **formales**.

6. *Método de Andrade*. Su propuesta, como ampliación de la propuesta de IDEAL, ofrece un mayor grado de detalle en lo que debe especificarse, aunque tiene todavía partes sin definir formalmente. **Semiformal**.

A modo de resumen, la Figura 2.43 recoge la distribución de los distintos métodos según el criterio de base formal. Se han utilizado las siguientes abreviaturas:

- EST, para los métodos estructurados.
- OOT, para el análisis de tareas de métodos orientados a objetos.
- OOC, para el análisis de clases de métodos orientados a objetos.
- AG, para métodos orientados a agentes.
- IDEAL, para el método del mismo nombre.
- CKADS, para CommonKADS,
- FM, para métodos formales matemáticos.
- FN, para métodos formales no matemáticos.
- ANDR, para el método de Andrade.

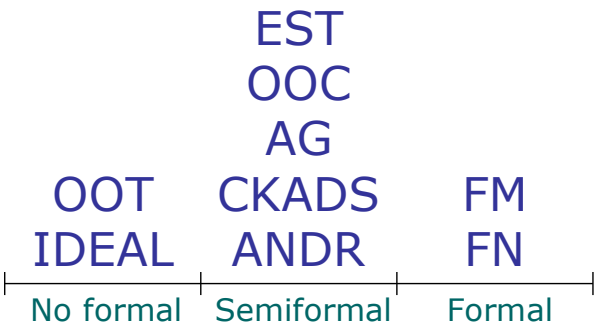


Figura 2.43 Clasificación de métodos según el criterio de base formal

2.7.2.2 Notación gráfica

1. *Métodos estructurados*. Mezclan una notación gráfica (DFD, diagramas Entidad – Relación, Diagramas de Estados), con notación textual (diccionario de datos, especificación de procesos). **Mixtos**.
2. *Métodos orientados a objetos*:
   a. *Análisis de tareas*: Mezclan diagramas (modelo casos de uso o diagramas de contexto) con especificaciones textuales (casos de uso o tareas). **Mixto**.
   b. *Análisis de clases*: Mezclan diagramas (relaciones entre clases, interacción entre clases), con especificaciones textuales (las clases de análisis). **Mixto**.
3. *Métodos orientados a agentes*: Utilizan diagramas más especificaciones textuales. **Mixto**.
4. *Métodos basados en el conocimiento*:
   a. *IDEAL*: Utiliza diagramas (diagrama de conceptos, descomposición de tareas, árboles de decisión, etc.) y especificaciones textuales en forma de tablas. **Mixto**.

b. *CommonKADS*: Utiliza diagramas (diagrama de conceptos y relaciones) más especificaciones en un lenguaje de modelado**. Mixto**.

5. *Métodos formales*: ambos grupos (matemáticos y no matemáticos) carecen de notaciones gráficas. **Sin notación gráfica**.
6. *Método de Andrade*. Como ampliación de IDEAL utiliza diagramas y especificaciones en forma de tablas. **Mixto**.

La Figura 2.44 recoge la clasificación de los métodos según el criterio de notaciones gráficas.

Figura 2.44 Clasificación de métodos según su notación gráfica

2.7.2.3 Notación textual

1. *Métodos estructurados*. La especificación de los procesos, diccionario de datos, etc., suele hacerse usando tablas con notación más o menos formal. **Tabular**.
2. *Métodos orientados a objetos*:
   a. *Análisis de tareas*: La descripción de los casos de uso o tareas suele estar escrita en lenguaje natural. **Informal**.
   b. *Análisis de clases*: La especificación de las clases suele expresarse en forma de tablas. **Tabular**.
3. *Métodos orientados a agentes*: Para especificar los agentes se rellena una serie de campos descriptivos. **Tabular**.
4. *Métodos basados en el conocimiento*:
   a. *IDEAL*: Los elementos textuales, como la tabla concepto / atributo / valor o la especificación de subpasos de bajo nivel se hace en forma de tablas. **Tabular**.
   b. *CommonKADS*: La especificación de los elementos se realiza en el lenguaje de modelado CML, que es más formal que las tablas de otros métodos. **Lenguaje no matemático**.
5. *Métodos formales*:
   a. *Matemáticos*: estos métodos usan lenguaje matemático. **Notación matemática**.

b. No matemáticos: estos métodos se diferencian de los anteriores por evitar el lenguaje matemático. **Lenguaje no matemático**.

6. *Método de Andrade*. Al igual que ocurre con IDEAL, se usan tablas para especificar los elementos de los modelos (en los catálogos). **Tabular**.

La Figura 2.45 recoge la clasificación de los métodos según el criterio de notaciones textuales.

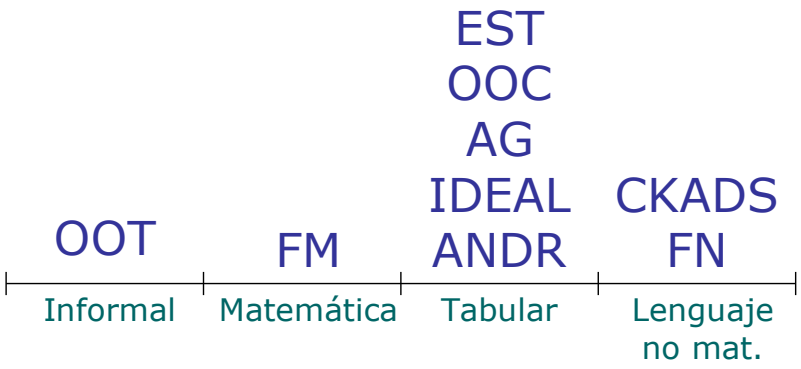


Figura 2.45 Clasificación de métodos según su notación textual

2.7.2.4 Independencia de paradigmas de diseño

1. *Métodos estructurados*. La gran mayoría de los elementos de modelado de las metodologías estructuradas llevan directamente a un diseño en el que se manejan de forma separada los procedimientos (diagrama de estructura) y los datos que utilizan. **Dependiente de diseño**.
2. *Métodos orientados a objetos*:
   a. *Análisis de tareas*: La descripción de los casos de uso o tareas es independiente de paradigmas de diseño, no llevando implícita la filosofía de la orientación a objetos. **Independiente de diseño**.
   b. *Análisis de clases*: Este análisis utiliza términos propios del diseño y de hecho sus autores lo consideran como una versión preliminar de un diseño orientado a objetos. **Dependiente de diseño**.
3. *Métodos orientados a agentes*: todas las metodologías descritas analizan el sistema presuponiendo que va a desarrollarse como un sistema basado en agentes, es decir, se utiliza el paradigma del diseño durante el análisis. **Dependiente de diseño**.
4. *Métodos basados en el conocimiento*:
   a. *IDEAL*: la conceptualización en esta metodología utiliza elementos independientes de diseño, aunque sí parecen depender de una arquitectura de sistema basado en el conocimiento. **Dependiente de arquitectura**.
   b. *CommonKADS*: aunque se trata de usar elementos de análisis independientes del diseño, los autores de la metodología defienden un "*diseño que conserva la estructura*" [Schreiber et al., 1999], que sólo es posible si el análisis está organizado de una forma adecuada para su posterior diseño e implementación. **Dependiente de arquitectura**.

5. *Métodos formales*: los dos grupos de métodos formales, por su propia naturaleza, utilizan elementos de modelado independientes del diseño. Sin embargo los tres métodos descritos (Z, VDM y PVS) llevan implícita una filosofía estructurada y las versiones más recientes, como Object-Z y VDM++ están basados en una filosofía orientada a objetos, filosofías que afectan a cómo se implementa el sistema final. **Dependientes de arquitectura**
6. *Método de Andrade*. Este método de modelado conceptual es más completo y no conlleva necesariamente la posterior utilización de una arquitectura de sistema basado en el conocimiento, como demuestra Andrade con su experimentación. **Independiente de diseño**.

La Figura 2.46 recoge la clasificación de los métodos según este criterio.

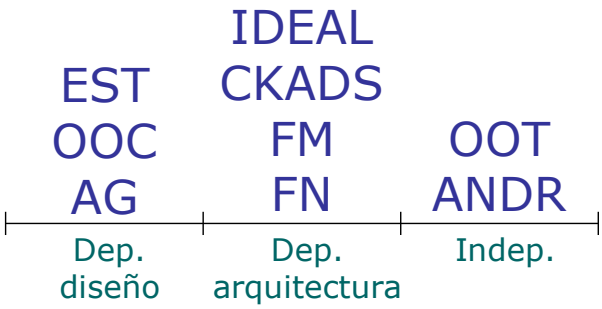


Figura 2.46 Clasificación de métodos según su dependencia de diseño

2.7.2.5 Adecuación a tipos de problemas

1. *Métodos estructurados*. Estos métodos están basados en que se puede tener una descripción completa y correcta del problema antes de comenzar el desarrollo, hecho que sólo ocurre en problemas algorítmicos. **Problemas algorítmicos**.
2. *Métodos orientados a objetos*:
   a. *Análisis de tareas*: esta parte del análisis no requiere un conocimiento preciso de todos los requisitos, pero sin embargo no cumple las dos propiedades exigidas por Schreiber y colegas para tratar problemas basados en conocimientos (representación de instancias y de control interno). **Problemas de soporte a la decisión**.
   b. *Análisis de clases*: esta parte empieza a representar el dominio del problema mediante las clases de análisis, pero no con el suficiente detalle como para abordar los problemas más complejos. **Problemas de soporte a la decisión**.
3. *Métodos orientados a agentes*: estas técnicas han sido definidas pensando en los sistemas multi-agente, que representan la máxima expresión de problemas complejos y basados en conocimientos. **Problemas basados en conocimientos**.
4. *Métodos basados en el conocimiento*: tanto IDEAL como CommonKADS fueron creados para resolver los problemas con gran dependencia de conocimientos. **Problemas basados en conocimientos**.
5. *Métodos formales*: todos los métodos formales requieren que el problema tratado pueda representarse completamente, de manera formal y no ambigua. Se trata pues

de métodos para **problemas algorítmicos**, tal como reconoce Blum, entre otros muchos autores [Blum, 1996]**.**

> *" ... para algunos problemas bien definidos, un lenguaje de especificación formal puede ser muy efectivo. Pero las dificultades esenciales del desarrollo de software están relacionadas con los problemas abiertos." [Blum, 1996]*.

6. *Método de Andrade*. Este método, como mejora de IDEAL, mantiene sus propiedades, lo que permite afrontar los problemas más complejos. **Problemas basados en conocimientos**.

En la Figura 2.47 se recoge la clasificación de métodos según el criterio de tipos de problemas.

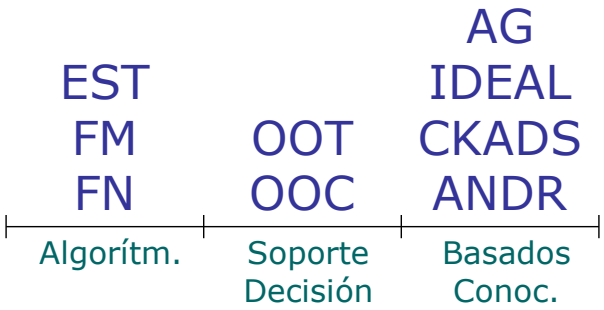


Figura 2.47 Clasificación de métodos según su adecuación a tipos de problemas

2.7.2.6 Resumen de la clasificación de los métodos

La clasificación anterior, realizada en cinco criterios, puede resumirse en la Figura 2.48. En esta figura se recogen los valores obtenidos en los cinco criterios por los nueve grupos de métodos identificados (estructurados, análisis de tareas en OO, análisis de clases en OO, métodos basados en agentes, IDEAL, CommonKADS, métodos formales con notación matemática, métodos formales con notación no matemática y, por último la propuesta de Andrade).

La primera conclusión de importancia es que ninguno de los métodos descritos hasta ahora reúne de forma simultánea los mejores valores en los cinco criterios.

Para entrar más en detalles conviene dividir los criterios en dos grupos. El primero recoge las cuestiones que, siguiendo a Aristóteles, pueden considerarse como accidentales (es decir, no inherentes al método): la existencia de notación gráfica y el tipo de notación textual. Tal como se comentó al definir estos criterios, lo deseable sería la existencia de notación gráfica y textual y, además, que la notación textual fuera un lenguaje (más riguroso que las tablas), pero evitando las notaciones matemáticas. De todos los métodos descritos, sólo CommonKADS obtiene la máxima puntuación en los dos criterios. En el extremo opuesto están los dos grupos de métodos formales, que tienen la gran desventaja de carecer de notación gráfica. Los seis métodos restantes obtienen la misma valoración para los criterios accidentales, muy cerca de CommonKADS.

Conviene indicar que los dos criterios considerados como accidentales son de gran importancia desde el punto de vista pragmático, es decir, en lo que respecta a la

aplicación práctica del método de análisis para proyectos reales. Sin embargo los problemas en estos dos criterios pueden mejorarse sin excesiva dificultad, ya que los métodos pueden ampliarse para incluir notaciones más adecuadas para su comprensión. De hecho, existe un considerable esfuerzo en mejorar los métodos formales mediante técnicas de visualización, como por ejemplo en [Dulac et al., 2002] y [Fenkam et al., 2002].

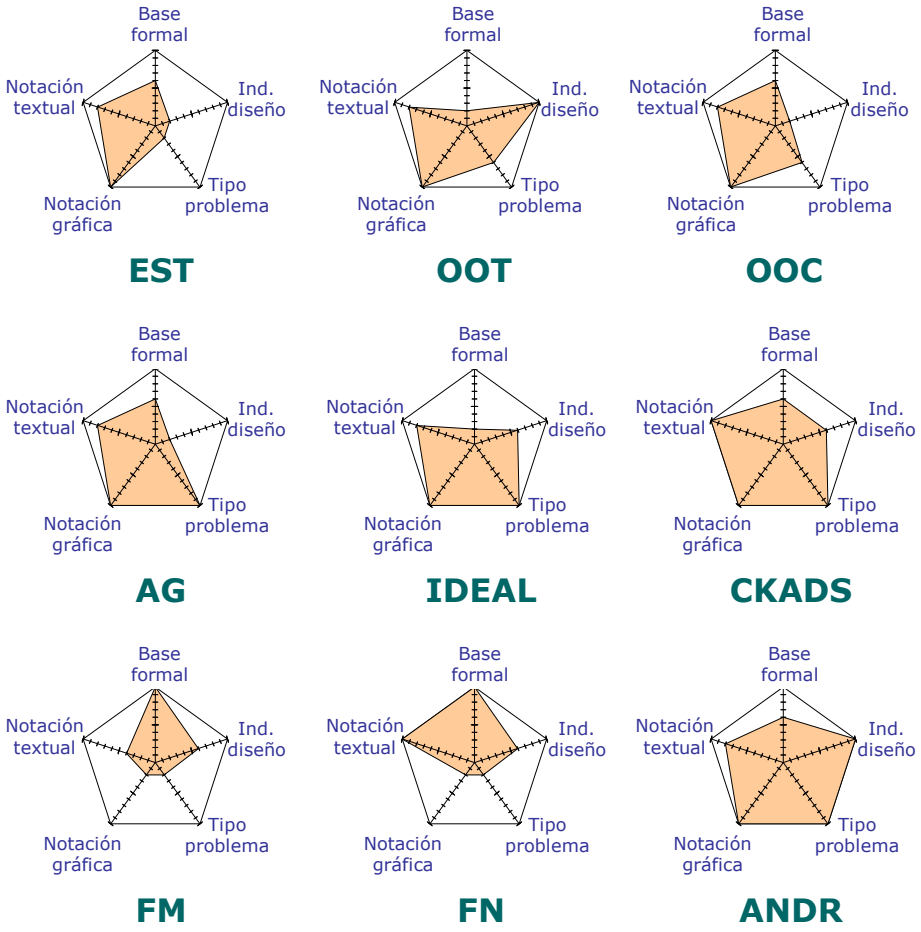


Figura 2.48 Clasificación de métodos según los criterios esenciales

El segundo grupo de criterios se refiere a características esenciales de los métodos: son inherentes a los mismos y, por tanto, más difíciles de modificar. Estos criterios son la base formal, la independencia del diseño y la adecuación a tipos de problemas. Dentro de la evaluación de los criterios esenciales destaca el método de Andrade, que obtiene la máxima valoración en dos de los tres criterios. Su parte menos fuerte es la falta de una base formal que permita definir de forma no ambigua todos los elementos utilizados para el modelado conceptual. El resto de métodos tiene una distribución poco uniforme, quedando de manifiesto que el poner énfasis en un criterio (por ejemplo, el tipo de problemas en CommonKADS e IDEAL), hace que los dos criterios restantes se resientan (así, estos métodos dependen de una arquitectura determinada del sistema).

2.8 SETCM

El estudio realizado en este capítulo sobre el estado de la cuestión en la disciplina del análisis del problema para el desarrollo de software, permite llegar a la conclusión de

que sigue siendo necesario trabajar en este campo, para lograr un método que tenga alta valoración en los cinco criterios definidos más arriba.

Éste es el objetivo de esta Tesis: la definición de un método de análisis que combine, al mismo tiempo (Figura 2.49):

- Una base formal.
- Una independencia de paradigmas y arquitecturas de diseño.
- Capacidad de analizar problemas de cualquier tipo.
- Disponibilidad de notación gráfica.
- Disponibilidad de una notación textual lo menos ambigua que sea posible.

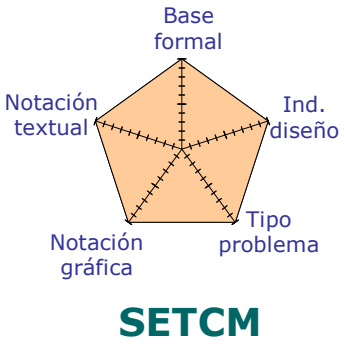


Figura 2.49 Los objetivos que pretende lograr SETCM

3 OBJETIVOS

La profunda discusión realizada en el estado de la cuestión permite enunciar de forma muy concreta los objetivos perseguidos con la realización de esta Tesis Doctoral.

El método propuesto en esta tesis debe permitir resolver los **problemas** planteados desde la introducción, a saber:

- Debe permitir *tratar el problema de la selección de enfoques* [Shapiro, 1997], con lo que el análisis del problema debe ser capaz de caracterizar los atributos más relevantes del mismo, independientemente del tipo de problema del que se trate.
- Debe *evitar los riesgos que pueden existir en la relación entre el análisis y el diseño* [Blum, 1996]:
    - Por un lado debe evitar que el análisis del problema sea demasiado superficial como para poder llevar a cabo un diseño adecuado con cierta garantía de éxito.
    - Por otro lado debe evitar que el análisis esté expresado en términos del diseño, lo cual llevaría a pensar más en la solución (es decir en el diseño) que en el problema planteado.
- Debe *evitar el dilema del análisis* [Alonso et al., 1999a], de forma que la selección del método propuesto no imponga ninguna restricción sobre la forma de continuar el desarrollo, salvo las derivadas de la propia naturaleza del problema.
- Debe *superar las barreras pragmáticas* [Knight et al., 1997] que afectan a la expansión del uso de métodos con base formal.

Por otra parte, el método propuesto deberá también cumplir con los **criterios mínimos** impuestos a cualquier técnica de análisis [Loucopoulos et al., 1995]:

- *Abstracción*, es decir, se debe poder representar el problema de manera abstracta.
- *No ambigüedad*, de forma que el modelo de análisis no admita varias interpretaciones.
- *Facilidad para la comprensión* del problema, de forma que tanto los desarrolladores como los clientes puedan comprender fácilmente el modelo resultante.
- *Facilidad para la verificación* del modelo, de forma que se puedan detectar de forma automática problemas en la especificación de un modelo de análisis.
- *Independencia* respecto a otras técnicas de desarrollo, de forma que pueda utilizarse de forma complementaria con cualquier técnica de diseño de software.

Las **características** fundamentales que debe tener el método propuesto para resolver esos problemas y cumplir los criterios mínimos son:

- Debe tener *una base formal*, que sea al mismo tiempo sencilla de comprender y consistente.
    - La base formal elegida es la *teoría de conjuntos*, por ser considerada una teoría indispensable para la conceptualización de problemas [Andrade, 2002].
    - La existencia de una base formal permite cumplir los criterios mínimos de abstracción, no ambigüedad y facilidad para la verificación (es decir, ¿es correcto el modelo con respecto a su sintaxis?).

- Debe tener en cuenta *aspectos pragmáticos*.
  - Para ello se deberán *evitar notaciones matemáticas* en la medida de lo posible.
  - La técnica propuesta deberá incluir la definición de una notación textual, para facilitar la comprensión del modelo generado:

    > *"La notación matemática no es familiar ... Por qué no se usaron palabras, que son comprendidas por todos, en vez de símbolos matemáticos" [Knight et al., 1997]*
    >
    > *"Los lenguajes de especificación formal no se han utilizado ampliamente en la industria. Un factor es su legibilidad" [Dulac et al., 2002]*

  - También se deberá incorporar una *notación gráfica*, que complemente a la notación textual para facilitar la comprensión de problemas complejos:

    > *"Incluso con una notación formal diseñada pensando en la legibilidad, la complejidad del comportamiento descrito abruma al lector ... La visualización es una forma de ayudar a las personas a comprender grandes cantidades de información" [Dulac et al., 2002]*
    >
    > *"Hay que evitar que el modelo parezca código fuente, sea demasiado largo y tenga demasiado texto" [Knight et al., 1997]*
    >
    > *"La representación de un diagrama 2D es mucho más expresiva que el texto" [Spinellis, 2003]*

  - Los aspectos pragmáticos están enfocados a mejorar la facilidad de comprensión del problema, que es una característica fundamental para poder realizar la validación de un modelo (así se responde a la pregunta: ¿el modelo representa correctamente el problema planteado?).
- Por último, *no deberá estar sujeto a ningún tipo de problema ni tipo de solución*:
  - Lo fundamental es evitar el uso de términos de diseño (como objeto, clase, operación de clase, módulo, etc.) para que la selección de técnicas de diseño dependa exclusivamente de las características del problema planteado.
  - Además el método propuesto deberá tener la capacidad expresiva suficiente para poder abarcar el análisis de problemas tanto de Ingeniería del Software como de la Ingeniería del Conocimiento.

En conclusión, el **objetivo** de esta tesis es definir una técnica de análisis que cumpla con las características propuestas mediante la definición de:

- Los **elementos** que componen los modelos resultantes, que deberán permitir reflejar tanto el dominio del problema, como el comportamiento deseado.
- Una **formalización** de los elementos anteriores utilizando la teoría de conjuntos.
- Una **notación textual** que permita especificar de forma completa todos los elementos de los modelos y que sea lo más fácil de comprender que sea posible.
- Una **notación gráfica** que sirva como complemento de la notación textual para facilitar la comprensión de los grandes rasgos de un problema.

- Unas nociones sobre el **proceso de análisis**, aunque esta tesis no define una metodología completa de análisis, sino únicamente una técnica para representar modelos de problemas.

4 SOLUCIÓN PROPUESTA

La exposición de la solución propuesta en esta tesis va a dividirse en cinco secciones. En primer lugar se definirán todos los elementos que forman parte de SETCM. En segundo lugar se formalizarán dichos elementos, usando teoría de conjuntos. El tercer apartado contiene una definición de un lenguaje textual para representar todos los elementos. Después se mostrará una notación gráfica que permite representar los elementos más importantes. Por último se hará una introducción a un posible proceso de análisis.

4.1 DEFINICIONES

En este apartado se van a dar las definiciones de los elementos principales que conforman el método propuesto. Según se vayan definiendo elementos se les va a asociar una abreviatura que luego se utilizará en el apartado de formalización de estos elementos. El anexo A recoge todas las abreviaturas y el resto de símbolos utilizados en la formalización.

Dentro de SETCM, se define un modelo conceptual o **conceptualización** (CNC_i) como un par formado por dos modelos: el estructural y el de comportamiento.

Un **Modelo Estructural** (EST_i) permite representar la estructura de un dominio de conocimiento (elementos y relaciones entre los mismos), así como el estado de ese domino en un momento dado.

Un **Modelo de Comportamiento** (COM_i) permite representar la resolución de problemas planteados en un dominio concreto. Esta resolución de problemas se plantea como el proceso de realizar cambios en el estado de un sistema, desde un estado inicial hasta el estado en el que se llega a la solución.

En los apartados siguientes se definirán los elementos que componen cada uno de estos dos modelos. Con el fin de facilitar la comprensión de estas definiciones, se van a ver acompañadas de diagramas que explican la relación entre los distintos elementos definidos. Para estos diagramas se va a utilizar la notación mostrada en la Figura 4.1.

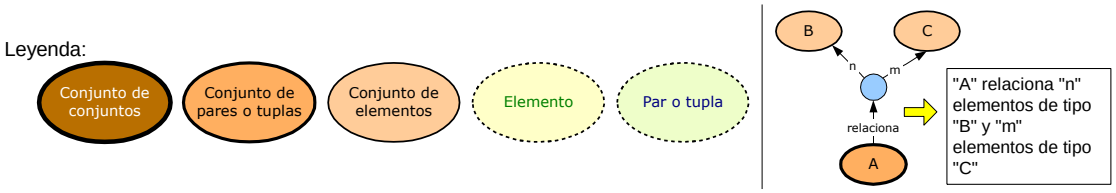


Figura 4.1 Notación utilizada en los diagramas que acompañan las definiciones

A lo largo de todo este capítulo de solución propuesta se va a ir desarrollando un ejemplo cuya descripción inicial se muestra a continuación.

Equipo de fútbol

El ejemplo que se va a tratar a lo largo de la exposición se basa en representar el dominio de un equipo de fútbol. Dentro de este dominio se van a considerar los siguientes elementos más relevantes:

- *Persona*: representa cualquier persona relacionada con el equipo de fútbol. Las personas podrán ser jugadores, técnicos (uno de los cuales es el entrenador) y directivos (uno de los cuales es el presidente). Toda persona tendrá un nombre completo, una fecha de nacimiento y una nacionalidad (país) que podrá ser doble (o incluso triple).

- *Jugador de fútbol*. Tienen una posición natural (portero, defensa central, lateral izquierdo, etc.), y unas aptitudes (fuerza, velocidad, capacidad de robar balones, regate, remate de cabeza, etc.). Los jugadores podrán estar lesionados o no. Existirán tres tipos de jugadores clasificados por su origen: nacionales, comunitarios y extranjeros. Existirán dos tipos básicos de jugadores en función de su posición: porteros y jugadores de campo. Cada uno de ellos tendrá un conjunto de aptitudes distintas.
- *Equipo de fútbol* que competirá en una categoría determinada (primera división, segunda, etc.) y que tendrá una serie de jugadores contratados.
- *Pertenencia* de una persona a un equipo. Podrá ser el fichaje de un jugador, el cargo de presidente, el cargo de entrenador, etc. En el caso de un fichaje de jugador, definirá un importe anual y una cláusula de rescisión.
- *Alineación* de un equipo para un partido determinado. Tendrá asociada una distribución táctica (4-4-2, 4-3-3, etc.). Para una misma táctica podrán existir varias alineaciones de uno o varios equipos. La alineación define la posición que ocupan los jugadores del equipo dentro de la táctica elegida.

4.1.1 Modelo Estructural

El modelo estructural está formado por ocho conjuntos: tipos, conceptos, atributos de concepto, asociaciones, atributos de asociación, clasificaciones de concepto y clasificaciones de asociación. Todos estos elementos se definirán en los apartados siguientes.

4.1.1.1 Tipos y Valores

En todo modelo existen elementos básicos cuya definición y existencia no depende del resto de elementos del modelo y es también independiente del tiempo. En el caso del modelo estructural estos elementos básicos se denominan valores y se agrupan en conjuntos denominados tipos. La Figura 4.2 muestra las relaciones entre estos elementos.

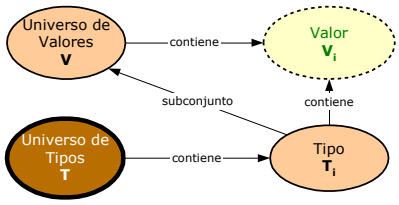


Figura 4.2 Relaciones entre tipos y valores

Se define el **Universo de Valores (V)** como el conjunto de todos los valores que se pueden manejar en un modelo conceptual.

Se define un **valor (V_i)** como un elemento básico del modelo estructural cuya definición no depende del resto de elementos del modelo (por ejemplo el número "10" o la cadena de caracteres "Juan"). Los valores no se crean ni se destruyen, existen siempre.

Se define el **Universo de Tipos (T)** como el conjunto de todos los tipos que pueden definirse en un modelo conceptual.

Se define cada **tipo (T_i)** como un conjunto de valores. Por tanto un tipo no es más que un subconjunto del universo de valores V. Los tipos tampoco cambian con el tiempo.

Dentro de los tipos, se distinguen dos grandes grupos, los tipos básicos y los tipos derivados.

Se define el **universo de los tipos básicos** (TB) como el conjunto de todos los tipos básicos.

Se define un **tipo básico** como aquél cuya definición es axiomática y no depende de ningún otro tipo. En SETCM se consideran cuatro tipos básicos, que son aplicables a cualquier dominio:

- El tipo de los **números reales** (R).
- El tipo de los **números enteros** (Z).
- El tipo de las **cadenas de caracteres** (CAD). Cada uno de los valores de este tipo está formado por secuencias de caracteres alfanuméricos de cualquier longitud.
- El tipo de los **valores booleanos** (B). Este tipos sólo tiene dos valores, v (verdadero) y f (falso).

Es importante señalar que, en principio, esta lista de tipos básicos no es cerrada. Para un dominio concreto podría existir algún tipo básico más. Así, por ejemplo, se podría definir un tipo básico con los valores de la lógica trivalente (verdadero, falso y desconocido).

Todo tipo (básico o derivado) tiene asignado un **nombre** que es una cadena de caracteres (es decir, un elemento de CAD) y que debe ser único dentro del conjunto de tipos de un modelo conceptual. Los nombres asignados a los tipos básicos son: "Real" para R, "Entero" para Z, "Cadena" para CAD y "Booleano" para B.

Se define el **universo de los tipos derivados** (TD) como el conjunto de todos los tipos derivados.

Se define un **tipo derivado** como un subconjunto de otro tipo (básico o derivado), que se especifica imponiendo restricciones sobre los valores del tipo del que deriva. Los elementos que permiten definir un tipo derivado son tres:

- *Nombre* del tipo derivado.
- *Tipo base*, que es el tipo del que deriva.
- *Restricción*, que define qué valores del tipo base pertenecen también al tipo derivado. Las restricciones pueden ser de cuatro tipos:
    - *Restricción de reales*: define los valores válidos del tipo derivado mediante un único intervalo de valores reales o bien mediante una unión de intervalos. Siguiendo la definición habitual en matemáticas, cada intervalo tiene dos extremos (inferior y superior), que pueden ser abiertos o cerrados. En este caso todo valor del tipo derivado debe pertenecer al intervalo o intervalos de la restricción.
    - *Restricción de enteros*: define los valores válidos del tipo derivado mediante un único intervalo de valores reales o bien mediante una unión de intervalos. En

este caso todo valor del tipo derivado debe pertenecer al intervalo o intervalos de la restricción.

- *Restricción de cadenas*: define un conjunto con aquellas cadenas que se consideran valores válidos del tipo derivado correspondiente. Se debe cumplir que estas cadenas son valores válidos del tipo base.
- *Restricción de diferencia*: define un subconjunto del tipo base considerado mediante diferencia de conjuntos entre el tipo base y otro tipo derivado. Para toda restricción de diferencia debe cumplirse que su tipo derivado sea uno de los tipos derivados ya definidos del tipo base asociado a la definición de tipo derivado correspondiente. En este caso todo valor del tipo derivado definido mediante la restricción debe pertenecer al tipo base y no pertenecer al tipo de la restricción.

A lo largo del presente documento se va a utilizar la notación matemática habitual para representar intervalos. Así, el intervalo con extremo inferior abierto 5 y extremo superior cerrado 15 se representará como (5, 15].

Equipo de fútbol

Tipos derivados para el ejemplo del equipo de fútbol:

- **País**: tipo que representa los nombres de los países del mundo. Su tipo base es CAD y su restricción es el conjunto de todos los nombres de países: "España", "Portugal", "Francia", ..., "Argentina", "Brasil", ..., "Japón", "Corea", ...
- **País comunitario**: representa los nombres de países de la unión europea, como un tipo derivado de "País" con la siguiente lista de cadenas: "Alemania", "Austria", "Bélgica", "Dinamarca", "España", "Finlandia", "Francia", "Gran Bretaña", "Grecia", "Irlanda", "Italia", "Luxemburgo", "Países Bajos", "Portugal" y "Suecia"
- **País nacional**: representa el nombre de país de un jugador nacional. Su tipo base es "País comunitario" y sólo admite un valor: "España".
- **País extranjero**: nombres de países no comunitarios. Es la diferencia entre "País" y "País comunitario"
- **Posición**: nombres de posiciones que puede ocupar un jugador en el campo. Deriva de CAD con los valores: "Portero", "Lateral izquierdo", "Lateral derecho", "Defensa central", "Líbero", "Medio centro", "Interior izquierdo", "Interior derecho", "Extremo izquierdo", "Extremo derecho", "Media punta", "Delantero"
- **Posición portero**: deriva de "Posición" y sólo admite el valor "Portero".
- **Posición campo**: representa todas las posiciones excepto la de portero. Es la diferencia entre "Posición" y "Posición portero".
- **Estado**: representa el estado físico de un jugador. Deriva de CAD con valores: "bien", "lesionado".
- **Valoración**: que representa la valoración realizada sobre una de las aptitudes que se consideran de los jugadores (velocidad, fuerza, etc.) como un número real entre 0 y 10. Deriva de R y define el intervalo [0, 10].
- **Dorsal**: representa el número de dorsal que puede asignarse a un jugador. Deriva de Z con intervalo [1, 25].

- **Categoría**: representa la categoría en la que juega un equipo de fútbol. Deriva de CAD con valores: "Primera", "Segunda", "Segunda B", "Tercera", "Regional".
- **Carácter**: representa el carácter ofensivo, defensivo o neutro de una táctica. Deriva de CAD con valores: "ofensivo", "defensivo", "neutro".
- **Día**: representa los números de día dentro de un mes. Deriva de Z con intervalo [1, 31].
- **Mes**: representa los números de mes del año. Deriva de Z con intervalo [1, 12].
- **Año**: representa los años relevantes para el problema. Deriva de Z con intervalo [1800, ∞).

4.1.1.2 Conceptos e Instancias

Se define el **universo de Instancias (I)** como el conjunto de todas las instancias.

Se define cada **instancia (I_i)** como un elemento concreto que aparece en el dominio bajo estudio. Las instancias no se pueden descomponer más y se diferencian de los valores en que se pueden crear y destruir a lo largo del tiempo de vida del sistema bajo estudio.

Todas instancia se caracteriza por tener un nombre (un elemento de CAD), que debe ser único. El **nombre de una instancia** es único, es decir, dos instancias diferentes deben tener nombres diferentes, independientemente de los conceptos a los que pertenezcan. Esto permite utilizar el nombre de una instancia (una cadena) como su identificador.

Cada instancia se caracterizará por tener asignados una serie de valores (mediante atributos) y por estar relacionada con otras instancias (mediante asociaciones) tal y como se verá en apartados siguientes.

Una restricción importante es que no tiene sentido ninguna instancia que no esté relacionada al menos con un valor o bien con otra instancia. En otro caso habría que considerar esa instancia más bien como un valor de tipo cadena.

Se define el **universo de Conceptos (C)** como el conjunto de todos los conceptos.

Se define cada **concepto C_i** como un conjunto de instancias, de forma que todas ellas cumplen una condición. Así un concepto es un subconjunto del universo de instancias I.

La Figura 4.3 representa de forma gráfica las relaciones que existen entre las instancias y los conceptos.

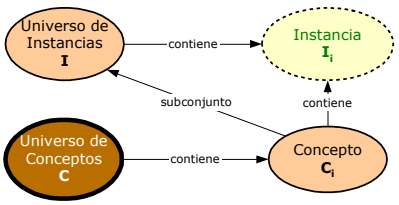


Figura 4.3 Relaciones entre instancias y conceptos

Los conceptos se especifican mediante dos elementos:

- *Nombre* del concepto, que debe ser único dentro de un modelo estructural.

- *Invariante* del concepto, que es la condición que deben cumplir todas las instancias del concepto.

Equipo de fútbol

Seguidamente se recogen los conceptos principales de este ejemplo:

- **Persona**: una persona cualquiera
- **Jugador**: una persona que es jugador de fútbol
- **Equipo**: un equipo de fútbol que juega en una categoría
- **Alineación**: la alineación de un equipo para un partido.
- **Táctica**: una táctica asignada a una alineación (4-4-2, por ejemplo).
- **Puesto**: cada uno de los puestos definidos en una táctica.
- **Fecha**: una fecha representada por un día, un mes y un año. Se define como invariante la condición de fecha correcta, que restringe los valores de día en función del mes y el año.

Para terminar el apartado dedicado a las instancias y conceptos conviene hacer una breve comparación entre tipos y conceptos. Desde el punto de vista de aplicación del modelo estructural para la conceptualización, **la distinción entre *concepto* y *tipo* es totalmente dependiente del dominio o incluso de la tarea que se esté analizando**. Habrá términos (como Color) que podrán ser conceptos (un color en un dominio en el que se manejan todos sus componentes y además tiene relaciones con otros conceptos) o tipos (un color en un dominio en el que sólo interese su nombre).

La diferencia fundamental radica en que todos los elementos de un tipo (los valores) son conocidos como parte de la estructura, mientras que los elementos de un concepto se crean y se destruyen como consecuencia de cambios de estado en el sistema.

Equipo de fútbol

En este ejemplo, el término "**Categoría**" puede ser un tipo o un concepto dependiendo del problema:

- Será un tipo (que es como se ha definido) si únicamente se necesita conocer la categoría de un equipo, sin necesidad de más datos.
- Será un concepto si la categoría debe recoger información sobre la competición: equipos participantes, clasificación, resultados, etc.

El término "**País**" también puede ser un tipo o un concepto.

- Será un tipo (que es como se ha definido) si únicamente se necesita conocer el nombre del país, sin necesidad de más datos.
- Será un concepto si se necesita más información sobre ese país: superficie, número de habitantes, PIB, etc.

4.1.1.3 Atributos de Concepto

Los atributos se utilizan para definir propiedades de los conceptos, que luego tendrán valor para cada una de sus instancias. Por ejemplo, el atributo "nombre" del concepto "Persona".

La Figura 4.4 ilustra gráficamente la definición de atributo como un conjunto de pares ordenados (*instancia*, *valor*). En esa figura se utiliza como ejemplo el atributo "curso"

del concepto “Asignatura”, que define el número de curso (tipo derivado “Curso” como un número natural entre 1 y 5) al que pertenece una asignatura de una carrera universitaria.

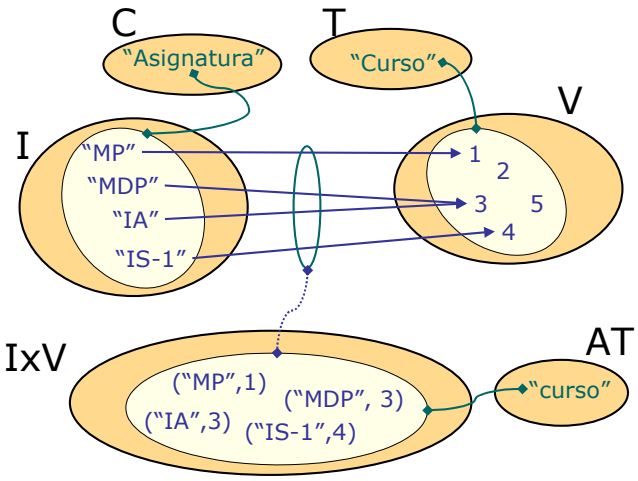


Figura 4.4 Un atributo es una correspondencia entre un concepto y un tipo

Los ejemplos de elementos del atributo “curso” del concepto “Asignatura” que aparecen en la figura son:

- *(“MP”,1)*: la asignatura “Metodología de la programación” es de primero
- *(“MDP”,3)*: la asignatura “Modelos de desarrollo de programas” es de tercero
- *(“IA”, 3)*: la asignatura “Inteligencia Artificial” es de tercero
- *(“IS-1”,4)*; la asignatura “Ingeniería del Software – 1” es de cuarto

Una vez visto el concepto intuitivo de atributo se procede a definir más formalmente los elementos que especifican un atributo en el modelo estructural.

Se define el **universo de atributos de concepto (AT)** como el conjunto de todos los atributos de concepto.

Se define cada **atributo de concepto (AT_i)** como una correspondencia entre un concepto (conjunto de instancias) y un tipo (conjunto de valores). En otras palabras, un atributo es un subconjunto del producto cartesiano entre el universo de instancias y el universo de valores. Por lo tanto un atributo es un conjunto de pares ordenados llamados instancias de atributo.

Se define **universo de instancias de atributo** (**IAT**) como el conjunto de todas las instancias de atributo.

Se define cada **instancia de atributo (IAT_i)** como un par ordenado en el que el primer elemento es una instancia de un concepto y el segundo elemento es un valor de un tipo. Las instancias de atributo se crean y se destruyen a lo largo del tiempo.

La Figura 4.5 representa gráficamente las relaciones entre conceptos, tipos y atributos de concepto.

Los atributos se especifican mediante cinco elementos:

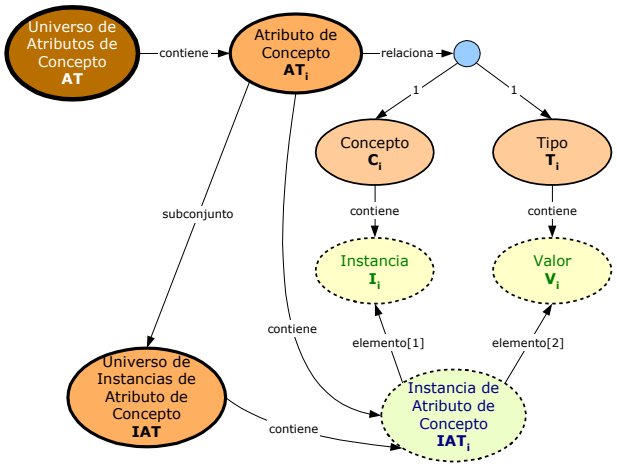


Figura 4.5 Relaciones entre atributo, concepto y tipo

- *Nombre* del atributo, que debe ser único para todos los atributos de un mismo concepto.
- *Concepto* al que pertenece el atributo. El identificador del atributo se forma uniendo el nombre del atributo y el nombre del concepto.
- *Tipo* en el que toma valores el atributo.
- *Grado* de participación del concepto, que restringe el número mínimo de valores que deben estar asignados a cada instancia del concepto. Puede ser "opcional" (las instancias pueden no tener valor asignado) u "obligatorio" (todas las instancias deben tener valor asignado).
- *Límite* de participación del concepto, que restringe el número máximo de valores que pueden estar asignados a cada instancia de concepto. Puede ser "único" (las instancias pueden tener como mucho un valor) o "múltiple" (las instancias pueden tener más de un valor asignado).
- *Valor por omisión*, que permite asociar un valor a un atributo para que sea utilizado por todas las instancias del concepto que no tengan valor asignado.

Equipo de fútbol

Seguidamente se describen atributos del ejemplo, ordenados por el concepto en el que se definen.

El concepto **Fecha** tiene tres atributos: número de día, número de mes y año.

- **día**, con tipo "Día", grado obligatorio, límite único y sin valor por omisión.
- **mes**, con tipo "Mes", grado obligatorio, límite único y sin valor por omisión.
- **año**, con tipo "Año", grado obligatorio, límite único y sin valor por omisión.

El concepto **Persona** tiene dos atributos: el nombre completo y el país (o países) que definen su nacionalidad.

- **nombre**, con tipo CAD, grado obligatorio, límite único y sin valor por omisión.
- **país**, con tipo "País", grado obligatorio, límite múltiple (una persona puede tener varias nacionalidades) y sin valor por omisión.

El concepto **Jugador** tiene seis atributos: posición de juego, número de dorsal, valoración de fuerza, velocidad, resistencia y, por último, su estado físico.

- **posición**, con tipo "Posición", grado obligatorio, límite único y sin valor por omisión.
- **dorsal**, con tipo "Dorsal", grado obligatorio, límite único y sin valor por omisión.
- **fuerza**, con tipo "Valoración", grado obligatorio, límite único y sin valor por omisión.
- **velocidad**, con tipo "Valoración", grado obligatorio, límite único y sin valor por omisión.
- **resistencia**, con tipo "Valoración", grado obligatorio, límite único y sin valor por omisión.
- **estado**, con tipo "Estado", grado obligatorio, límite único y valor por omisión "bien" (esto quiere decir que un jugador está bien salvo que se diga lo contrario).

El concepto **Equipo** tiene dos atributos: su nombre y la categoría en la que compite.

- **nombre**, con tipo CAD, grado obligatorio, límite único y sin valor por omisión.
- **categoría**, con tipo "Categoría", grado obligatorio, límite único y sin valor por omisión.

El concepto **Alineación** tiene un único atributo (su nombre), ya que todo su significado reside en su relación con otros conceptos.

- **nombre**, con tipo CAD, grado obligatorio, límite único y sin valor por omisión.

El Concepto **Táctica** tiene dos atributos: su nombre y su carácter (que puede ser ofensivo, defensivo o neutro).

- **nombre**, con tipo CAD, grado obligatorio, límite único y sin valor por omisión.
- **carácter**, con tipo "Carácter", grado obligatorio, límite único y valor por omisión "neutro" (el carácter de una táctica es neutro salvo que se especifique lo contrario).

El Concepto **Puesto** tiene dos atributos: la posición en el campo y las posiciones afines (para el caso en el que no se encuentre un jugador que juegue en la posición adecuada).

- **posición**, con tipo "Posición", grado obligatorio, límite único y sin valor por omisión.
- **afines**, con tipo "Posición", grado obligatorio, límite múltiple y sin valor por omisión.

Una vez vistos los atributos, pueden mostrarse algunos **ejemplos de instancias de atributo**:

```
"Persona"."nombre" = {("Loïc", "Loïc Antonio Martínez Normand"), ...}

"Persona"."país" = {("Loïc", "España"), ("Loïc", "Francia"), ...}

"Jugador"."posición" = {("Casillas", "Portero"), ("Kluivert",
                        "Delantero"), ...}
```

```
"Jugador"."fuerza" = {("Figo", 7), ("Rivaldo", 8), ...}
"Jugador"."velocidad" = {("Figo", 8), ("Rivaldo", 9), ...}
"Jugador"."resistencia" = {("Figo", 8), ("Rivaldo", 7), ...}
"Jugador"."estado" = {("Figo", "bien"), ("Rivaldo", "lesionado"),
                      ("Casillas", "bien"), ...}
"Equipo"."nombre" = {("Real Madrid", "Real Madrid Club de Fútbol"),
                     "Barcelona", "Fútbol Club Barcelona"), ...}
"Equipo"."categoría" = {("Barcelona", "Primera"), ("Alcalá", "Tercera"),
                        ...}
"Alineación"."nombre" = {("RM 4-4-2", "4-4-2 con doble pivote"), ...}
"Fecha"."día" = {("16-10-1969", 16), ...}
"Fecha"."mes" = {("16-10-1969", 10), ...}
"Fecha"."año" = {("16-10-1969", 1969), ...}
```

4.1.1.4 Asociaciones y Tuplas

Las asociaciones representan *correspondencias* establecidas entre conceptos, tipos e incluso otras asociaciones del modelo estructural.

Las asociaciones se consideran elementos del mismo nivel que los conceptos, y por lo tanto el modelo estructural incluye la definición de invariantes de asociación, atributos de asociación (descritos en el apartado siguiente), así como la participación de asociaciones en otras asociaciones.

El caso más habitual de asociación es la que se establece entre una serie de conceptos participantes (dos o más). Esta asociación será un conjunto de tuplas formadas por instancias de los conceptos participantes. En ese caso, la asociación es un subconjunto del producto cartesiano de los conceptos participantes.

La Figura 4.6 muestra gráficamente un ejemplo de este tipo de asociaciones simples: la asociación "Propiedad" establecida entre los conceptos "Persona" y "Vivienda" que representa la relación entre las viviendas y sus propietarios. En la figura aparecen algunos ejemplos de pares ordenados pertenecientes a la asociación, que indican lo siguiente:

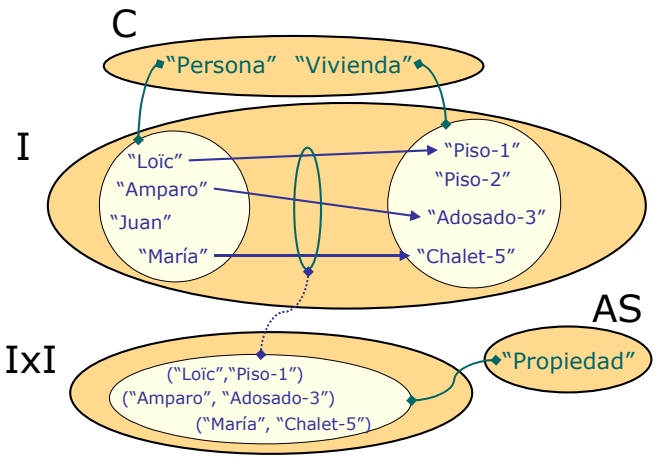


Figura 4.6 Una asociación puede ser una correspondencia entre dos o más conceptos

- "Loïc" es propietario de la vivienda "Piso-1".

- "Amparo" es propietaria de la vivienda "Adosado-3".
- "María" es propietaria de la vivienda "Chalet-5".
- "Juan" no es propietario de ninguna vivienda, y "Piso-2" no tiene propietario.

También se admiten asociaciones en las alguno de los participantes (o incluso todos) son tipos. Así, por ejemplo, se puede definir una asociación "Bisiesto" entre el tipo "Año" y el tipo de los valores booleanos (B), de forma que a cada valor de Año se le asigna el valor cierto si es bisiesto y falso en caso contrario.

La Figura 4.7 muestra gráficamente esta asociación, con algunos ejemplos de pares ordenados pertenecientes a la asociación (son pares de valores). Así, por ejemplo, el año 2000 es bisiesto, mientras que los años 2001, 2002 y 2003 no lo son.

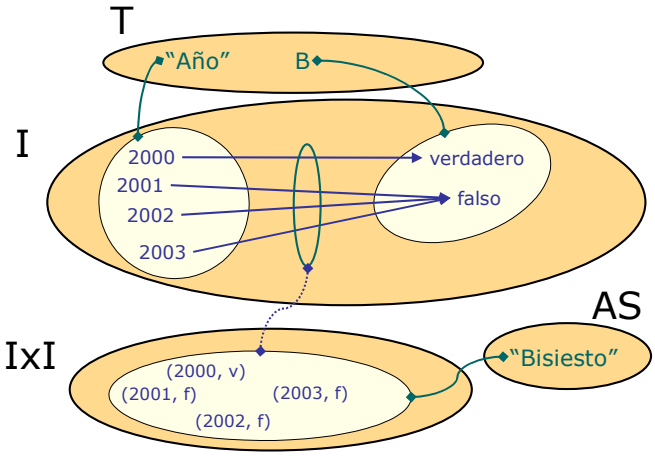


Figura 4.7 Una asociación puede ser una correspondencia entre tipos

Las asociaciones en las que sólo participan conceptos o tipos se denominan *asociaciones simples*, ya que cada uno de los elementos de sus tuplas (instancias de concepto o valores de tipo) son elementos simples, que no admiten más descomposición.

Tal y como se ha comentado anteriormente, también pueden existir asociaciones más complejas en las que uno o más participantes sean, a su vez, asociaciones. Estas asociaciones se denominan *asociaciones de orden superior*.

Una asociación de este tipo será un conjunto de tuplas de forma que cada elemento de la tupla sea una instancia de concepto, un valor de un tipo o bien un elemento de una asociación (una tupla), según corresponda. De esta forma, la asociación será un subconjunto del producto cartesiano de los conceptos, tipos y asociaciones participantes.

La Figura 4.8 muestra un ejemplo de asociación de orden superior. En concreto se trata de la asociación "Fecha de propiedad", establecida entre la asociación "Propiedad" (del ejemplo anterior) y el concepto "Fecha". Esta asociación permite asignar a cada tupla de la asociación "Propiedad" la fecha en la cual comenzó dicha propiedad.

En este ejemplo se han establecido las siguientes tuplas de la asociación "Fecha de propiedad":

- La persona "Loïc" es propietaria de la vivienda "Piso-1" desde la fecha "20.7.1999".

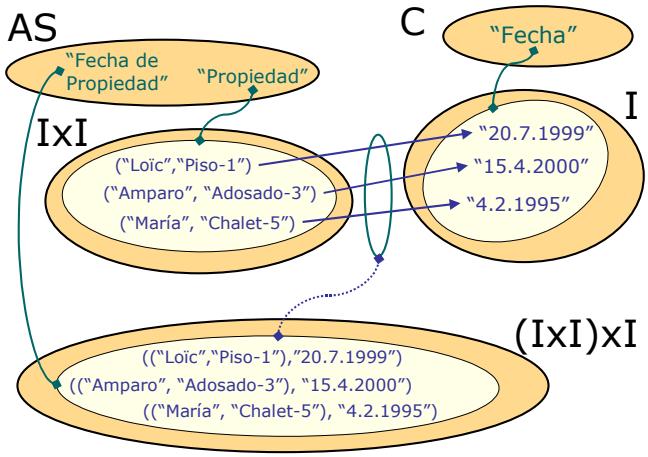


Figura 4.8 Asociación en la que participan otras asociaciones (orden superior)

- "Amparo" es propietaria de "Adosado-3" desde la fecha "15.4.2000"
- "María" es propietaria de "Chalet-5" desde la fecha "4.2.1995"

Una característica importante de las asociaciones es su *direccionalidad*: toda asociación se define entre uno o más orígenes y un único destino. En el ejemplo "Propiedad" visto más arriba el origen sería el concepto "Persona" y el destino el concepto "Vivienda". De esta forma se podrá utilizar notación relacional ("Propiedad"("Loïc", "Piso-1")) o funcional ("Propiedad"("Loïc")= "Piso-1") según sea más adecuado para el objetivo perseguido.

Una vez visto el concepto intuitivo de asociación se procede a definir los elementos que permiten especificar asociaciones en el modelo estructural.

Se define el **universo de Asociaciones (AS)** como el conjunto de todas las asociaciones que pueden aparecer en modelos estructurales.

Se define cada **asociación** (**AS_i**) como una correspondencia de conjuntos, en la que los conjuntos participantes pueden ser conceptos, tipos u otras asociaciones. Por lo tanto asociación será un conjunto de tuplas, llamadas instancias de asociación.

Se define el **universo de instancias de asociación** (**IAS**) como el conjunto de todas las instancias de asociación posibles.

Se define cada **instancia de asociación** (**IAS_i**) como una tupla, de forma que cada elemento de la tupla es una instancia de concepto, un valor de un tipo o una instancia de asociación, según corresponda con cada participante de la asociación a la que pertenece la tupla. Las instancias de asociación se crean y se destruyen a lo largo del tiempo.

La Figura 4.9 muestra gráficamente las relaciones existentes entre asociaciones, tipos y conceptos según las definiciones anteriores.

Las asociaciones se especifican mediante siete elementos:

- *Nombre* de la asociación, que debe ser único para todas las asociaciones que comparten el mismo origen.
- *Dimensión* de la asociación, que define el número de participantes.

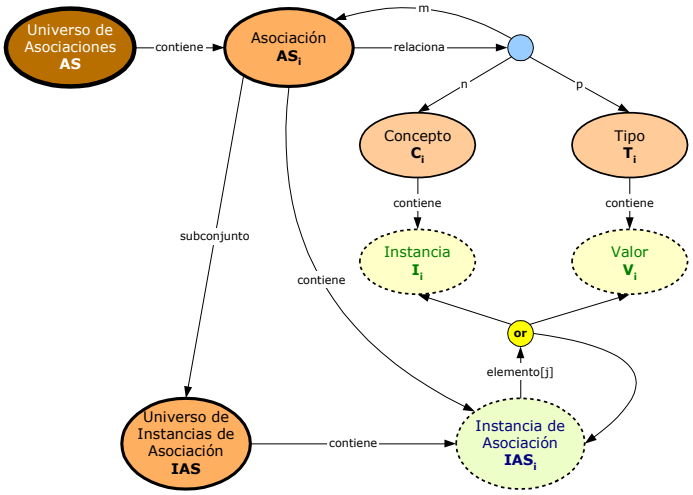


Figura 4.9 Relaciones entre asociaciones, tipos y conceptos

- *Origen* de la asociación, que es una tupla de roles de asociación que permiten definir los participantes del origen (conceptos, tipos o asociaciones) y sus restricciones de participación en la asociación. El número de elementos del origen debe ser igual a la dimensión de la asociación menos 1. El identificador de la asociación se forma uniendo el nombre de la asociación y los identificadores de los participantes del origen.
- *Destino* de la asociación, definido como un rol de asociación.
- *Completitud* de la asociación, que indica si cualquier combinación de elementos del origen tiene asociado o no un elemento del destino. Podrá tener dos valores:
  - "Completa": si para cualquier posible tupla de elementos del origen debe existir al menos un elemento del destino de la asociación.
  - "Parcial": si existen tuplas de elementos del origen que no están relacionados con ningún elemento del destino.
- *Funcionalidad* de la asociación, que indica si la asociación es o no una función y de qué tipo. Los valores de esta propiedad son los siguientes:
  - "función": si para cualquier tupla de elementos del origen que tiene asociado un elemento del destino, ese elemento es único (es decir, no tiene dos o más elementos asociados del destino).
  - "inyectiva": si es función y, además, dadas dos tuplas diferentes de elementos del origen pertenecientes a la asociación, sus elementos del destino correspondientes son siempre diferentes.
  - "sobreyectiva": si es función y, además, para todo elemento del destino existe al menos una tupla de elementos del origen relacionada.
  - "biyectiva": si es función inyectiva y sobreyectiva
  - "no función": si no se cumplen ninguna de las condiciones anteriores
- *Invariante* de la asociación, que es una condición lógica que deben cumplir todas las instancias de la asociación.

Para especificar cada uno de los participantes de una asociación se ha utilizado el término "rol de asociación", que se define a continuación.

Se define el **universo de roles** (**RL**) como el conjunto de todos los roles de asociación que pueden aparecer en modelos estructurales.

Se define cada **rol de asociación** (**RL_i**) como la especificación de un participante de una asociación (concepto, rol o asociación), incluyendo las restricciones de participación de cada uno de sus elementos.

Los roles se especifican mediante cuatro elementos:

- *Nombre* del rol, que es independiente del nombre del concepto, tipo o asociación participante. Esto es necesario para distinguir roles cuando en una asociación hay varios participantes con el mismo elemento. El nombre de rol debe ser único para todos los roles de una misma asociación.
- *Participante* del rol, que identifica al concepto, tipo o asociación participante en la asociación.
- *Grado* de participación del rol, que restringe el número mínimo de apariciones de cada elemento del participante dentro de la asociación. Puede ser "opcional" (los elementos del participante pueden no aparecer) u "obligatorio" (todos los elementos del participante deben aparecer en la asociación).
- *Límite* de participación del rol, que restringe el número máximo de apariciones de cada elemento del participante dentro de la asociación. Puede ser "único" (los elementos pueden aparecer como mucho una vez) o "múltiple" (los elementos pueden aparecer varias veces en la asociación).

Equipo de fútbol

Seguidamente se recogen las asociaciones existentes:

Nacimiento: asociación entre una persona y su fecha de nacimiento. Toda persona tendrá una y sólo una fecha de nacimiento. Una fecha concreta podrá estar relacionada con cualquier número de personas (incluso ninguno). Características:

- Dimensión = 2.
- Origen: "persona", del concepto "Persona", con grado obligatorio y límite único.
- Destino: "fecha", del concepto "Fecha", con grado opcional y límite múltiple.
- Completitud: "completa".
- Funcionalidad: "función".
- Invariante: no se define.

Pertenencia: asociación entre una persona, una fecha y el equipo al que pertenece esa persona a partir de esa fecha. La fecha podrá participar cualquier número de veces en la asociación. La persona podrá participar cualquier número de veces, siempre que sea en fechas distintas (esto último se debería definir en la invariante de la asociación). Un equipo deberá aparecer al menos una vez en la asociación (sino no es equipo). Características

- Dimensión = 3.
- Origen 1: "persona", del concepto "Persona", con grado opcional y límite múltiple.
- Origen 2: "fecha", del concepto "Fecha", con grado opcional y límite múltiple.
- Destino: "equipo", del concepto "Equipo", con grado obligatorio y límite múltiple.
- Completitud: "parcial".
- Funcionalidad: "sobreyectiva".
- Invariante: si una persona aparece más de una vez, debe ser en fechas diferentes.

Fin de pertenencia: asociación entre la pertenencia de una persona a un equipo y la fecha de final de dicha pertenencia. Esta asociación es de orden superior, ya que uno de sus participantes es una asociación ("Pertenencia"). Cada instancia de la asociación pertenencia podrá aparecer como máximo una vez en la asociación. La fecha podrá participar cualquier número de veces. Características:

- Dimensión = 2.
- Origen: "pertenencia", de la asociación "(Persona, Fecha).Pertenencia", con grado opcional y límite único.
- Destino: "fecha fin", del concepto "Fecha", con grado opcional y límite múltiple.
- Completitud: "parcial".
- Funcionalidad: "función"
- Invariante: no se define.

Convocatoria: asociación entre un equipo, una fecha y su alineación para ese día. Un equipo tendrá al menos una alineación y podrá tener varias posibles. Una fecha podrá participar varias veces en la asociación. Una alineación sólo podrá estar asociada a un equipo en un día concreto. Características:

- Dimensión = 3.
- Origen 1: "equipo", del concepto "Equipo", con grado obligatorio y límite múltiple.
- Origen 2: "fecha", del concepto "Fecha", con grado opcional y límite múltiple.
- Destino: "alineación", del concepto "Alineación", con grado obligatorio y límite único.
- Completitud: "parcial".
- Funcionalidad: "biyectiva".
- Invariante: no se define.

Táctica prevista: asociación entre una alineación y la táctica prevista (esa táctica definirá cuáles serán los puestos de la alineación). Una alineación sólo podrá aparecer una única vez (es decir, sólo tendrá una táctica), mientras que una táctica podrá aparecer cualquier número de veces. Características:

- Dimensión = 2.

- Origen: "alineación", del concepto "Alineación", con grado obligatorio y límite único.
- Destino: "táctica elegida", del concepto "Táctica", con grado opcional y límite múltiple.
- Completitud: "completa".
- Funcionalidad: "función".
- Invariante: no se define.

Puestos: Asociación entre una táctica y sus puestos. Una táctica tendrá que aparecer al menos una vez (en realidad 16 veces). Cada puesto podrá aparecer cualquier número de veces (al menos una). Como invariante habrá que decir que, para cada táctica, deberá haber 11 titulares (uno de ellos portero) y 5 suplentes (uno de ellos también portero). Características

- Dimensión = 2.
- Origen: "táctica", del concepto "Táctica", con grado obligatorio y límite múltiple.
- Destino: "puesto elegido", del concepto "Puesto", con grado obligatorio y límite múltiple.
- Completitud: "completa".
- Funcionalidad: "no función".
- Invariante: no se define.

Asignación: asociación que representa la asignación de un jugador a un puesto en una alineación determinada. El origen de la asociación será el puesto y la alineación, y su destino será el jugador. Una alineación tendrá que aparecer al menos una vez en la asociación (y como máximo 16 veces: 11 titulares y 5 suplentes). Un puesto podrá aparecer cualquier número de veces en la asociación (una vez como máximo por alineación) siempre y cuando esté asignado a la táctica de la alineación (esto forma parte de la invariante). Un jugador podrá aparecer varias veces en la asociación, siempre que sea en alineaciones distintas (esto último se definirá mediante una invariante). Esta asociación tendrá más invariantes que se dejarán para más adelante (por ejemplo, sólo puede haber 3 extranjeros titulares). Características:

- Dimensión: 3.
- Origen 1: "alineación", del concepto "Alineación", con grado obligatorio y límite múltiple.
- Origen 2: "puesto asignado", del concepto "Puesto", con grado opcional y límite múltiple.
- Destino: "jugador", del concepto "Jugador", con grado opcional y límite múltiple.
- Completitud: "parcial".
- Funcionalidad: "función".
- Invariante: El jugador deberá pertenecer al equipo asociado a la alineación **y** un jugador sólo podrá aparecer una vez en cada alineación **y** un puesto sólo podrá aparecer una vez en cada alineación **y** cada puesto deberá pertenecer a la táctica prevista en la alineación **y** no puede haber más de 3 extranjeros titulares.

Bisiesto: asociación entre un año y un valor booleano que indica si el año es bisiesto o no. Características:

- Dimensión: 2.
- Origen 1: "año", del tipo "Año", con grado obligatorio y límite único.
- Destino: "bisiesto", del tipo B, con grado obligatorio y límite múltiple.
- Completitud: "completa".
- Funcionalidad: "sobreyectiva"
- Invariante: un año es bisiesto si es múltiplo de 4 pero no múltiplo de 100, excepto los múltiplos de 400 que siempre son bisiestos.

Seguidamente se recogen **ejemplos de instancias de asociación**:

- **Nacimiento** (en las fechas se va a poner como nombre una representación de la misma).

```
"Nacimiento" = {("Loïc", "16.10.1969"), ...}
```

- **Pertenencia**

```
"Pertenencia" = {("Rivaldo", "1.8.1998" "Barcelona"), ("Figo",
                "1.8.2000", "Real Madrid"), ...}
```

- **Fin de pertenencia**

```
"Fin de Pertenencia" = {(("Rivaldo", "1.8.1998" "Barcelona"),
                       "1.8.2002"), ...}
```

- **Convocatoria**

```
"Convocatoria" = {("Real Madrid", "7.9.2002", "RM Habitual"),
       ("Barcelona", "29.9.2002", "B Especial"), ...}
```

- **Táctica prevista**

```
"Táctica prevista" = {("RM Habitual", "4-4-2"), ("B Especial", "4-3-3"),
                                    ...}
```

- **Puestos**

```
"Puestos" = {("4-4-2", "Portero titular"), ("4-4-2", "Interior derecho
                           titular"), ...}
```

- **Asignación**

```
"Asignación" = {("RM Habitual", "Portero titular", "Casillas"), ("B
     Especial", "Delantero centro titular", "Kluivert"), ...}
```

- **Bisiesto**

```
"Bisiesto" = {(2000, v), (2001, f), (2002, f), ... }
```

4.1.1.5 Atributos de Asociación

Ya se ha comentado en el apartado anterior que las asociaciones son elementos del modelo estructural con el mismo nivel de importancia que los conceptos. En el caso de los conceptos se definían los atributos como correspondencias entre un concepto (conjunto de instancias) y un tipo (conjunto de valores). Pues bien, lo mismo puede hacerse con las asociaciones, ya que una asociación es un conjunto de tuplas.

Los atributos de asociación se utilizan para definir propiedades de las asociaciones, que luego tendrán valor para cada una de sus instancias. Desde el punto de vista matemático un atributo de asociación es una correspondencia entre una asociación (conjunto de tuplas) y un tipo (conjunto de valores).

La Figura 4.10 ilustra gráficamente la definición de atributo de asociación como un conjunto de pares ordenados (*instancia de asociación, valor*). En esa figura se utiliza como ejemplo el atributo "importe" de la asociación "Propiedad", que define el importe de compra de una vivienda cuando la adquirió su propietario. El importe se representará como un número entero.

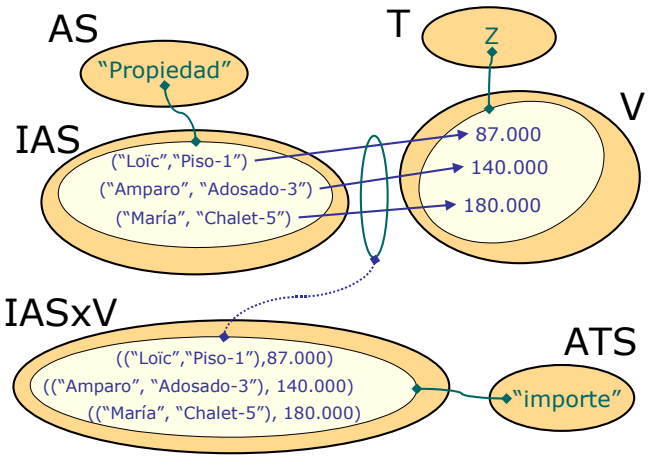


Figura 4.10 Un atributo de asociación relaciona una asociación y un tipo

Los ejemplos de elementos del atributo "importe" de la asociación "Pertenencia" que aparecen en la figura son:

- "Loïc" compró la vivienda "Piso-1" por un importe de 87.000 €
- "Amparo" adquirió "Adosado-3" por un importe de 140.000 €
- "María adquirió "Chalet-5" por un importe de 180.000 €

Una vez visto el concepto intuitivo de atributo de asociación se procede a definir más formalmente los elementos que permiten especifican atributos de asociación.

Se define el **universo de atributos de asociación (ATS)** como el conjunto de todos los atributos de asociación.

Se define cada **atributo de asociación (ATS_i)** como una correspondencia entre una asociación (conjunto de tuplas) y un tipo (conjunto de valores). En otras palabras, un atributo es un subconjunto del producto cartesiano entre el universo de instancias de asociación y el universo de valores. Por lo tanto un atributo es un conjunto de pares ordenados llamados instancias de atributo de asociación.

Se define **universo de instancias de asociación** (**IATS**) como el conjunto de todas las instancias de asociación.

Se define cada **instancia de atributo de asociación ($IATS_i$)** como un par ordenado en el que el primer elemento es una tupla (instancia de asociación) y el segundo elemento es un valor de un tipo. Las instancias de atributo de asociación se crean y se destruyen a lo largo del tiempo.

La Figura 4.11 representa gráficamente las relaciones entre asociaciones, tipos y atributos de asociación.

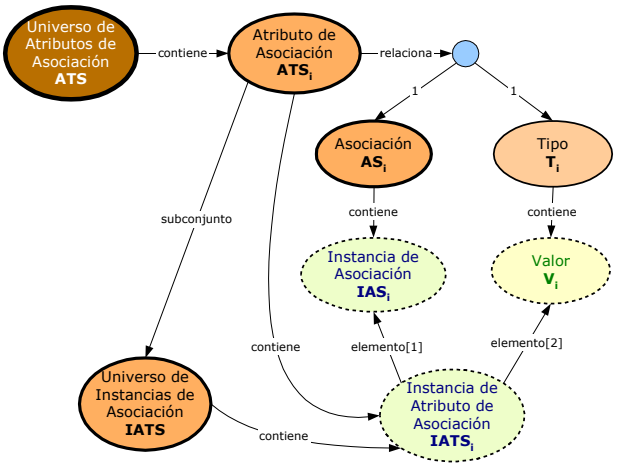


Figura 4.11 Relaciones entre asociaciones, tipos y atributos de asociación

Los atributos de asociación se especifican mediante cinco elementos:

- *Nombre* del atributo de asociación, que debe ser único para todos los atributos de una misma asociación.
- *Asociación* a la que pertenece el atributo. El identificador del atributo se forma uniendo el nombre del atributo y el identificador de la asociación.
- *Tipo* en el que toma valores el atributo.
- *Grado* de participación de la asociación, que restringe el número mínimo de valores que deben estar asignados a cada tupla de la asociación. Puede ser "opcional" (las tuplas pueden no tener valor asignado) u "obligatorio" (todas las tuplas deben tener valor asignado).
- *Límite* de participación de la asociación, que restringe el número máximo de valores que pueden estar asignados a cada tupla de la asociación. Puede ser "único" (las tuplas pueden tener como mucho un valor) o "múltiple" (las tuplas pueden tener más de un valor asignado).
- *Valor por omisión*, que permite asociar un valor a un atributo para que sea utilizado por todas las instancias del concepto que no tengan valor asignado.

Equipo de fútbol

En la asociación "Pertenencia" entre equipo, fecha y persona, se define un atributo "**es socio**" que indica si la persona, además de pertenecer contractualmente al equipo es socio del mismo. Características de este atributo:

- Nombre: "es socio".
- Asociación: "(Persona, Fecha).Pertenencia".
- Tipo: B (Booleano).
- Grado: "obligatorio".
- Límite: "único".
- Valor por omisión: f (falso). Normalmente la persona no es socio del equipo al que pertenece.

Seguidamente se muestran algunas instancias de este atributo de asociación:

```
(Persona,Fecha).Pertenencia.es socio = {((“Rivaldo”, “1.8.1998”
“Barcelona”), f), ((“Figo”, “1.8.2000”, “Real Madrid”), f), ((“Raúl”,
“1.8.1995”, “Real Madrid”), v),  ...}
```

4.1.1.6 Clasificaciones de Concepto

La definición de las clasificaciones se ha divido por claridad en dos partes. En primer lugar se verá su definición y en segundo lugar se tratará el tema de la herencia (simple y múltiple).

4.1.1.6.1 Definiciones de clasificación de concepto

Una clasificación define la relación que existe entre un concepto más general (el concepto padre o “superconcepto”) un conjunto de conceptos más específicos (los conceptos hijos o “subconceptos”), de forma que cada uno de los subconceptos está incluido en el superconcepto. Dicho de otro modo, toda instancia de un subconcepto es también instancia del superconcepto.

La Figura 4.12 muestra gráficamente un ejemplo de clasificación para el concepto “Persona”. En función de su edad se pueden clasificar las personas en “adultos” y “no adultos”. La figura muestra cómo los conceptos “Adulto” y “No adulto” son subconjuntos del concepto “Persona”.

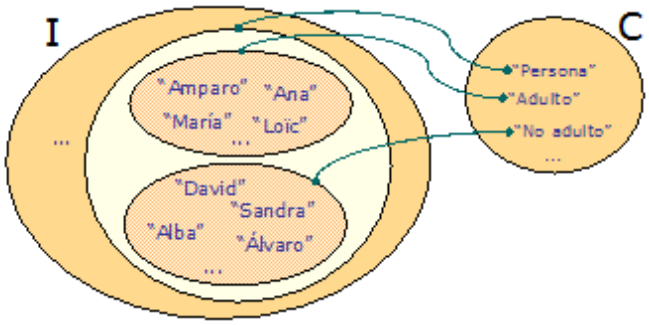


Figura 4.12 Ejemplo de clasificación: las personas son “adultos” o “no adultos”.

Dado que una clasificación no es más que la descomposición de un conjunto (el superconcepto) en subconjuntos (los subconceptos) se pueden producir las siguientes circunstancias:

- Pueden existir varias formas de descomponer un conjunto en subconjuntos, es decir, pueden existir *varias clasificaciones distintas para el mismo superconcepto*. Por ejemplo, las personas también pueden clasificarse en “hombres” y “mujeres” y esta clasificación es independiente de la clasificación por edad. Este hecho hace que sea conveniente identificar cada clasificación con un nombre para poder distinguirlas.
- Puede ocurrir que la intersección de los subconceptos sea vacía o no. En el caso de que la intersección sea vacía se dice que la clasificación es *disjunta*. Los dos ejemplos comentados hasta ahora son clasificaciones disjuntas. Un ejemplo de clasificación no disjunta (o superpuesta) es la clasificación de los “animales” en “cazadores” y “presas”. Algunos animales serán sólo cazadores, otros sólo presas, pero existen animales que, al mismo tiempo, son presas de animales más grandes y cazadores de animales pequeños.

- Puede ocurrir que la unión de los subconceptos sea exactamente el superconcepto o, en el caso contrario, pueden quedar instancias del superconcepto que no pertenezcan a ningún subconcepto. En el primer caso se dice que la clasificación es *completa*. Los dos ejemplos comentados de clasificación de personas son completas. Un ejemplo de clasificación incompleta es, en el ejemplo de un equipo de fútbol, la descomposición de personas en jugadores, técnicos y directivos, ya que podrán existir personas que no pertenezcan a ningún equipo y, por lo tanto, no pertenezcan a ninguno de esos subconceptos.

Por otro lado, las clasificaciones permitirán que los subconceptos, al ser más específicos, definan nuevos atributos o asociaciones. También se podrán definir restricciones en los subconceptos sobre los atributos o asociaciones en las que participa el superconcepto. Estas restricciones reciben el nombre de *especializaciones*, y estarán asociadas a cada uno de los subconceptos de una clasificación.

Una vez descrito el concepto de clasificación, es el momento de pasar a la definición de los elementos que permiten tratar con clasificaciones de concepto.

Se define el **universo de clasificaciones de concepto (CL)** como el conjunto de todas las clasificaciones de concepto que pueden definirse.

Se define cada **clasificación de concepto** (**CL_i**) como la relación que existe entre un concepto más general y otros más específicos. Por ejemplo, el concepto "Persona" puede clasificarse en dos conceptos más específicos "Adulto" y "No adulto". Las clasificaciones son invariantes en el tiempo.

Se denomina **superconcepto** al concepto general de una clasificación. Se denomina **subconcepto** a cada uno de los conceptos específicos de una clasificación.

Cuando se define una clasificación se cumple que los subconceptos son subconjuntos del superconcepto, con características especializadas.

La Figura 4.13 muestra cómo una clasificación relaciona un concepto (el superconcepto) con otros (los subconceptos) y cómo se establece una relación de inclusión entre ellos.

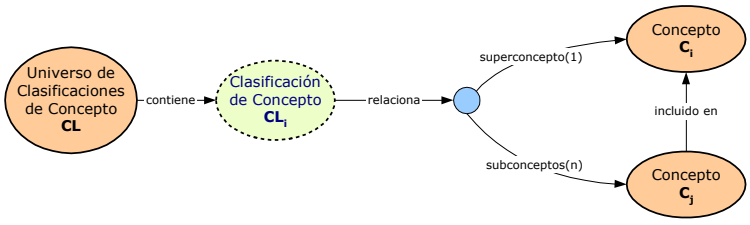


Figura 4.13 Relación entre clasificación de concepto y conceptos

Para especificar una clasificación se utilizan cinco elementos:

- *Nombre* de la clasificación, que permite identificarla cuando hay más de una clasificación para el mismo superconcepto.
- *Superconcepto*, es decir, el concepto más general de la clasificación.
- *Subconceptos* de la clasificación. Cada definición de subconcepto especifica cómo cambia el subconcepto respecto al superconcepto. En relación con los subconceptos se deben cumplir dos condiciones:

1. *Regla de inclusión:* cada subconcepto es un subconjunto del superconcepto.
2. *Regla de no circularidad*: ningún subconcepto debe ser igual al superconcepto o pertenecer al conjunto de sus antecesores.

- *Disjunta*, propiedad que indica si la clasificación es disjunta o no. Si la clasificación no es disjunta, indica que los subconceptos pueden solaparse, es decir, que pueden tener instancias en común. En otro caso, los conjuntos de instancias de los subconceptos serán disjuntos.
- *Completa*, propiedad que indica si la clasificación es completa o no. Si la clasificación es completa indica que la unión de los conjuntos de instancias de los subconceptos es exactamente igual al conjunto de instancias del superconcepto. En otro caso, la unión de los conjuntos de instancias de los subconceptos será un subconjunto del conjunto de instancias del superconcepto.

Antes de seguir con la definición de la especificación de los conceptos, se verán dos ejemplos más de clasificaciones, relacionadas con el ejemplo global de este capítulo (equipo de fútbol).

En primer lugar se muestra un ejemplo de clasificación no disjunta. La Figura 4.14 muestra la clasificación correspondiente a Persona. Una persona puede ser Jugador, Técnico o Directivo (clasificación no disjunta e incompleta). En la figura se muestran instancias que no son personas (como la fecha "fecha1", la táctica "4-4-2" y el equipo "Real Madrid"). Dentro de las personas, las hay que no son ni directivos, ni técnicos ni jugadores (como "Loïc"). Hay otras que son técnicos y jugadores a la vez (como "Vialli").

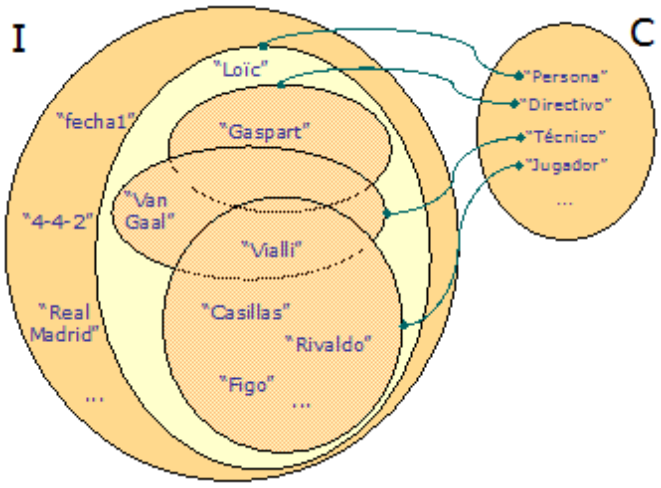


Figura 4.14 Clasificación del concepto Persona en Directivo, Técnico y Jugador

En segundo lugar se muestra un ejemplo en el que hay dos clasificaciones diferentes para un mismo concepto. La Figura 4.15 muestra las dos clasificaciones del concepto Jugador. por posición (Jugador Campo y Portero) o por país de origen (Nacional, Comunitario, Extranjero). Estas dos clasificaciones son disjuntas y completas. Sin embargo, dado que son dos clasificaciones distintas del mismo superconcepto, puede ocurrir que una instancia pertenezca a subconceptos de clasificaciones distintas. Por ejemplo, "Figo" es un "Jugador Campo" y "Comunitario".

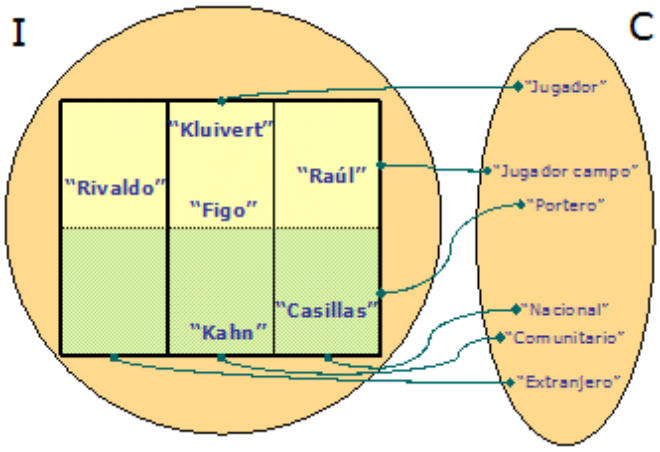


Figura 4.15 Clasificaciones del concepto Jugador: por país y por posición

Queda pendiente definir cómo se representan los subconceptos. Para la especificación de cada uno de los subconceptos se utilizan dos campos:

- *Concepto* que se identifica como subconcepto en la clasificación.
- *Especializaciones*, que son restricciones que impone un subconcepto sobre los atributos del superconcepto y las asociaciones en las que participa el superconcepto. Es importante tener en cuenta que sólo sirven para restringir más y no para ampliar las posibilidades. Existen seis tipos de especialización de subconcepto:
  - *Especialización de grado de participación en asociación* (EGAS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa el superconcepto, el grado de participación en el subconcepto se vuelve más restrictivo (es decir, se convierte en "obligatorio").
  - *Especialización de límite de participación en asociación* (ELAS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa el superconcepto, el límite de participación en el subconcepto se vuelve más restrictivo (es decir, se convierte en "único").
  - *Especialización de grado de atributo* (EGAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su grado se vuelve más restrictivo en el subconcepto (es decir se convierte en "obligatorio").
  - *Especialización de límite de atributo* (ELAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su límite se vuelve más restrictivo en el subconcepto (es decir se convierte en "único").
  - *Especialización de tipo de atributo* (ETAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su tipo se vuelve más restrictivo en el subconcepto. Para ello el tipo definido en la especialización del atributo debe ser un tipo derivado (directa o indirectamente) del tipo definido en el atributo del superconcepto.
  - *Especialización de valor por omisión de atributo* (EVAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, el valor por omisión de ese atributo cambia para las instancias del subconcepto considerado.

Respecto a las especializaciones definidas en un subconcepto, *debe cumplirse que el conjunto de especializaciones asignadas a una definición de subconcepto sea consistente*. Para ello:

- Para cada asociación en la participa el superconcepto no deben existir más de una especialización de grado y de límite para cada rol en el que participe el superconcepto.
- Para cada atributo del superconcepto sólo debe existir una especialización de cada tipo: grado, límite, tipo del atributo y valor por omisión.
- Sólo se podrá especializar aquello que no esté restringido. Por ejemplo, no se podrá especializar el grado de un rol que ya es "obligatorio".

Equipo de fútbol

Una vez definida la clasificación se puede ampliar la especificación de los conceptos de un equipo de fútbol. Lo primero es definir **nuevos conceptos**:

- **Técnico**: persona que desempeña el papel de técnico de un equipo.
- **Directivo**: persona que desempeña el papel de directivo en un equipo.
- **Nacional**: un tipo de jugador que se caracteriza porque su país de origen es el país donde se celebran las competiciones (en este caso es España).
- **Comunitario**: un tipo de jugador que se caracteriza porque su país de origen pertenece a la comunidad europea.
- **Extranjero**: un tipo de jugador que se caracteriza porque su país de origen no está en la comunidad europea.
- **Portero**: un jugador cuya posición es la de portero. Los porteros tienen como nuevos atributos evaluados sus reflejos, juego con el pie, paradas por alto, paradas por bajo y salidas.
- **Jugador campo**: un jugador que no juega de portero. Los jugadores de campo tienen como nuevos atributos su capacidad de cortar balones, de centrar, de regatear, de rematar con el pie y de rematar con la cabeza.
- **Titular**: un tipo de puesto que se corresponde con uno de los titulares de la alineación.
- **Suplente**: un tipo de puesto que se corresponde con uno de los suplentes de la alineación.

A partir de estos nuevos conceptos se pueden definir las siguientes **clasificaciones de concepto**:

- **Personas por ocupación**: Hay tres tipos de personas, en función de su ocupación en el equipo. El superconcepto es "Persona" y los subconceptos son "Técnico", "Directivo" y "Jugador". Esta clasificación es no disjunta e incompleta. En esta clasificación se puede definir una *especialización de grado de asociación*: todo jugador debe pertenecer obligatoriamente a un equipo para ser considerado como tal. Para ello hay que especializar el grado del rol "persona" de la asociación "Pertenencia":
- **Jugador por país**: hay tres tipos de jugadores, en función de su nacionalidad. El superconcepto es "Jugador" y los subconceptos son "Nacional", "Comunitario" y "Extranjero". Esta clasificación es disjunta y completa. Aquí se especializa el atributo "país" para los tres subconceptos: "Nacional" usará el tipo "País nacional", "Comunitario" el tipo "País comunitario" y "Extranjero" el tipo "País extranjero".

- **Jugador por posición**: Hay dos tipos de jugadores en función de su posición en el campo. El superconcepto es "Jugador" y los subconceptos son "Portero" y "Jugador campo". Esta clasificación es disjunta y completa. Aquí se especializa el atributo "posición" para los dos subconceptos, asignando el tipo "Posición portero" al concepto "Portero" y el tipo "Posición campo" al concepto "Jugador campo".
- **Puesto por titularidad**. Hay dos tipos de puestos dentro de una táctica. El superconcepto es "Puesto" y los subconceptos son "Titular" y "Suplente".

En cuanto a las instancias en los conceptos "Persona", "Jugador", "Técnico", "Directivo", "Portero" y "Jugador campo", y teniendo en cuenta las clasificaciones anteriores, se podrán tener los siguientes casos:

- Una persona que sea jugador y técnico a la vez:

"Viali"∈"Jugador" y "Viali"∈"Técnico"

- Un jugador que sea jugador de campo y extranjero:

"Rivaldo"∈"Jugador Campo" y "Rivaldo"∈"Extranjero"

- Un jugador que sea portero y nacional:

"Casillas"∈"Portero" y "Casillas"∈"Nacional"

En cambio no podrá ocurrir que un jugador sea portero y de campo simultáneamente, así como tampoco podrá ser nacional y extranjero a la vez.

4.1.1.6.2 Herencia simple y múltiple

A partir de las definiciones anteriores *se puede deducir de forma natural la herencia de atributos y asociaciones*. Para ello la idea fundamental es que, dada una clasificación, cualquier subconcepto es un subconjunto del superconcepto. Esto quiere decir que toda instancia de un subconcepto es también instancia del superconcepto.

Sea I_k una instancia del subconcepto de una clasificación. Entonces, dado que I_k pertenece también al superconcepto:

- I_k puede aparecer en las instancias de asociación (n-tuplas) de cualquier asociación en la que participe el superconcepto. Por lo tanto *el subconcepto hereda la participación en asociaciones*.
- I_k puede aparecer en las instancias de cualquier atributo definido para el superconcepto. Por lo tanto *el subconcepto hereda las definiciones de atributos del superconcepto*.

Esta herencia de asociaciones y atributos se realiza de forma que los subconceptos pueden especializar la participación de asociaciones y la definición de atributos, tal y como se ha visto en las definiciones anteriores. Esto es así ya que un subconcepto es más especializado que el superconcepto y se pueden restringir más esas definiciones.

Equipo de fútbol

Como ejemplo se va a aplicar la herencia simple al concepto "Jugador Campo" teniendo en cuenta las clasificaciones definidas más arriba. Un "Jugador Campo" es un subconcepto de "Jugador" que, a su vez, es un subconcepto de "Persona".

Atributos

- Definidos en el concepto "Jugador campo"

"centros", "regate", "remate cabeza", "remate pie", "robos"

- Heredados desde el concepto "Jugador"

"estado", "dorsal", "fuerza", "posición", "resistencia", "velocidad"

- Heredados desde el concepto "Persona"

"nombre", "país"

Participación en asociaciones

- Definidas para el concepto "Jugador campo"

-

- Heredadas desde el concepto "Jugador"

Rol "jugador" en "Asignación"

- Heredadas desde el concepto "Persona"

Rol "persona" en "Nacimiento"

Rol "persona" en "Pertenencia"

La Figura 4.16 muestra gráficamente cómo se produce la herencia simple.

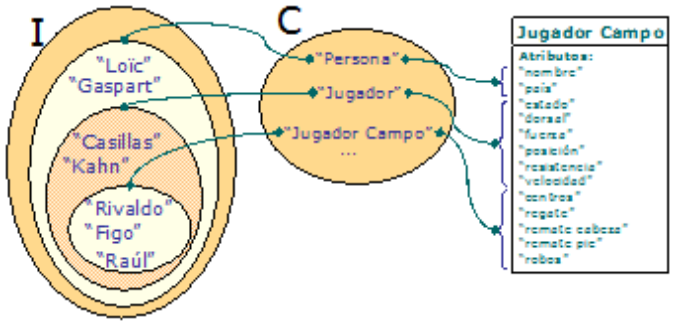


Figura 4.16 Herencia simple de atributos en el concepto "Jugador Campo"

A la izquierda de la imagen aparece la inclusión de unos conceptos dentro de otros ("Jugador campo" está incluido en "Jugador", que a su vez está incluido en "Persona".

A la derecha de la imagen se muestran todos los atributos del concepto "Jugador Campo", indicándose de dónde procede cada uno.

Por otro lado, tal y como se han definido las clasificaciones, puede ocurrir que exista un concepto que sea subconcepto en varias clasificaciones diferentes (siempre y cuando no incumpla las reglas de inclusión y de no circularidad). Dicho de otra forma, ese concepto tendría varios padres. Esto tiene varias implicaciones.

Se dice que existe *herencia múltiple* si existe al menos un concepto tal que su conjunto de padres tiene más de un elemento. En ese caso se debe cumplir que la intersección de todos los padres de ese concepto no sea nula.

Cuando existe herencia múltiple se pueden producir ambigüedades en la herencia de asociaciones y atributos. Ninguna de las técnicas existentes en la bibliografía para tratar la herencia múltiple resuelve completamente las ambigüedades (ver [Gómez et al., 1997]), por lo que será misión del analista su resolución según va completando un modelo.

Equipo de fútbol

Supóngase que, para poder trabajar con la selección nacional de fútbol, interesa clasificar el concepto "Nacional" en dos subconceptos: "Jugador Campo Nacional" y "Portero Nacional":

```
CL("por posición", "Nacional", {("Jugador Campo Nacional", ∅), ("Portero Nacional, ∅)}, "exclusiva", "exhaustiva")
```

Estos dos conceptos podrían ser, a su vez, subconceptos de "Jugador Campo" y "Portero", respectivamente:

```
CL("por país", "Jugador Campo", {("Jugador Campo Nacional", ∅)}, "exclusiva", "incompleta")

CL("por país", "Portero", {("Portero Nacional", ∅)}, "exclusiva", "incompleta")
```

En este caso se tendría herencia múltiple:

- El concepto "Jugador Campo Nacional" es subconcepto de "Jugador Campo" y "Nacional".
- El concepto "Portero Nacional" es subconcepto de "Portero" y "Nacional"

La Figura 4.17 muestra gráficamente el ejemplo anterior. El concepto "Jugador" se ha clasificado horizontalmente por la posición en "Jugador Campo" (arriba) y "Portero" (abajo) y se ha clasificado verticalmente por el país en "Nacional", "Comunitario" y "Extranjero" (De derecha a izquierda). El rectángulo azul se corresponde con el concepto "Jugador Campo Nacional" y el rectángulo verde más intenso se corresponde con el concepto "Portero Nacional".

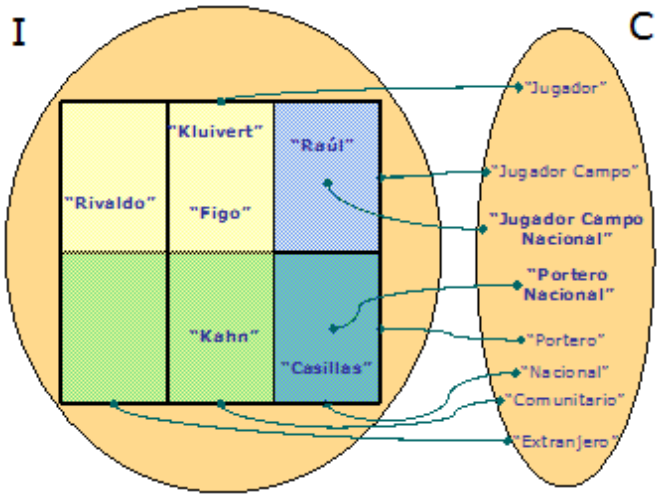


Figura 4.17 Ejemplo de clasificaciones con herencia múltiple

Por último conviene comentar cómo afectan las clasificaciones a la pertenencia de instancias a conceptos. Con la definición dada de clasificación, *puede ocurrir que una instancia pertenezca a varios conceptos de forma simultánea*, sin que estos conceptos tengan una relación superconcepto – subconcepto directa.

Existen dos posibilidades:

- Cuando se tiene una *clasificación no disjunta* una instancia puede pertenecer a varios de los subconceptos. (Por ejemplo la clasificación de los animales en cazadores y presas)

Equipo de fútbol

Aquí el ejemplo es una persona que puede ser al mismo tiempo "Técnico" y "Jugador" (o "Directivo" y "Técnico").

- Cuando se tienen *varias clasificaciones para el mismo superconcepto*, suele ocurrir que una instancia pertenezca a subconceptos de cada una de las clasificaciones. (Por ejemplo, al clasificar personas por edad, y por sexo, una instancia puede ser joven [edad] y mujer [sexo] al mismo tiempo).

Equipo de fútbol

Aquí el ejemplo es una persona que puede ser al mismo tiempo "Jugador campo" y "Nacional" (o "Portero" y "Extranjero"").

La única restricción que existe es que una instancia sólo podrá pertenecer a dos o más conceptos si estos conceptos tienen un antecesor común.

4.1.1.7 Clasificaciones de Asociación

Dado que las asociaciones son también conjuntos de elementos, en el modelo estructural también se puede representar que el conjunto de instancias de una asociación está incluido dentro del conjunto de instancias de otra.

Cada clasificación de asociación define la relación que existe entre una asociación más general (llamada asociación padre o "superasociación") y un conjunto de asociaciones más específicas (llamadas asociaciones hijas o "subasociaciones"), de forma que cada una de las subasociaciones está incluida en la superasociación. Dicho de otro modo, toda instancia de una subasociación es también una instancia de la superasociación.

La Figura 4.18 muestra gráficamente un ejemplo de clasificación para la asociación "Contrato", establecida entre los conceptos "Persona" y "Empresa". En función de su temporalidad se pueden clasificar los contratos en temporales e indefinidos. La figura muestra cómo las asociaciones "Contrato temporal" y "Contrato indefinido" son subconjuntos de la asociación "Contrato".

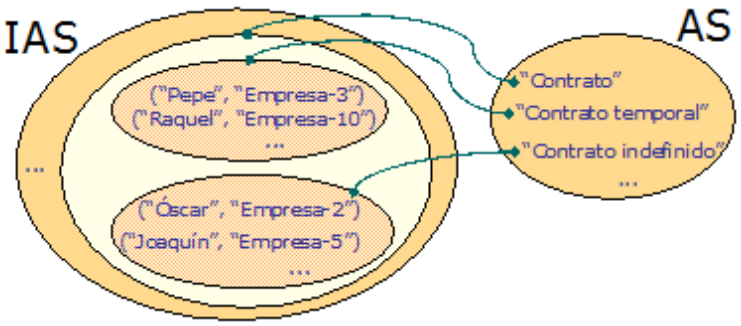


Figura 4.18 Ejemplo de clasificación de asociación: tipos de contratos.

En esta figura se muestra cómo los contratos de "Pepe" con la empresa "Empresa-3" y de "Raquel" con la "Empresa-10" son temporales mientras que los contratos de "Óscar" con "Empresa-2" y de "Joaquín" con "Empresa-5" son indefinidos.

Del mismo modo que con las clasificaciones de concepto, se pueden producir varias circunstancias:

- Pueden existir varias formas de descomponer un conjunto en subconjuntos, es decir, *pueden existir varias clasificaciones distintas para la misma superasociación*.
- Puede ocurrir que la intersección de las subasociaciones sea vacía o no. En el caso de que la intersección sea vacía se dice que la clasificación es *disjunta*.
- Puede ocurrir que la unión de las subasociación sea exactamente igual a la superasociación o, en el caso contrario, pueden quedar instancias de la superasociación que no pertenezcan a ninguna subasociación. En el primer caso se dice que la clasificación de asociación es *completa*.

Por otro lado, las clasificaciones de asociación permitirán que las subasociaciones, al ser más específicas, definan nuevos atributos o asociaciones. También se podrán definir restricciones en las subasociaciones sobre los atributos o asociaciones en las que participa la superasociación. Estas restricciones reciben el nombre de *especializaciones*, y estarán asociadas a cada una de las subasociaciones de una clasificación.

Una vez descrito el término de clasificación de asociación, es el momento de pasar a la definición de los elementos que permiten tratarlas en el modelo estructural.

Se define el **universo de clasificaciones de asociación (CLS)** como el conjunto de todas las clasificaciones de asociación que pueden definirse.

Se define cada **clasificación de asociación** (**CLS_i**) como la relación que existe entre una asociación más general y otras más específicas. Por ejemplo, la relación "Contrato" entre "Persona" y "Empresa" puede clasificarse en "Contrato temporal" y "Contrato indefinido". Se debe cumplir que las asociaciones más específicas (subasociaciones) sean subconjuntos de la asociación general (superasociación). Las clasificaciones de asociación son invariantes en el tiempo.

Se denomina **superasociación** a la asociación general de una clasificación. Se denomina **subasociación** a cada una de las asociaciones específicas de una clasificación.

Cuando se define una clasificación de asociación se cumple que las subasociaciones son subconjuntos de la superasociación, con características especializadas.

La Figura 4.19 muestra cómo una clasificación de asociación relaciona una asociación (la superasociación) con otras (las subasociaciones) y cómo se establece una relación de inclusión entre ellas.

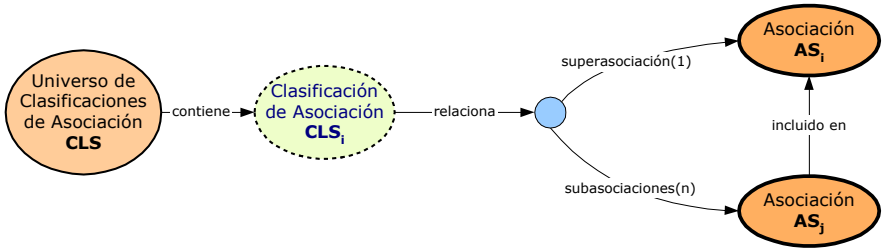


Figura 4.19 Relación entre clasificaciones de asociación y las asociaciones

Para especificar una clasificación de asociación se utilizan cinco elementos muy similares a los de clasificación de concepto:

- *Nombre* de la clasificación, que permite identificarla cuando hay más de una clasificación para el mismo superconcepto.
- *Superasociación* de la clasificación.
- *Subasociaciones* de la clasificación. Se deben cumplir las mismas dos condiciones que en las clasificaciones de concepto: regla de inclusión y regla de no circularidad.
- *Disjunta*.
- *Completa*.

Las subasociaciones se especifican de forma similar a los subconceptos, mediante dos campos:

- *Asociación* que se identifica como subasociación en la clasificación.
- *Especializaciones*, que son restricciones que impone una subasociación sobre los atributos de la superasociación y las asociaciones en las que participa la superasociación. Es importante tener en cuenta que sólo sirven para restringir más y no para ampliar las posibilidades. También existen seis tipos de especialización de subasociación:
  - *Especialización de grado de participación en asociación* (EGASS).
  - *Especialización de límite de participación en asociación* (ELASS).
  - *Especialización de grado de atributo* (EGATS).
  - *Especialización de límite de atributo* (ELATS).
  - *Especialización de tipo de atributo* (ETATS).
  - *Especialización de valor por omisión de atributo* (EVAT).

Respecto a las especializaciones definidas en una subasociación, *debe cumplirse que el conjunto de especializaciones asignadas a una definición de subasociación sea consistente*. Para ello:

- Para cada asociación en la participa la superasociación no deben existir más de una especialización de grado y de límite para cada rol en el que participe la superasociación.
- Para cada atributo de la superasociación sólo debe existir una especialización de cada tipo: grado, límite, tipo del atributo y valor por omisión.
- Sólo se podrá especializar aquello que no esté restringido. Por ejemplo, no se podrá especializar el grado de un rol que ya es "obligatorio".

Equipo de fútbol

Una vez definida la clasificación de asociaciones se puede ampliar la especificación de las asociaciones de un equipo de fútbol.

En primer lugar se definen **nuevas asociaciones**:

- **Fichaje**: asociación con origen "Jugador" y "Fecha" y con destino "Equipo" que representa el contrato entre un jugador y un equipo a partir de una fecha. En esta asociación se definen dos nuevos atributos: el importe de la ficha y el importe de la cláusula de rescisión.

- **Entrenador**: asociación con origen "Técnico" y "Fecha" y con destino "Equipo" que identifica al técnico que ejerce de entrenador a partir de una fecha. En esta asociación se definen dos nuevos atributos: el sueldo y el objetivo planteado al contratar al entrenador.
- **Presidente**: asociación con origen "Directivo" y "Fecha" y con destino "Equipo" que identifica al directivo que ejerce de presidente a partir de una fecha. Esta asociación tiene un atributo para indicar si el presidente es, además, propietario del equipo (con valor por omisión "falso").
- **Titulares**: asociación con origen "Táctica" y con destino "Titular" que identifica los puestos titulares de una táctica.
- **Suplentes**: asociación con origen "Táctica" y con destino "Suplente" que identifica los puestos suplentes de una táctica.

Tras estas definiciones, puede procederse a definir **clasificaciones de asociación**:

- **Pertenencia por papel**. Hay tres tipos de pertenencia de personas a equipos, en función de su papel: Fichaje, Entrenador y Presidente. Esta clasificación es no disjunta e incompleta.
- **Puestos por titularidad**. Hay dos tipos de asociaciones entre tácticas y puestos, en función de su titularidad: titulares y suplentes. Esta clasificación es disjunta y completa.

En cuanto a las instancias en las asociaciones "Pertenencia", "Fichaje", "Entrenador" y "Presidente", y teniendo en cuenta la clasificación anterior, se podrán tener los siguientes casos:

- Si una persona es jugador y entrenador a la vez, la tupla correspondiente pertenecerá simultáneamente a las asociaciones "Fichaje" y "Entrenador":

("Vialli", "1.5.1990", "Arsenal")∈"Fichaje" y ("Vialli", "1.5.1990", "Arsenal")∈"Entrenador"

- Una persona puede ser directivo de un equipo pero no ser el presidente:

("Valdano", "1.1.2000", "Real Madrid")∈"Pertenencia" y ("Valdano", "1.1.2000", "Real Madrid")∉"Presidente"

La Figura 4.20 muestra la clasificación correspondiente a Pertenencia. Una relación de pertenencia de una persona a un equipo en una fecha dada puede ser un Fichaje (si la persona es Jugador), la designación de esa persona como Entrenador (si es un Técnico) o la elección de esa persona como Presidente (si es Directivo). Es una clasificación no disjunta e incompleta.

En la figura se muestran instancias de la asociación Pertenencia que no son fichajes, ni entrenadores, ni presidentes (como la pertenencia de "Valdano" desde el 1 de enero de 2000 al Real Madrid como director deportivo). También hay otras pertenencias que son instancias de Fichaje y Entrenador a la vez (como la pertenencia de "Vialli" desde el 1 de mayo de 1990 al Arsenal).

De la misma forma que ocurre en el caso de las clasificaciones de concepto, en las clasificaciones de asociación puede hablarse de herencia simple, herencia múltiple y pertenencia de una tupla a varias asociaciones diferentes.

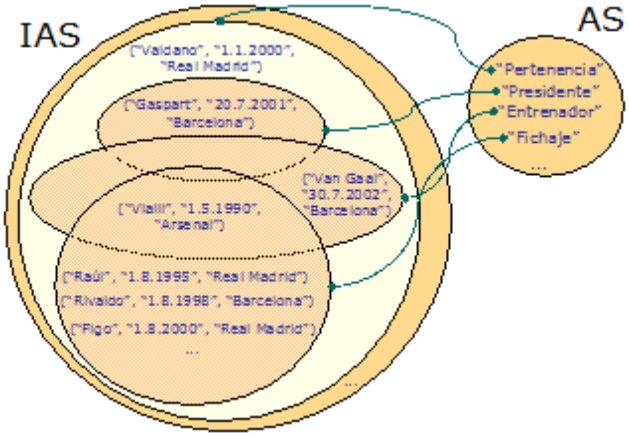


Figura 4.20 Clasificación de Pertenencia en Fichaje, Entrenador y Presidente.

Equipo de fútbol

Como ejemplo se va a aplicar la herencia simple a la asociación "Fichaje" teniendo en cuenta la clasificación definida más arriba. Un "Fichaje" es una subasociación de "Pertenencia".

Atributos de asociación

- Definidos en la asociación "Fichaje"

"ficha", "rescisión"

- Heredados desde la asociación "Pertenencia"

"es socio"

Participación en asociaciones

- Definidas para la asociación "Fichaje"

-

- Heredadas desde la asociación "Pertenencia"

Rol "pertenencia" en "Fin de pertenencia"

La Figura 4.21 muestra gráficamente cómo se produce la herencia. A la izquierda de la imagen aparece la inclusión de unas asociaciones dentro de otras. A la derecha de la imagen se muestran todos los atributos de la asociación "Fichaje", indicándose de dónde procede cada uno.

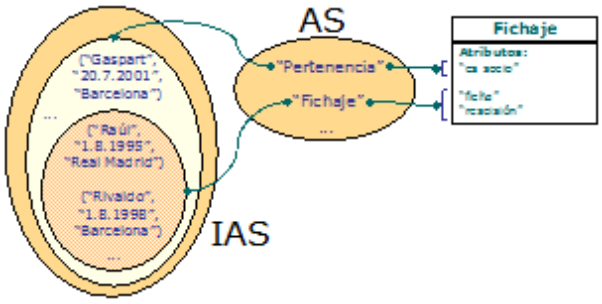


Figura 4.21 Herencia simple de atributos en la asociación "Fichaje"

4.1.1.8 Estructura y Estado

Un modelo estructural se puede considerar dividido en dos partes bien diferenciadas: estructura y estado.

Se define la **estructura** de un modelo estructural como la colección de todos los elementos no dependen del tiempo: definición de tipos (y restricciones sobre sus valores), conceptos, asociaciones, atributos (de concepto y asociación) y clasificaciones (de concepto y asociación). Todas estas definiciones imponen restricciones sobre los estados posibles del sistema.

La Figura 4.22 recoge los elementos principales que definen la estructura.

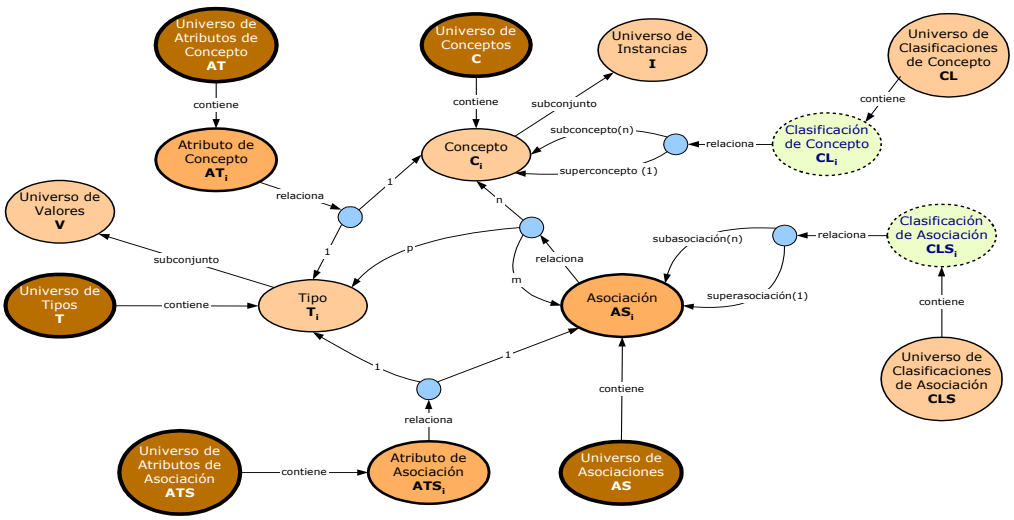


Figura 4.22 Elementos que definen la estructura del dominio

Por otro lado está el **estado**, que está compuesto por aquellos elementos del modelo estructural cuya existencia depende del tiempo. Estos elementos son:

- Instancias de concepto.
- La relación de pertenencia de una instancia a un concepto concreto (que puede tener un tiempo de vida diferente que la instancia: una instancia puede pertenecer a distintos conceptos a lo largo del tiempo).
- Instancias de atributo.
- Instancias de asociación o tuplas.
- La relación de pertenencia de una instancia de asociación a una asociación concreta (que puede tener un tiempo de vida diferente que la tupla: una instancia puede pertenecer a distintas asociaciones a lo largo del tiempo).
- Instancias de atributo de asociación

La Figura 4.23 recoge los elementos que definen el estado.

Para todos los elementos anteriores se va a representar el **paso del tiempo** mediante dos valores (que son números naturales):

- **Instante de creación** (tc): representa el instante de tiempo en el cual se ha creado el elemento correspondiente. Si el elemento se crea al arrancar el sistema, su instante de creación será 0.
- **Instante de destrucción** (td): representa el instante de tiempo en el cual se ha destruido el elemento. Si un elemento todavía no ha sido destruido, su instante de destrucción será ∞.

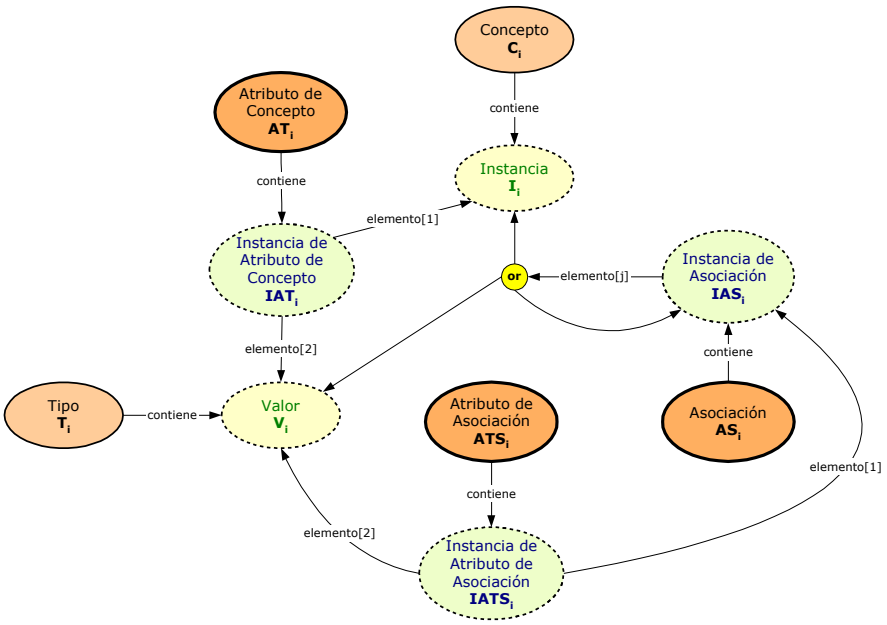


Figura 4.23 Elementos que definen el estado en un momento dado

Partiendo de estas definiciones de tiempo de vida, se puede definir el **estado global de un modelo estructural en un instante t (ST_{EST})** como las instancias de concepto, de atributo de concepto, de asociación y de atributo de asociación cuyo tiempo de vida incluye el valor t. Es decir, aquellos elementos tales que tc sea menor o igual que t y que td sea mayor o igual que t.

Se define por tanto el **universo de estados** (ST) como el conjunto de todos los estados válidos de un sistema (para cualquier instante de tiempo t).

A partir de estas definición de estado global se pueden definir los **estados parciales** de concepto, atributo, asociación y atributo de asociación:

- **Estado de un concepto en un instante t**: conjunto de instancias del concepto cuyo tiempo de creación es menor o igual que t y cuyo tiempo de destrucción no está definido o bien es mayor o igual que t.
- **Estado de un atributo de concepto en un instante t**: conjunto de instancias del atributo de concepto cuyo tiempo de creación es menor o igual que t y cuyo tiempo de destrucción no está definido o bien es mayor o igual que t.
- **Estado de una asociación en un instante t**: conjunto de tuplas de la asociación cuyo tiempo de creación es menor o igual que t y cuyo tiempo de destrucción no está definido o bien es mayor o igual que t.
- **Estado de un atributo de asociación en un instante t**: conjunto de instancias del atributo de concepto cuyo tiempo de creación es menor o igual que t y cuyo tiempo de destrucción no está definido o bien es mayor o igual que t.

Después de estas definiciones puede describirse **cómo se representa un estado** para un modelo estructural.

El estado global del sistema es una tupla formada por cuatro elementos: estado de todos los conceptos, estado de todos los atributos de concepto, estado de todas las asociaciones y estado de todos los atributos de asociación.

- El estado de todos los conceptos es un conjunto, de forma que cada uno de sus elementos representa el estado de uno de los conceptos del modelo estructural.
    - El estado de cada concepto es a su vez un conjunto que contiene las instancias que existen en el instante de tiempo considerado.
- El estado de todos los atributos de concepto es un conjunto, de forma cada uno de sus elementos representa el estado de cada uno de los atributos de concepto del modelo estructural.
    - El estado de cada atributo de concepto es un conjunto formado por las instancias del atributo (pares ordenados) que existen en el instante de tiempo considerado.
- El estado de todas las asociaciones es un conjunto, de forma cada uno de sus elementos representa el estado de cada una de las asociaciones del modelo estructural.
    - El estado de cada asociación es un conjunto formado por las instancias de la asociación (tuplas) que existen en el instante de tiempo considerado.
- El estado de todos los atributos de asociación es un conjunto, de forma cada uno de sus elementos representa el estado de cada uno de los atributos de asociación del modelo estructural.
    - El estado de cada atributo de concepto es un conjunto formado por las instancias del atributo (pares ordenados) que existen en el instante de tiempo considerado.

La Figura 4.24 resume gráficamente la estructura de un estado en un instante t.

Sobre el estado de un sistema existen una serie de **restricciones** que afectan a los tiempos de vida de algunos elementos en relación a otros:

- El tiempo de vida de una instancia de atributo de concepto tiene que estar incluido en el tiempo de vida de la instancia de concepto correspondiente. En otras palabras:
    - El tiempo de creación de la instancia de atributo tiene que ser igual o superior al tiempo de creación de la instancia de concepto.
    - El tiempo de destrucción de la instancia de atributo tiene que ser igual o inferior al tiempo de destrucción de la instancia de concepto.
- El tiempo de vida de una tupla (instancia de asociación) tiene que estar incluido en el tiempo de vida de todas los participantes en la tupla (exceptuando los valores de tipo, que existen siempre). En otras palabras:
    - El tiempo de creación de la tupla tiene que ser igual o superior al máximo de los tiempos de creación máximo de los elementos de la tupla.
    - El tiempo de destrucción de la tupla tiene que ser igual o inferior al mínimo de los tiempos de destrucción de los elementos de la tupla.

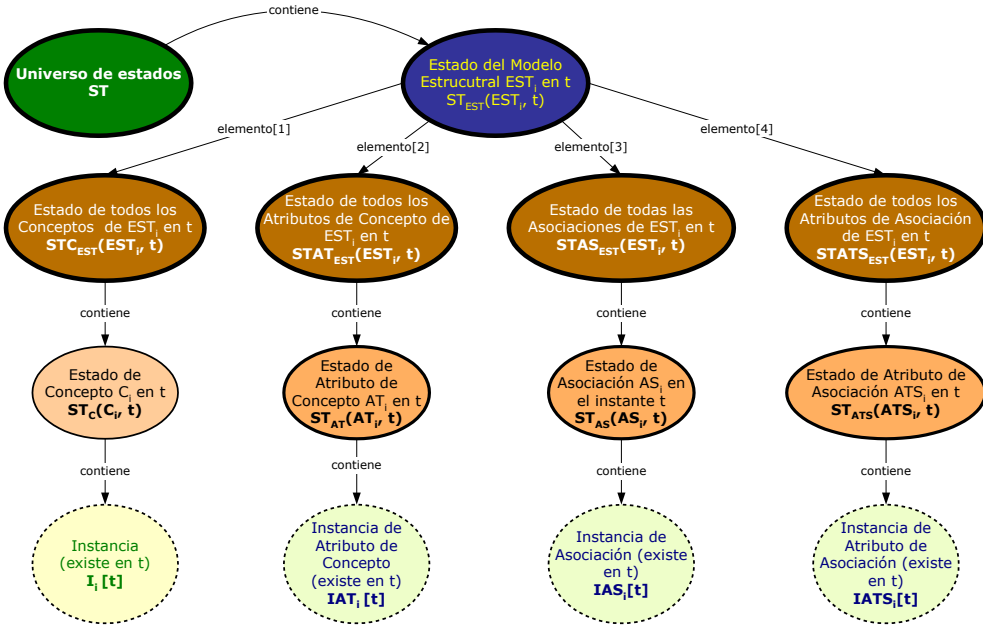


Figura 4.24 El estado de un modelo estructural en un instante t

- El tiempo de vida de una instancia de atributo de asociación tiene que estar incluido en el tiempo de vida de la tupla (instancia de asociación) correspondiente. En otras palabras:
  - El tiempo de creación de la instancia de atributo de asociación tiene que ser igual o superior al tiempo de creación de la tupla.
  - El tiempo de destrucción de la instancia de atributo de asociación tiene que ser igual o inferior al tiempo de destrucción de la tupla.

4.1.2 Modelo de Comportamiento

El modelo de comportamiento está formado por dos conjuntos: tareas y métodos de tarea. La definición de estos elementos utiliza dos conjuntos más que son predefinidos y existen siempre: operadores y funciones de consulta. Todos estos elementos se definirán en los apartados siguientes.

Equipo de fútbol

Respecto al modelo de comportamiento del equipo de fútbol, se van a considerar como ejemplos tres tareas de alto nivel:

- **Mejorar la fuerza de un jugador en un incremento determinado**. Se proporciona el jugador y el incremento de fuerza deseado. El objetivo será elegir la mejor forma de entrenar al jugador para que mejore su fuerza en esa cantidad.
- **Crear un jugador de campo desde cero**, proporcionando todos sus datos: (1) datos como persona: nombre completo, país de origen y fecha de nacimiento (2) datos como jugador: posición, dorsal, fuerza, velocidad, resistencia, estado, equipo al que pertenece, fecha de comienzo de la pertenencia (fichaje), importe de la ficha, cláusula de rescisión (3) datos como jugador de campo: valoración de robos, centros, regate, remate con el pie y remate con la cabeza.

- **Crear la alineación de un equipo** para un partido concreto. Para ello habrá que elegir una táctica y luego rellenar los puestos con los mejores jugadores disponibles.

4.1.2.1 Tareas

Se define el comportamiento de un sistema como su evolución a lo largo del tiempo, partiendo de un estado inicial (estado problema) hasta llegar a un estado final (estado solución). Los elementos básicos que permiten especificar el comportamiento de un sistema son las tareas que permiten reflejar transiciones entre estados.

Partiendo de esta idea, se puede representar la historia del comportamiento de un sistema cuando se ha resuelto un problema como un grafo en el que los nodos son estados y los arcos son instancias concretas de tareas, tal y como se muestra en la Figura 4.25. Bajo este punto de vista las tareas definen correspondencias entre estados.

Figura 4.25 Historia de la resolución de un problema

Se define el **universo de Tareas (TA)** como el conjunto de todas las tareas que pueden definirse.

Se define cada **tarea** (**TA_i**) como una correspondencia entre el universo de estados y el universo de estados. Por lo tanto una tarea es un conjunto de pares ordenados llamados instancias de tarea.

Se define el **universo de instancias de tarea (ITA)** como el conjunto de todas las instancias de tarea posibles.

Se define **instancia de tarea (ITA_i)** como un par ordenado formado por dos estados, que representa la ejecución de una tarea partiendo de un estado concreto en un instante determinado. El primer estado se denomina **estado inicial** de la instancia de tarea y el segundo estado se denomina **estado final** de la instancia de tarea. Es importante no olvidar que cada uno de estos dos estados tiene asociado un instante de tiempo.

La Figura 4.26 muestra de forma gráfica las relaciones existentes entre tareas y estados.

La forma de definir esa correspondencia entre estados consiste en definir dos condiciones:

- Primero, la condición que debe cumplir un estado para que pueda aparecer en alguna tupla de la tarea como estado inicial. Esta condición se denomina *precondición*.
- En segundo lugar, la condición que debe cumplir un estado para que pueda aparecer en alguna tupla de la tarea como estado final, conociéndose el estado inicial. Esta condición se denomina *postcondición* y generalmente define las propiedades del estado final de forma relativa al estado inicial.

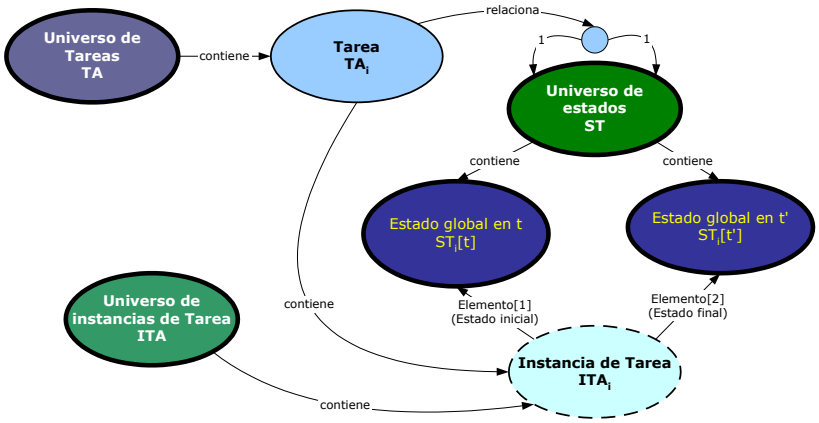


Figura 4.26 Relaciones entre tareas y estados

En resumen una tarea no es más que una descripción declarativa de un problema: dice qué condiciones deben cumplirse en el estado inicial y final pero no dice cómo resolver el problema.

Para especificar una tarea se requieren cinco elementos:

- *Nombre* de la tarea, que debe ser único dentro de un modelo de comportamiento.
- *Entrada* de la tarea, que contiene una serie de variables sin cuantificar que forman la entrada de la tarea (variables cuya asignación debe ser conocida antes de poder evaluar la precondición de la tarea).
- *Precondición* de la tarea: condición lógica que deberá ser cierta en el estado inicial para poder llevar a cabo la tarea.
- *Salida* de la tarea: conjunto de variables sin cuantificar cuyo valor deberá ser conocido después de haber realizado la tarea.
- *Postcondición* de la tarea: condición binaria que deberá ser cierta al terminar la tarea, comparando los estados inicial y final.

Las variables y condiciones lógicas se definen en el apartado 4.1.3.

Una tarea del modelo de comportamiento equivale a la definición formal de un problema de la siguiente forma:

- SEAN:
  - un instante de tiempo inicial t_i.
  - una asignación de valores concretos a las variables de la entrada en ese instante de tiempo,
  - tales que la evaluación de la precondición de la tarea en ese instante de tiempo y con esa asignación de variables es cierta
- ENTONCES:
  - existe un tiempo final t_f.
  - y existe una asignación de valores a las variables de salida,
  - tales que la evaluación de la postcondición de la tarea en los instantes inicial y final y con la unión entre las dos asignaciones de sea cierta.

Equipo de fútbol

Se muestran tres ejemplos de tareas.

(1) Mejorar la fuerza de un jugador en un incremento determinado. Se proporciona el jugador y su incremento de fuerza.

- Nombre: "Mejorar fuerza con entrenamiento".
- Entrada 1: un jugador (instancia del concepto "Jugador").
- Entrada 2: un incremento de fuerza (un número real, valor de R).
- Precondición: la fuerza del jugador debe poder incrementarse (es decir que es menor que 10) y el incremento solicitado es mayor que cero y la suma de la fuerza actual más el incremento es menor o igual que 10.
- Postcondición: la fuerza del jugador en el estado final debe ser igual que la fuerza en el estado inicial más el incremento solicitado.

(2) Crear un jugador de campo desde cero, proporcionando todos sus datos (nombre completo, país de origen, fecha de nacimiento, posición, dorsal, fuerza, velocidad, resistencia, nombre del equipo al que pertenece, fecha de comienzo del fichaje, importe de la ficha, cláusula de rescisión, valoración de robos, centros, regate, remate con el pie y remate con la cabeza.

- Nombre: "Crear jugador campo".
- Entrada 1: nombre (una cadena).
- Entrada 2: país (un valor del tipo "País").
- Entrada 3: día de nacimiento (un valor del tipo "Día").
- Entrada 4: mes de nacimiento (un valor del tipo "Mes").
- Entrada 5: año de nacimiento (un valor del tipo "año").
- Entrada 6: posición preferida (un valor del tipo "Posición").
- Entrada 7: número de dorsal (un valor del tipo "Dorsal").
- Entrada 8: valoración de fuerza (un valor del tipo "Valoración").
- Entrada 9: valoración de velocidad (un valor del tipo "Valoración").
- Entrada 10: valoración de resistencia (un valor del tipo "Valoración").
- Entrada 11: valoración de robos (un valor del tipo "Valoración").
- Entrada 12: valoración de centros (un valor del tipo "Valoración").
- Entrada 13: valoración de regate (un valor del tipo "Valoración").
- Entrada 14: valoración de remate con el pie (un valor del tipo "Valoración").
- Entrada 15: valoración de remate con la cabeza (un valor del tipo "Valoración").
- Entrada 16: nombre del equipo (una cadena).
- Entrada 17: día de fichaje (un valor del tipo "Día").
- Entrada 18: mes de fichaje (un valor del tipo "Mes").
- Entrada 19: año de fichaje (un valor del tipo "año").
- Entrada 20: importe de la ficha (un número entero).
- Entrada 21: importe de la cláusula de rescisión (un número entero).

- Precondición: la posición del jugador debe ser una posición de campo y debe existir un equipo con el nombre de equipo indicado y no debe existir ningún jugador con ese nombre y no debe existir ningún jugador que pertenezca al equipo con el mismo dorsal.
- Salida: una variable que representa el jugador creado, dos variables que representan fecha de nacimiento y fecha de fichaje y, por último, una variable que representa el equipo al que se ha asignado el jugador.
- Postcondición: el equipo tiene el nombre indicado en el estado final y el nuevo jugador de campo tiene los valores de atributos correspondientes (nombre, país, posición, dorsal, fuerza, velocidad, resistencia, robos, centros, regate, remate con el pie y remate con la cabeza) y está relacionado con la fecha de nacimiento (determinada en la entrada por día, mes y año) y, en función del país, el jugador pertenece a los conceptos "Nacional", "Comunitario" o "Extranjero" y está relacionado con el equipo y la fecha de comienzo de fichaje (fecha para la que se conocen su día, mes y año) y ese fichaje tiene los valores de ficha y cláusula de rescisión especificados en la entrada. Todas estas condiciones hacen referencia al estado final.

Este ejemplo podría reducirse en cantidad de entradas si en el modelo estructural se definiera un nuevo concepto, llamado "Ficha jugador campo" que representara la información necesaria para el alta de un jugador de campo (todas las entradas desde 1 a 21). En ese caso bastaría con tener como entrada una variable de ese concepto.

(3) Crear la alineación de un equipo para un partido concreto. Para ello habrá que elegir una táctica y luego rellenar los puestos con los mejores jugadores disponibles.

- Nombre: "Crear alineación de equipo"
- Entrada 1: equipo (instancia del concepto "Equipo")
- Entrada 2: día de la alineación (un valor del tipo "Día").
- Entrada 4: mes de la alineación (un valor del tipo "Mes").
- Entrada 5: año de la alineación (un valor del tipo "año").
- Precondición: no debe existir ninguna otra alineación asignada al equipo en una fecha con el mismo día, mes y año.
- Salida 1: alineación creada (instancia del concepto "Alineación")
- Salida 2: táctica elegida (instancia del concepto "Táctica")
- Salida 3: fecha de la alineación (instancia del concepto "Fecha")
- Postcondición: la nueva alineación está asignada al equipo en la nueva fecha y la nueva alineación está asociada a la táctica elegida y todos los puestos de la táctica elegida están asignados en la alineación a jugadores que juegan en la posición correspondiente o en una de sus posiciones afines. Todas estas condiciones hacen referencia al estado final.

4.1.2.2 Métodos de tarea

En muchos tipos de problemas la definición declarativa de tareas es suficiente para especificar un sistema. Sin embargo hay ejemplos de problemas, como muchos de los sistemas basados en el conocimiento, en los cuales es muy importante definir cómo se resuelven los problemas en el mundo real. Por ello es necesario un segundo elemento dentro del modelo de comportamiento: los métodos de tarea.

Los métodos de tarea permiten representar la forma de resolver una tarea. En general un método descompone una tarea en tareas más sencillas (las subtareas) y también define cómo se debe controlar la ejecución de las tareas para conseguir resolver la tarea principal del método.

Se define el **universo de métodos de tarea (M)** como el conjunto de todos los métodos de tarea que pueden definirse.

Se define cada **método (M_i)** como una de las formas de realizar una tarea (o, dicho de otro, modo una de las formas de resolver un problema). Para ello el método define la descomposición de la tarea en subtareas y la especificación del control que debe llevarse a cabo. Pueden existir varios métodos diferentes para realizar una misma tarea.

La Figura 4.27 representa gráficamente la relación entre métodos y tareas.

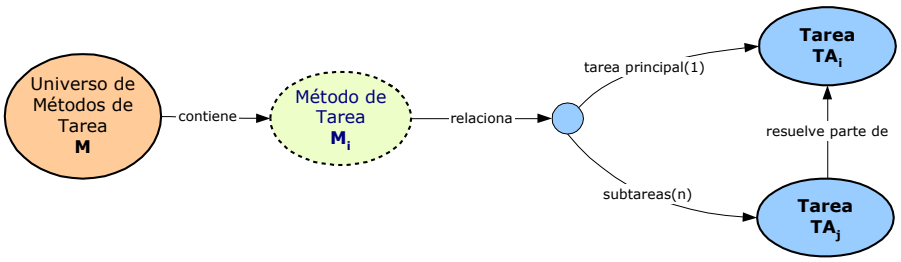


Figura 4.27 Relación entre métodos y tareas

Para especificar un método de tarea son necesarios los siguientes elementos:

- *Nombre* del método.
- *Tarea* que se resuelve con este método.
- *Subtareas* definidas por este método. Pueden existir casos en los cuales un método no defina subtareas. En esos casos el control únicamente podrá utilizar como acciones la realización de cambios elementales en el estado del modelo (operadores).
- *Control*, que especifica cómo se controla la ejecución de las subtareas y operadores para conseguir resolver la tarea global. La especificación del control puede usar como variables los elementos de la entrada y la salida de la tarea principal del método y usará alguno de los siguientes elementos:
  - *Expresiones*, que pueden ser elementales (como 2+3-4) o bien llamadas a funciones de consulta.
  - *Operadores*, que representan cambios elementales en el estado del sistema
  - Ejecución de una *subtarea*.
  - Instrucción de control imperativo (secuencia, condición, bucle, etc.)
  - Instrucción de control dirigido por eventos (cuando "evento" hacer "acción"), siendo un evento un cambio de estado o una condición sobre el instante de tiempo actual.
  - Instrucción de control heurístico ("case" con una condición heurística).

Equipo de fútbol

Se muestra a continuación un **método para resolver la tarea "Crear alineación equipo"**. El método consiste en elegir en primer lugar una táctica y después asignar los mejores jugadores posibles a los puestos de esa táctica. Elementos de este método:

- Nombre: "Partiendo de táctica".
- Tarea: "Crear alineación equipo".
- Subtarea 1: "Elegir táctica"
- Subtarea 2: "Asignar jugadores"
- Control: Se siguen dos pasos de forma secuencial (1) Elegir táctica (2) Asignar jugadores.

En esta especificación de método han surgido dos tareas nuevas ("Elegir táctica" y "Asignar jugadores") que deberían ser descritas como elementos del modelo de comportamiento.

4.1.2.3 Operadores

Se define un **operador** como un cambios elementales en el estado del sistema. Los operadores no forman parte de ningún modelo de comportamiento concreto, sino que se consideran predefinidos (como los tipos básicos) y pueden utilizarse dentro de las especificaciones de control de los métodos de tarea.

La lista de operadores existentes está basada en las dos operaciones básicas de la teoría de conjuntos, que son:

- Insertar un elemento en un conjunto.
- Eliminar un elemento de un conjunto.

A partir de estas dos operaciones básicas, y teniendo en cuenta todos los elementos definidos dentro del modelo estructural, se definen los operadores que aparecen en la Tabla 4.1 (están recogidos en orden alfabético).

Tabla 4.1 Lista de operadores predefinidos

Operador	Entrada	Salida	Descripción
AsignarInstanciaConcepto	IxC	∅	Asigna una instancia a un concepto. A partir de ese momento la instancia "hereda" atributos y asociaciones.
AsignarInstanciaAsociación	IASxAS	∅	Asigna una instancia de asociación a otra asociación
AñadirValorAtributoConcepto	IxVxAT	∅	Añade un par a un atributo de concepto.
AñadirValorAtributoAsociación	IASxVxAT	∅	Añade un par a un atributo de asociación.
BorrarInstanciaConcepto	I	∅	Borra definitivamente una instancia, eliminándola de todos los conceptos, atributos, asociaciones, ... en los que aparecía
BorrarInstanciaAsociación	IAS	∅	Elimina una instancia de asociación de todas las asociaciones a las que pertenece.

Operador	Entrada	Salida	Descripción
CrearInstanciaConcepto	(CAD∪∅)xC	I	Crea una instancia de concepto, dado su nombre (puede ser vacío) y dado el concepto al que pertenece. Devuelve la instancia creada. Si el nombre es vacío se genera uno automáticamente.
CrearInstanciaAsociación	IASxAS	∅	Crea una nueva instancia de una asociación
EliminarInstanciaConcepto	IxC	∅	Elimina una instancia de un concepto, pero sólo borra la instancia si únicamente pertenecía a ese concepto.
EliminarInstanciaAsociación	IASxAS	∅	Elimina una instancia de una asociación, pero no borra la instancia si pertenece a otra asociación
EliminarValorAtributoConcepto	IxVxAT	∅	Elimina un par de un atributo de concepto.
EliminarValorAtributoAsociación	IASxVxAT	∅	Elimina un par de un atributo de asociación.
EliminarValoresAtributoConcepto	IxAT	∅	Elimina todos los pares de un atributo de una instancia de concepto determinada
EliminarValoresAtributoAsociación	IASxAT	∅	Elimina todos los pares de un atributo de una instancia de asociación determinada
SustituirValorAtributoConcepto	IxVxAT	∅	Sustituye todos los pares del atributo de una instancia de concepto por un único par con el nuevo valor.
SustituirValorAtributoAsociación	IASxVxAT	∅	Sustituye todos los pares del atributo de una instancia de asociación por un único par con el nuevo valor.

Dado que en la definición de estado se recoge información sobre el tiempo, a la hora de simular el comportamiento de un sistema se supondrá que cada uno de estos operadores necesita únicamente una unidad de tiempo para poder realizarse.

4.1.2.4 Funciones de consulta

Al igual que los cambios elementales del estado, dentro de la especificación del control de los métodos es necesario poder consultar cuáles son los elementos del estado actual. Con este fin se han diseñado una serie de funciones de consulta.

Se define una **función de consulta** como una acción elemental que permite averiguar parte de la información representada en el estado actual. Las funciones de consulta no modifican el estado, sino que únicamente obtienen información.

Al igual que ocurre con los operadores, las funciones de consulta están predefinidas y pueden utilizarse para cualquier modelo de comportamiento.

En la Tabla 4.2 se recogen en orden alfabético las funciones de consulta existentes dentro de SETCM.

Tabla 4.2 Funciones de consulta predefinidos

Operador	**Entrada**	**Salida**	**Descripción**
Cardinalidad	CJTO	N	Devuelve el número de elementos de un conjunto
Diferencia	CJTOxCJTO	CJTO	Calcula la diferencia entre dos conjuntos de elementos
ElegirInstanciasConcepto	CxCND	CJTO	Devuelve un conjunto con las instancias del concepto que cumplen una condición
ElegirInstanciasAsociación	ASxCND	CJTO	Devuelve un conjunto con las instancias de la asociación que cumplen una condición
Intersección	CJTOxCJTO	CJTO	Calcula la intersección entre dos conjuntos de elementos
MáximoCadena	CJTO	CAD	Devuelve el máximo de un conjunto de cadenas (mayor en orden alfabético)
MáximoNúmero	CJTO	NUM	Devuelve el máximo de un conjunto de números
MínimoCadena	CJTO	CAD	Devuelve el mínimo de un conjunto de cadenas (menor en orden alfabético)
MínimoNúmero	CJTO	NUM	Devuelve el mínimo de un conjunto de números
ProyectarConjuntoTuplas	CJTOxRL	CJTO	Devuelve un conjunto con las instancias genéricas que aparecen en el rol correspondiente de un conjunto de tuplas de asociación
Primero	CJTO	ELTO	Devuelve el primer elemento de un conjunto
ProyectarTupla	IASxRL	IG	Devuelve la instancia genérica que aparece en el rol correspondiente de una tupla de asociación
Resto	CJTO	CJTO	Devuelve el conjunto resultante de eliminar el primer elemento
TuplasAsociación	AsxRLxIG	CJTO	Devuelve el conjunto de tuplas de la asociación en los que participa la instancia genérica en el rol correspondiente
TodosValoresAtributo	AT	CJTO	Devuelve un conjunto con todos los valores que aparecen en alguno de los pares del atributo
TodosValoresAtributoAsociación	ATS	CJTO	Devuelve un conjunto con todos los valores que aparecen en alguno de los pares del atributo de asociación
Unión	CJTOxCJTO	CJTO	Calcula la unión entre dos conjuntos de elementos
ValorAtributo	ATxI	VALOR	Devuelve el valor asignado en el atributo a una instancia de concepto (sólo para atributos monovaluados)

Operador	Entrada	Salida	Descripción
ValorAtributoAsociación	ATSxI	VALOR	Devuelve el valor asignado en el atributo a una instancia de asociación (sólo es válida para atributos monovaluados)
ValoresAtributo	ATxI	CJTO	Devuelve un conjunto con los valores asignados a la instancia en el atributo
ValoresAtributoAsociación	ATSxI	CJTO	Devuelve un conjunto con los valores asignados a la instancia de asociación en el atributo

Estas funciones de consulta pueden utilizarse dentro de los parámetros de los operadores, o bien formando parte de cualquier expresión de un tipo compatible con la salida de la función.

4.1.3 Condiciones Lógicas

En las definiciones realizadas hasta ahora hay cuatro elementos que requieren la utilización de condiciones lógicas: las invariantes de concepto, las invariantes de asociaciones, las precondiciones de tarea y las postcondiciones de tarea.

Por ello es necesario definir cómo se especifican las condiciones lógicas dentro de un modelo conceptual. Los elementos que componen una condición lógica son cuatro:

- *Condiciones*: representan expresiones lógicas que incluyen la definición de variables (cuantificadas o no).
- *Fórmulas*: representan expresiones lógicas en las que ya no hay definiciones de variables.
- *Átomos*: representan condiciones elementales sobre las instancias (de concepto, de atributo, de asociación, de atributo de asociación) y sobre los valores de tipos del modelo estructural.
- Expresiones de valor: representan expresiones que devuelven valores de los tipos básicos (enteros, reales, cadenas y booleanos).

Estos elementos se organizan de forma jerárquica. Las condiciones lógicas definen variables y se construyen a partir de fórmulas (condiciones sin variables). Las fórmulas se construyen a partir de átomos. Por último en los átomos se utilizan expresiones de valor (Figura 4.28).

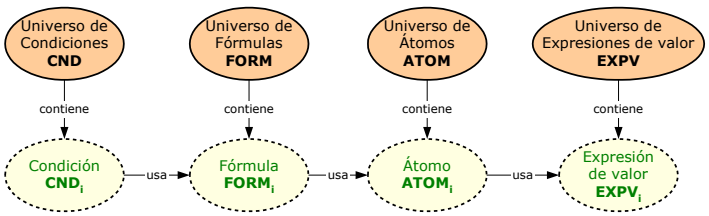


Figura 4.28 Condiciones, fórmulas, átomos y expresiones

La mayoría de las condiciones lógicas se evalúan sobre un único estado. Esto es así en el caso de invariantes de concepto, invariantes de asociación y precondiciones de tarea. Sin embargo las postcondiciones de tarea deben evaluarse en dos estados, ya que

representan las características del estado final de la tarea en función de cómo era el estado inicial de la misma. Por lo tanto se va a distinguir entre condiciones de un estado (o unarias) y condiciones de dos estados (o binarias).

La exposición de este apartado procede como sigue: en las cinco primeras secciones se definirán los elementos que componen las condiciones lógicas sobre un estado (condiciones, variables, fórmulas, átomos y expresiones), mientras que la sexta sección estará dedicada a comentar los cambios necesarios para especificar condiciones sobre dos estados.

4.1.3.1 Condiciones

Se define el **universo de condiciones** (**CND**) como el conjunto de todas las condiciones que incluyen definición de variables.

Se define cada **condición** (**CND_i**) como una expresión lógica que incluye variables y otras condiciones.

Las condiciones pueden ser de 6 tipos (Figura 4.29):

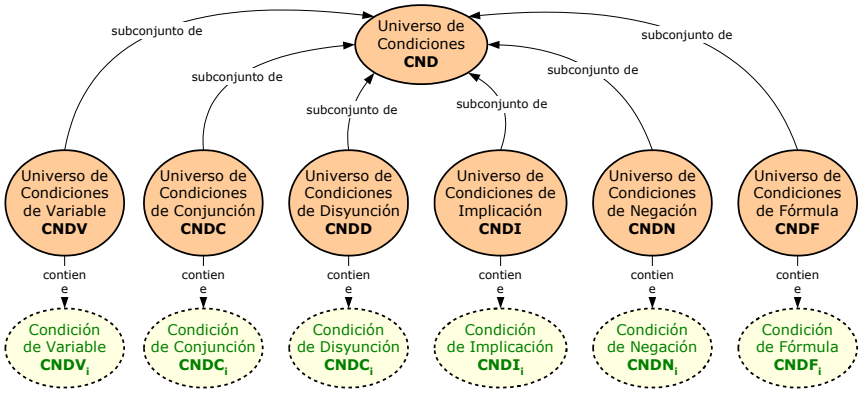


Figura 4.29 Los seis tipos de condiciones

- Una definición de variable que afecta a otra condición lógica.
- Una conjunción de dos condiciones lógicas.
- Una disyunción de dos condiciones lógicas.
- Una implicación de dos condiciones lógicas.
- Una negación de una condición lógica.
- Una fórmula (condición sin variables).

Seguidamente se definen estos seis tipos de condiciones:

- Se define el **universo de condiciones con variable** (**CNDV**) como el conjunto de condiciones que definen una variable que afecta a otra condición. Cada condición con variable se especifica mediante dos elementos:
  - *Variable* definida en la condición (véase la sección siguiente).
  - *Condición* afectada por la variable anterior.
- Se define el **universo de condiciones de conjunción** (**CNDC**) como el conjunto de condiciones que definen una conjunción (*and*) de dos condiciones. Cada condición

de conjunción se especifica mediante un par ordenado formado por las dos condiciones afectadas por la conjunción.

- Se define el **universo de condiciones de disyunción** (**CNDD**) como el conjunto de condiciones que definen una disyunción (*or*) de dos condiciones. Cada condición de disyunción se especifica mediante un par ordenado formado por las dos condiciones afectadas por la disyunción.
- Se define el **universo de condiciones de implicación** (**CNDI**) como el conjunto de condiciones que definen una implicación entre dos condiciones. Cada condición de implicación se especifica mediante un par ordenado formado por las dos condiciones afectadas por la implicación.
- Se define el **universo de condiciones de negación** (**CNDN**) como el conjunto de condiciones que definen una negación (*not*) de otra condición. Cada condición de negación se especifica mediante la condición que se está negando.
- Se define el **universo de condiciones de fórmula** (**CNDF**) como el conjunto de condiciones que simplemente definen una fórmula (condición que ya no define nuevas variables). Cada condición de fórmula se especifica mediante la fórmula correspondiente.

4.1.3.2 Variables

Siguiendo el esquema habitual de la lógica, las variables que se utilizan en las condiciones lógicas podrán ser cuantificadas o libres. Las variables cuantificadas toman valor durante la evaluación de las condiciones, mientras que las variables libres deben tener un valor asignado al comenzar la evaluación.

Las variables podrán tomar valores dentro de conjuntos que podrán ser tipos (el valor de la variable será un valor del tipo), conceptos (el valor de la variable será una instancia del concepto) y asociaciones (el valor de la variable será una instancia de asociación). También existe la posibilidad de que las variables tomen como valor conjuntos de esos elementos, es decir, conjuntos de valores de tipos, de instancias de concepto o de instancias de asociación.

Se define el **universo de definiciones de variables** (**VAR**), como el conjunto formado por todas las definiciones de variables de condiciones.

Se define cada **variable** (**VAR_i**) como la especificación de una variable dentro de una condición. Esta especificación tendrá los siguientes elementos:

- *Nombre* de la variable, que deberá ser único dentro de una condición.
- *Cuantificador* de la variable, que podrá no estar definido (variable libre) o tener uno de los siguientes valores:
  - *Para todo*: la condición afectada por esta variable deberá cumplirse para cualquier posible valor de la variable.
  - *Existe*: la condición afectada por esta variable deberá cumplirse al menos para uno de los valores de la variable.

  - *Existe único*: la condición afectada por esta variable deberá cumplirse sólo para uno de los valores de la variable.

- *Conjunto* en el que toma valores la variable. Puede ser:
  - *Tipo*: la variable representa un valor del tipo.
  - *Concepto*: la variable representa una instancia del concepto.
  - *Asociación*: la variable representa una instancia de la asociación (tupla).
  - *Partes de un tipo*: la variable representa un conjunto de valores del tipo.
  - *Partes de un concepto*: la variable representa un conjunto de instancias del concepto.
  - *Partes de una asociación*: la variable representa un conjunto de tuplas de la asociación.

4.1.3.3 Fórmulas

Se define el **universo de fórmulas** (**FORM**) como el conjunto de todas las condiciones que no pueden definir nuevas variables.

Se define cada **fórmula** (**FORM$_i$**) como una expresión lógica que incluye otras fórmulas y átomos.

Las fórmulas pueden ser de 5 tipos (Figura 4.30):

- Una conjunción de dos fórmulas.
- Una disyunción de dos fórmulas.
- Una implicación de dos fórmulas.
- Una negación de una fórmula.
- Un átomo (condición elemental).

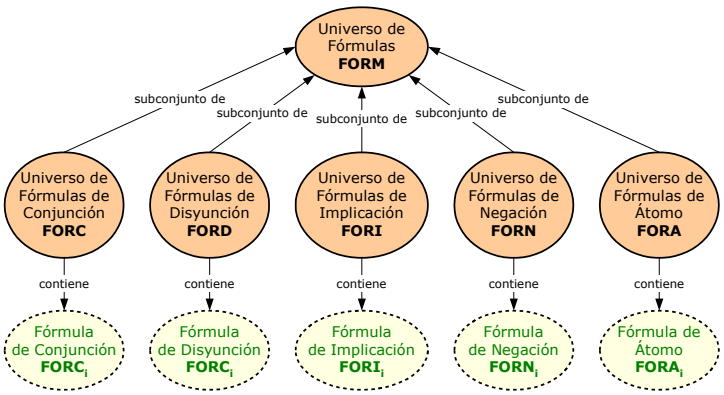


Figura 4.30 Los cinco tipos de fórmulas

Seguidamente se definen estos cinco tipos de fórmulas:

- Se define el **universo de fórmulas de conjunción** (**FORC**) como el conjunto de fórmulas que definen una conjunción (*and*) de dos fórmulas. Cada fórmula de conjunción se especifica mediante un par ordenado formado por las dos fórmulas afectadas por la conjunción.

- Se define el **universo de fórmulas de disyunción** (**FORD**) como el conjunto de fórmulas que definen una disyunción (*or*) de dos fórmulas. Cada fórmula de disyunción se especifica mediante un par ordenado formado por las dos fórmulas afectadas por la disyunción.
- Se define el **universo de fórmulas de implicación** (**FORI**) como el conjunto de fórmulas que definen una implicación entre dos condiciones. Cada fórmula de implicación se especifica mediante un par ordenado formado por las dos fórmulas afectadas por la implicación.
- Se define el **universo de fórmulas de negación** (**FORN**) como el conjunto de fórmulas que definen una negación (*not*) de otra fórmula. Cada fórmula de negación se especifica mediante la fórmula que se está negando.
- Se define el **universo de fórmulas de átomo** (**FORA**) como el conjunto de fórmulas que simplemente definen un átomo (condición elemental sobre el estado). Cada fórmula de átomo se especifica mediante el átomo correspondiente.

4.1.3.4 Átomos

Un átomo representa una condición elemental sobre el estado del sistema, en la que participan instancias (de concepto, de atributo, de asociación, de atributo de asociación) y valores de tipos del modelo estructural.

Se define el **universo de los átomos** (**ATOM**) como el conjunto de todos los átomos que pueden aparecer en condiciones lógicas.

Se define cada **átomo** (**ATOM$_i$**) como una condición simple sobre el estado del sistema.

Los átomos pueden ser de 8 tipos (Figura 4.31):

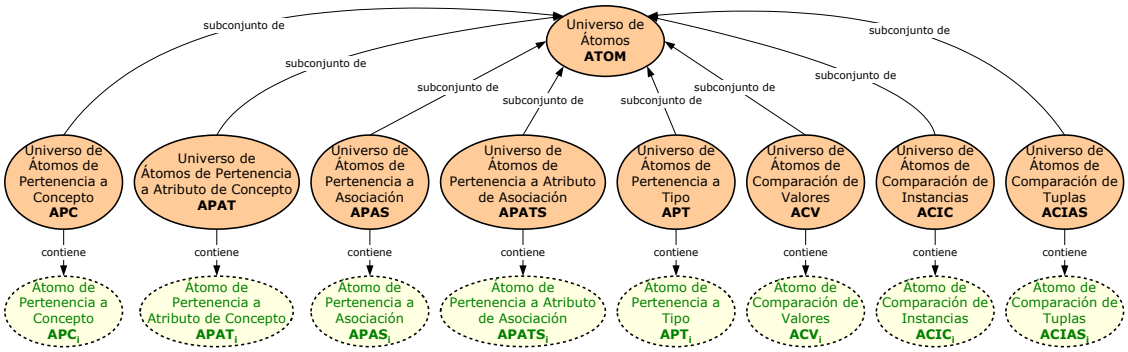


Figura 4.31 Los ocho tipos de átomos

- Pertenencia de una instancia a un concepto.
- Pertenencia de un par (instancia de concepto, valor) a un atributo.
- Pertenencia de una tupla a una asociación.
- Pertenencia de un par (tupla, valor) a un atributo de asociación.
- Pertenencia de un valor a un tipo.
- Comparación entre valores de un tipo.

- Comparación entre instancias de concepto.
- Comparación entre instancias de asociación.

Seguidamente se definen estos ocho tipos de átomos:

- Se define el **universo de átomos de pertenencia a concepto** (**APC**) como el conjunto de aquellos átomos que representan la condición de pertenencia de una instancia de concepto a un concepto concreto. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Instancia* de concepto del átomo, que podrá ser una instancia de concepto, una variable cuyos valores son instancias de concepto o bien una función de consulta que devuelva una instancia de concepto.
  - *Concepto* del átomo, que podrá ser un concepto, una variable cuyos valores son subconjuntos de un concepto o bien una función de consulta que devuelva un conjunto de instancias de concepto.
- Se define el **universo de átomos de pertenencia a atributo de concepto** (**APAT**) como el conjunto de aquellos átomos que representan la condición de pertenencia de un par (instancia de concepto, valor) a un atributo de concepto concreto. Cada átomo de este tipo se especifica mediante tres elementos:
  - *Instancia* de concepto del átomo, con las mismas posibilidades ya descritas.
  - *Valor* del átomo, que podrá ser un valor concreto, una variable que represente valores de un tipo, una expresión que devuelva valores de tipo o bien una función de consulta que devuelva un valor de un tipo.
  - *Atributo* de concepto del átomo, que sólo podrá ser un atributo de concepto concreto.
- Se define el **universo de átomos de pertenencia a asociación** (**APAS**) como el conjunto de aquellos átomos que representan la condición de pertenencia de una tupla (instancia de asociación) a una asociación concreta. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Tupla* del átomo, que podrá ser una tupla de elementos (formada por instancias o variables), una variable cuyos valores sean tuplas, o bien una función de consulta que devuelva una tupla.
  - *Asociación* del átomo, que podrá ser una asociación, una variable cuyos valores son subconjuntos de una asociación o bien una función de consulta que devuelva un conjunto de tuplas.
- Se define el **universo de átomos de pertenencia a atributo de asociación** (**APATS**) como el conjunto de aquellos átomos que representan la condición de pertenencia de un par (tupla, valor) a un atributo de asociación concreto. Cada átomo de este tipo se especifica mediante tres elementos:
  - *Tupla* del átomo, con las mismas posibilidades ya descritas.
  - *Valor* del átomo, con las posibilidades descritas anteriormente.

  - *Atributo* de asociación del átomo, que sólo podrá ser un atributo de asociación concreto.

- Se define el **universo de átomos de pertenencia a tipo** (**APT**) como el conjunto de aquellos átomos que representan la condición de pertenencia de un valor a un tipo concreto. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Valor* del átomo, con las posibilidades de representación ya descritas.
  - *Tipo* del átomo, que podrá ser un tipo, una variable cuyos valores son subconjuntos de un tipo o bien una función de consulta que devuelva un conjunto de valores de un tipo.
- Se define **el universo de átomos de comparación de valores** (**ACV**) como el conjunto de aquellos átomos que representan una comparación entre dos valores de tipo. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Operador de comparación* del átomo, que podrá ser: igual, distinto, menor, menor o igual, mayor, mayor o igual.
  - *Valores* del átomo, que es un par ordenado de valores, de forma que el átomo compara el primer elemento con el segundo. Cada uno de esos dos valores puede representarse de la misma forma que ya se ha descrito anteriormente.
- Se define **el universo de átomos de comparación de instancias de concepto** (**ACIC**) como el conjunto de aquellos átomos que representan una comparación entre dos instancias de concepto. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Operador de comparación* del átomo, que podrá ser: igual, distinto.
  - *Instancias* de concepto del átomo, que es un par ordenado de instancias de concepto, de forma que el átomo compara el primer elemento con el segundo. Cada una de esas dos instancias de concepto puede representarse de la misma forma que ya se ha descrito anteriormente.
- Se define **el universo de átomos de comparación de instancias de asociación** (**ACIAS**) como el conjunto de aquellos átomos que representan una comparación entre dos instancias de concepto. Cada átomo de este tipo se especifica mediante dos elementos:
  - *Operador de comparación* del átomo, que podrá ser: igual, distinto.
  - *Tuplas* del átomo, que es un par ordenado de tuplas (instancias de asociación), de forma que el átomo compara el primer elemento con el segundo. Cada una de esas dos tuplas puede representarse de la misma forma que ya se ha descrito anteriormente.

4.1.3.5 Expresiones de valor

Las expresiones se utilizan como parte de los átomos, tal y como se ha visto en el apartado anterior. Las expresiones se clasifican por el tipo de valores que devuelven y por el número de operandos que admiten.

EXPV es el **conjunto de las expresiones que devuelven valores**. Estas expresiones pueden ser de cuatro tipos (Figura 4.32):

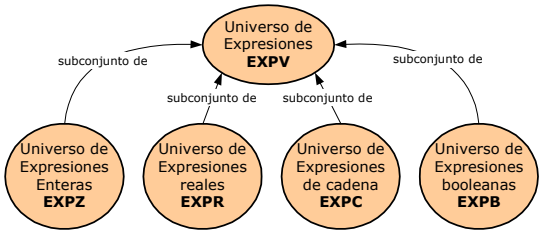


Figura 4.32 Tipos de expresiones de valor

- Expresiones enteras, que devuelven números enteros.
- Expresiones reales, que devuelven números reales.
- Expresiones de cadena, que devuelven cadenas de caracteres
- Expresiones booleanas, que devuelven valores booleanos

EXPZ es el **conjunto de las expresiones que devuelven valores enteros**. Estas expresiones pueden ser de tres tipos:

- Una operación con dos operandos enteros (suma, resta, multiplicación, división entera, módulo, potencia).
- Una operación con un operando entero (cambio de signo, factorial).
- Un valor entero o variable que represente enteros.

EXPR es el **conjunto de las expresiones que devuelven valores reales**. Estas expresiones pueden ser de tres tipos:

- Una operación con dos operandos reales (suma, resta, multiplicación, división real, división entera, módulo, potencia). Se admite que uno de los operandos sea un número entero.
- Una operación con un operando real (cambio de signo)
- Un valor real o variable que represente números reales.

EXPC es el **conjunto de las expresiones que devuelven cadenas**. Estas expresiones pueden ser de dos tipos:

- Una operación con dos operandos de tipo cadena (concatenación).
- Un valor de cadena o variable que represente cadenas.

EXPB es el **conjunto de las expresiones que devuelven valores booleanos**. Estas expresiones pueden ser de tres tipos:

- Una operación con dos operandos booleanos (and, or, xor, implica).
- Una operación con un operando booleano (not).
- Un valor booleano o variable que represente booleanos.

4.1.3.6 Condiciones unarias y binarias

Todas las definiciones anteriores (condiciones, variables, fórmulas, átomos y expresiones) permiten construir condiciones de un estado, llamadas condiciones binarias.

Sin embargo también son necesarias condiciones sobre dos estados, llamadas condiciones binarias. Estas condiciones lógicas deben permitir comparar expresiones de dos estados diferentes o, lo que es lo mismo, en dos instantes de tiempo diferentes (tiempo 1 o inicial y tiempo 2 o final).

Las condiciones lógicas binarias se construyen siguiendo la misma estructura que las condiciones unarias:

- Condiciones binarias que definen variables y que pueden ser: condición con variable, conjunción, disyunción, implicación, negación y fórmula.
- Fórmulas binarias que pueden ser: conjunción, disyunción, implicación, negación y átomo.
- Átomos binarios, que pueden ser:
  - Pertenencia de una instancia a un concepto en el instante de tiempo inicial o final.
  - Pertenencia de un par (instancia de concepto, valor) a un atributo en el instante de tiempo inicial o final.
  - Pertenencia de una tupla a una asociación en el instante de tiempo inicial o final.
  - Pertenencia de un par (tupla, valor) a un atributo de asociación en el instante de tiempo inicial o final.
  - Pertenencia de un valor a un tipo (átomo independiente del tiempo).
  - Comparación entre valores de un tipo (átomo independiente del tiempo).
  - Comparación entre instancias de concepto (átomo independiente del tiempo).
  - Comparación entre instancias de asociación (átomo independiente del tiempo).
- Expresiones, que son las mismas que en el caso de condiciones en un estado.

Por lo tanto, la diferencia más importante entre condiciones unarias y condiciones binarias radica en cuatro átomos, cuya evaluación es dependiente del tiempo: esos cuatro átomos pueden preguntar por el estado en el instante inicial o bien por el estado en el instante final.

4.2 FORMALIZACIÓN DEL MODELO

En este apartado se van a formalizar todos los elementos del método propuesto usando los elementos básicos de la teoría de conjuntos.

La estrategia de definición formal que se ha seguido es la siguiente:

- Se ha tomado como punto de partida la definición de “universos”: conjuntos primarios de elementos de un determinado tipo.

- Para definir las propiedades de cada elementos, se utilizan tuplas de funciones, Cada una de esas funciones representa una de las propiedades del elemento correspondiente.
- Según se han ido formalizando los elementos, se han identificado y formalizado restricciones sobre cada elemento y restricciones entre elementos diferentes.

A lo largo de la formalización se va a utilizar un amplio conjunto de símbolos y abreviaturas, que se encuentran recogidos en el Anexo A.

Empezando por el elemento de más alto nivel, corresponde formalizar en primer lugar lo que significa un modelo conceptual o formalización.

El **universo de conceptualizaciones** (**CNC**) es un conjunto de conceptualizaciones (CNC_i), cada una de las cuales se representa mediante un par ordenado: su modelo estructural y su modelo de comportamiento.

$$CNC = \{CNC_i \mid CNC_i \cong (EST_{CNC}(CNC_i), COM_{CNC}(CNC_i))\}$$

Los elementos que definen una conceptualización son:

- **EST_{CNC}** (**Estructura** de la conceptualización) es una función entre el conjunto de las conceptualizaciones (*CNC*) y el conjunto de los modelos estructurales (*EST*), de forma que a cada conceptualización se le asocia un modelo estructural:

$$EST_{CNC}: CNC \rightarrow EST$$

- **COM_{CNC}** (**Comportamiento** de la conceptualización) es una función entre el conjunto de las conceptualizaciones (*CNC*) y el conjunto de los modelos de comportamiento (COM), de forma que a cada conceptualización se le asocia un modelo de comportamiento:

$$COM_{CNC}: CNC \rightarrow COM$$

El orden de exposición de la formalización de los elementos del modelo va a ser similar al orden utilizado en la exposición de las definiciones de elementos.

4.2.1 Formalización del Modelo Estructural

El **universo de modelos estructurales** (**EST**) es un conjunto de modelos estructurales (EST_i), cada uno de los cuales se representa mediante una séptupla: tipos, conjuntos, atributos de concepto, asociaciones, atributos de asociación, clasificaciones de concepto y clasificaciones de asociación.

$$EST = \{EST_i \mid EST_i \cong (T_{EST}(EST_i), C_{EST}(EST_i), AT_{EST}(EST_i), AS_{EST}(EST_i), ATS_{EST}(EST_i), CL_{EST}(EST_i), CLS_{EST}(EST_i))\}$$

Los elementos que definen un modelo estructural son:

- **T_{EST}** (**Tipos** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de los tipos ($\Pi(T)$), de forma que a cada modelo estructural se le asocia un conjunto de tipos:

$$T_{EST}: EST \rightarrow \Pi(T)$$

- **C_{EST}** (**Conceptos** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de los conceptos (*Π(C)*), de forma que a cada modelo estructural se le asocia un conjunto de conceptos:

$$C_{EST}: EST \rightarrow \Pi(C)$$

- **AT_{EST}** (Atributos **de concepto** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de los atributos de concepto (*Π(AT)*), de forma que a cada modelo estructural se le asocia un conjunto de atributos de concepto:

$$AT_{EST}: EST \rightarrow \Pi(AT)$$

- **AS_{EST}** (**Asociaciones** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de las asociaciones (*Π(AS)*), de forma que a cada modelo estructural se le asocia un conjunto de asociaciones:

$$AS_{EST}: EST \rightarrow \Pi(AS)$$

- **ATS_{EST}** (**Atributos de asociación** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de los atributos de asociación (*Π(ATS)*), de forma que a cada modelo estructural se le asocia un conjunto de atributos de asociación:

$$ATS_{EST}: EST \rightarrow \Pi(ATS)$$

- **CL_{EST}** (Clasificaciones **de concepto** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de las clasificaciones de concepto (*Π(CL)*), de forma que a cada modelo estructural se le asocia un conjunto de clasificaciones de concepto:

$$CL_{EST}: EST \rightarrow \Pi(CL)$$

- **CLS_{EST}** (**Clasificaciones de asociación** del modelo estructural) es una función entre el conjunto los modelos estructurales (*EST*) y las partes del conjunto de las clasificaciones de asociación (*Π(CLS)*), de forma que a cada modelo estructural se le asocia un conjunto de clasificaciones de asociación:

$$CLS_{EST}: EST \rightarrow \Pi(CLS)$$

Sobre algunos de estos elementos se definen una serie de restricciones (por ejemplo, tanto el concepto como el tipo de cada atributo de concepto de un modelo estructural deben pertenecer a ese modelo estructural) que se formalizarán según se vaya formalizando cada uno de los elementos en los apartados siguientes.

4.2.1.1 Formalización de Tipos y Valores

Este apartado se va a dividir en tres partes: la formalización de los tipos y valores, la formalización de las restricciones de tipo y la formalización de los intervalos.

4.2.1.1.1 Tipos y Valores

El **universo de valores** (V) es el conjunto de todos los valores (V_i):

$$V = \{V_i\}$$

El **universo de tipos** (T) es un conjunto de subconjuntos de V, de forma que cada tipo (T_i) es un tipo básico o un tipo derivado:

$$T = \{T_i \subset V \mid T_i \in TB \vee T_i \in TD\}, \text{ es decir, } T = TB \cup TD$$

Todos los tipos tienen asociado un **nombre**, mediante la función correspondiente que permite asignar una cadena de caracteres a cada elemento del universo de tipos:

$$N_T: T \rightarrow CAD$$

El **nombre de un tipo debe ser único**, es decir, no pueden existir dos tipos diferentes con el mismo nombre. Por lo tanto el nombre de un tipo es su identificador:

$$\forall T_i, T_j \in T \bullet N_T(T_i) = N_T(T_j) \Leftrightarrow T_i = T_j$$

El **universo de tipos básicos** (TB) es un conjunto formado por cuatro conjuntos: los números enteros (Z), los números reales (R), las cadenas de caracteres (CAD) y los valores booleanos (B).

$$TB = \{Z, R, CAD, B\}$$

$$B = \{v, f\}$$

Como los tipos básicos están predefinidos, su nombre también lo está:

- $N_T(Z)$ = "Entero"
- $N_T(R)$ = "Real"
- $N_T(CAD)$ = "Cadena"
- $N_T(B)$ = "Booleano"

El **universo de tipos derivados** (TD) es un conjunto cuyos elementos se definen mediante una terna: su nombre, el tipo del que deriva y una restricción sobre los valores de ese tipo.

$$TD = \{TD_i \subset V \mid TD_i \cong (N_T(TD_i), B_T(TD_i), R_T(TD_i)\}$$

Como el nombre del tipo ya está formalizado, queda por ver los dos elementos restantes:

- $\mathbf{B_T}$ (**Base** del tipo) es una aplicación entre el conjunto de los tipos derivados (*TD*) y el tipo (básico o derivado) del que se deriva cada tipo derivado:

  $$B_T: TD \rightarrow T$$

  Debe cumplirse que el tipo derivado es un subconjunto del tipo del que deriva:

  $$\forall TD_i \in TD \bullet TD_i \subset B_T(TD_i)$$

- $\mathbf{R_T}$ (**Restricción** del tipo) es una aplicación inyectiva entre el conjunto de los tipos derivados (*TD*) y el conjunto de restricciones de tipo (*RES*), de forma que a cada tipo derivado se le asigna una única restricción de tipo:

  $$R_T: TD \rightarrow RES$$

  Las restricciones de tipo se formalizarán en el apartado siguiente.

Equipo de fútbol

Seguidamente se muestran formalmente los tipos derivados:

- País:

```
TD("País", CAD, {"España", "Portugal", "Francia", ..., "Argentina",
"Brasil", ..., "Japón", "Corea", ...})
```

- País comunitario:

```
TD("País comunitario", "País", { "Alemania", "Austria", "Bélgica",
"Dinamarca", "España", "Finlandia", "Francia", "Gran Bretaña", "Grecia",
"Irlanda", "Italia", "Luxemburgo", "Países Bajos", "Portugal", "Suecia"})
```

- País nacional:

```
TD("País nacional", "País comunitario", {"España"})
```

- País extranjero:

```
TD("País extranjero", "País", RDIF("País comunitario"))
```

- Posición:

```
TD("Posición", CAD, {"Portero", "Lateral izquierdo", "Lateral derecho",
   "Defensa central", "Líbero", "Medio centro", "Interior izquierdo",
   "Interior derecho", "Extremo izquierdo", "Extremo derecho", "Media
                        punta", "Delantero"})
```

- Posición portero:

```
TD("Posición portero", "Posición", {"Portero"})
```

- Posición campo:

```
TD("Posición campo", "Posición", RDIF("Posición portero"))
```

- Estado físico:

```
TD("Estado", CAD, {"bien", "lesionado"} )
```

- Valoración:

```
TD("Valoración", R, RR([0, 10]))
```

- Dorsal:

```
TD("Dorsal", Z, RZ([1, 25]))
```

- Categoría:

```
TD("Categoría", CAD, {"Primera", "Segunda", "Segunda B", "Tercera",
"Regional"})
```

- Carácter:

```
TD("Carácter", CAD, {"ofensivo", "defensivo", "neutro"})
```

- Día:

```
TD("Día", Z, RZ([0, 31]))
```

- Mes:

```
TD("Mes", Z, RZ([1, 12]))
```

- Año:

```
TD("Año", Z, RZ([1800, ∞)))
```

Teniendo en cuenta que todo tipo derivado está siempre incluido en su tipo base, que los únicos tipos básicos que se consideran son Z, R, CAD, B y dado que todos los números enteros son reales ($Z \subset R$), entonces se puede decir que el universo de valores es un conjunto que únicamente contiene números reales, cadenas y valores booleanos:

$$V = \{V_i \mid V_i \in R \vee V_i \in CAD \vee V_i \in B\},\ es\ decir,\ V = R \cup CAD \cup B$$

Por comodidad a la hora de continuar con la formalización de los modelos, se va a considerar que está predefinido el tipo de los números naturales, es decir, el tipo de los números enteros no negativos:

$$N \in TD \wedge B_T(N) = \text{"Natural"} \wedge B_T(N) = Z \wedge R_T(N) \in RZ \wedge I_T(R_T(N)) = [0, \infty)$$

Queda por hacer una definición relacionada con los tipos que será utilizada en las restricciones y también en la formalización de la clasificación de conceptos: los antecesores de un tipo.

Se define **antecesores de tipo** (ANT_T) como una aplicación entre el universo de tipos y las partes del universo de tipos tal que asocia a cada tipo un conjunto (que puede ser vacío) con los tipos de los que deriva (directa o indirectamente). En el caso de los tipos básicos, su conjunto de antecesores es vacío.

$$ANT_T: T \rightarrow \Pi(T)$$

$$\forall TD_i \in TD \bullet ANT_T(TD_i) = \{T_j \in T \mid T_j = B_T(TD_i) \vee T_j \in ANT_T(B_T(TD_i))\}$$

$$\forall TB_i \in TB \; ANT_T(TB_i) = \varnothing$$

Equipo de fútbol

Ejemplos de antecesores de tipo:

- ANT_T("Año") = {Z}
- ANT_T("País") = {CAD}
- ANT_T("País comunitario") = {"País", CAD}
- ANT_T("País nacional") = {"País comunitario", "País", CAD}

4.2.1.1.2 Restricciones de tipos derivados

El **universo de restricciones de tipo** (RES) es la unión de cuatro conjuntos: restricciones de enteros, restricciones de reales, restricciones de cadenas y restricciones de diferencia.

$$RES = RZ \cup RR \cup RCAD \cup RDIF$$

- Cada **restricción de entero** restringe un tipo numérico mediante un intervalo de valores enteros o una unión de intervalos de valores enteros.
- Cada **restricción de real** restringe un tipo numérico mediante un intervalo de valores reales o una unión de intervalos de valores reales.
- Cada **restricción de cadena** define explícitamente el conjunto de cadenas de caracteres que se consideran valores válidos del tipo derivado correspondiente.
- Cada **restricción de diferencia** define un subconjunto del tipo base considerado mediante la diferencia de conjuntos con otro tipo derivado.

En este apartado se van a formalizar cada uno de estos tipos de restricciones.

El **universo de restricciones de entero** (RZ) está formado por elementos que se definen mediante una única función. Esta función asigna un intervalo de números enteros a cada restricción de entero:

$$RZ = \{RZ_i / RZ_i \cong IT_{RZ}(RZ_i)\}$$

- **IT_{RZ}** (**intervalo de valores enteros**) es una aplicación entre el conjunto de las restricciones de entero (RZ) y el conjunto de los intervalos numéricos (IT), de forma que a cada restricción de entero se le asocia un intervalo que define los valores válidos del tipo derivado correspondiente.

  $$IT_{RZ}: RZ \rightarrow IT$$

  IT es el conjunto de los intervalos numéricos. Cada intervalo IT_i es un conjunto de números y se define de la forma habitual en matemáticas: mediante dos extremos (límite inferior y límite superior) que pueden ser abiertos (no se incluye el valor del límite) o cerrados (sí se incluye el valor del límite).

  Así, por ejemplo, el intervalo [3, 6] de los números naturales representa los valores {3, 4, 5, 6}, mientras que el intervalo (3, 6) representa los valores {4, 5}. También se permite expresar cada IT_i como una unión de uno o más intervalos elementales. Por ejemplo: $[3, 6] \cup (8, 10]$.

  Con el fin de no interrumpir la exposición de las restricciones de tipos, la definición formal de los intervalos se deja para el apartado siguiente.

Sólo se podrán definir restricciones de entero para aquellos tipos derivados que incluyan Z como antecesor y, además, debe cumplirse que el intervalo esté incluido dentro de los valores válidos del tipo base.

$$\forall TD_i \in TD \bullet R_T(TD_i) \in RZ \Rightarrow Z \in ANT_T(TD_i) \wedge IT_{RZ}(R_T(TD_i)) \subset B_T(TD_i)$$

Equipo de fútbol

Algunos de los tipos derivados definidos anteriormente para este ejemplo incluyen restricciones de entero. En concreto son:

- Dorsal: número entero entre 1 y 25

```
RZ([1, 25])
```

- Día: número entero entre 1 y 31

```
RZ([1, 31])
```

- Mes: número entero entre 1 y 12

```
RZ([1, 12])
```

- Año: número entero entre 1800 e infinito

```
RZ([1800, ∞))
```

El **universo de restricciones de real** (RR) está formado por elementos que se definen mediante una única función. Esta función asigna un intervalo de números reales a cada restricción de real:

$$RR = \{RR_i / RR_i \cong IT_{RR}(RR_i)\}$$

- **IT_{RR}** (**intervalo de valores reales**) es una aplicación entre el conjunto de las restricciones de real (RR) y el conjunto de los intervalos numéricos (IT), de forma que a cada restricción de real se le asocia un intervalo que define los valores válidos del tipo derivado correspondiente.

  $$IT_{RR}: RR \rightarrow IT$$

Sólo se podrán definir restricciones de real para aquellos tipos derivados que incluyan R como antecesor y, además, debe cumplirse que el intervalo esté incluido dentro de los valores válidos del tipo base.

$$\forall TD_i \in TD \bullet R_T(TD_i) \in RR \Rightarrow R \in ANT_T(TD_i) \wedge IT_{RR}(R_T(TD_i)) \subset B_T(TD_i)$$

Equipo de fútbol

El tipo derivado "Valoración" define una restricción de real (entre 0 y 10):

```
RR([0, 10])
```

El **universo de restricciones de cadena** (RCAD) está formado por elementos que se definen mediante una única función. Esta función asigna un conjunto de cadenas de caracteres a cada restricción de cadena:

$$RCAD = \{RCAD_i / RCAD_i \cong SC_R(RCAD_i)\}$$

- **SC_R (subconjunto de cadenas**) es una aplicación entre el conjunto de restricciones de cadena (RCAD) y las partes del conjunto de cadenas (Π(CAD)), de forma que a cada restricción de cadena se le asocia un único subconjunto de CAD.

$$SC_R: RCAD \rightarrow \Pi(CAD)$$

Sólo se podrán definir restricciones de cadena para aquellos tipos derivados que incluyan CAD como antecesor y, además, debe cumplirse que el conjunto de valores esté incluido dentro de los valores válidos del tipo base.

$$\forall TD_i \in TD \bullet R_T(TD_i) \in RCAD \Rightarrow CAD \in ANT_T(TD_i) \wedge SC_R(R_T(TD_i)) \subset B_T(TD_i)$$

Equipo de fútbol

Algunos de los tipos derivados definidos anteriormente para este ejemplo incluyen restricciones de cadena. En concreto son:

- País

```
RCAD({"España","Portugal","Francia",...,"Argentina","Brasil", ...,
                        "Japón", "Corea", ...})
```

- País comunitario

```
RCAD({ "Alemania", "Austria", "Bélgica", "Dinamarca", "España",
"Finlandia", "Francia", "Gran Bretaña", "Grecia", "Irlanda", "Italia",
"Luxemburgo", "Países Bajos", "Portugal", "Suecia"})
```

- País nacional

```
RCAD({"España"})
```

- Posición

```
RCAD({"Portero", "Lateral izquierdo", "Lateral derecho", "Defensa
central", "Líbero", "Medio centro", "Interior izquierdo", "Interior
 derecho", "Extremo izquierdo", "Extremo derecho", "Media punta",
                        "Delantero"})
```

- Posición portero

```
RCAD({"Portero"})
```

- Estado

```
RCAD({"bien", "lesionado"})
```

- Categoría

```
RCAD({"Primera", "Segunda", "Segunda B", "Tercera", "Regional"})
```

- Carácter

RCAD({"ofensivo", "defensivo", "neutro"})

El **universo de restricciones de diferencia** (RDIF) está formado por elementos que se definen mediante una única función. Esta función asigna un tipo derivado a cada restricción de diferencia:

$$RDIF = \{RDIF_i / RDIF_i \cong T_R(RDIF_i)\}$$

- $\mathbf{T_R}$ (**tipo derivado** de restricción de diferencia) es una aplicación entre el conjunto de las restricciones de diferencia (RDIF) y el conjunto de tipos derivados (TD), de forma que a cada restricción de diferencia se le asocia un único tipo derivado:

$$T_R: RDIF \rightarrow TD$$

Si la restricción de un tipo derivado es de diferencia, se debe cumplir que el tipo asignado a la restricción sea un tipo derivado del tipo base asociado y que el tipo resultante sea igual a la diferencia entre tipo base y tipo asignado a la restricción:

$$\forall TD_i \in TD \bullet R_T(TD_i) \in RDIF \Rightarrow B_T(TD_i) \in ANT_T(T_R(R_T(TD_i))) \wedge TD_i = B_T(TD_i) - T_R(R_T(TD_i))$$

Equipo de fútbol

En este ejemplo hay dos tipos derivados definidos con restricciones calculadas:

- "País extranjero", que representa todos los países excepto los comunitarios):

RDIF("País comunitario")

- "Posición campo", que representa todas las posiciones excepto la de portero:

RDIF("Posición portero")

Una vez definidas formalmente las restricciones que permiten especificar los tipos derivados, es el momento de especificar la condición de pertenencia de un valor a un tipo derivado:

- Si el tipo derivado se define con una restricción de entero, todo valor del tipo derivado debe ser un entero que pertenezca al intervalo.

$$\forall TD_i \in TD \bullet R_T(TD_i) = RZ_i \in RZ \Rightarrow (\forall x \in V \bullet x \in TD_i \Leftrightarrow x \in Z \wedge x \in IT_{RZ}(RZ_i))$$

- Si el tipo derivado se define con una restricción de real, todo valor del tipo derivado debe ser un real que pertenezca al intervalo.

$$\forall TD_i \in TD \bullet R_T(TD_i) = RR_i \in RR \Rightarrow (\forall x \in V \bullet x \in TD_i \Leftrightarrow x \in R \wedge x \in IT_{RZ}(RR_i))$$

- Si el tipo derivado se define con una restricción de cadena, todo valor del tipo derivado debe ser una cadena perteneciente al conjunto de cadenas definido.

$$\forall TD_i \in TD \bullet R_T(TD_i) = RCAD_i \in RCAD \Rightarrow (\forall x \in V \bullet x \in TD_i \Leftrightarrow x \in SC_R(RCAD_i)$$

- Si el tipo derivado se define con una restricción de diferencia, todo valor del tipo derivado debe pertenecer al tipo base y no pertenecer al tipo de la restricción.

$$\forall TD_i \in TD \bullet R_T(TD_i) = RDIF_i \in RDIF \Rightarrow (\forall x \in V \bullet x \in TD_i \Leftrightarrow x \in B_T(TD_i) \wedge x \notin T_R(RDIF_i))$$

4.2.1.1.3 Formalización de Intervalos

Los intervalos forman parte de la definición de restricciones de entero (RZ) o real (RR).

IT es el **universo de intervalos numéricos**. Estos intervalos pueden ser de dos tipos: intervalos básicos (definidos por sus límites) e intervalos de unión (definidos como la unión de intervalos básicos).

$$IT = ITB \cup ITU$$

ITB es el **conjunto de intervalos numéricos básicos**. Cada uno de ellos se define mediante dos extremos (inferior y superior).

$$ITB = \{ITB_i \mid ITB_i \cong (EI_{ITB}(ITB_i), ES_{ITB}(ITB_i))\}$$

- **EI_{ITB}** (**Extremo inferior** de intervalo básico). Es una aplicación entre el conjunto de los intervalos numéricos básicos (ITB) y el conjunto de extremos de intervalo (EXT), de forma que asocia un extremo inferior a cada intervalo básico.

  $$EI_{ITB}: ITB \rightarrow EXT$$

  EXT es el **conjunto de los extremos de intervalo**. Cada extremo se define mediante un valor numérico y un tipo (que indica si es abierto o cerrado).

  $$EXT = \{EXT_i \mid EXT_i \cong (V_{EXT}(EXT_i), T_{EXT}(EXT_i))\}$$

  - **V_{EXT}** (**Valor** de extremo de intervalo). Es una aplicación entre el conjunto de los extremos de intervalo (EXT) y el conjunto de números reales (R), de forma que a cada extremo se le asocia un único valor real.

    $$V_{EXT}: EXT \rightarrow R$$

  - **T_{EXT}** (**Tipo** de extremo de intervalo). Es una aplicación entre el conjunto de los extremos de intervalo (EXT) y el conjunto de tipos de extremo (TEX), de forma que a cada extremo se le asocia un único tipo.

    $$T_{EXT}: EXT \rightarrow TEX$$

    TEX es el **conjunto de tipos de extremo** de intervalos, y está formado por dos valores: "abierto" y "cerrado":

    $$TEX = \{\text{“abierto”, “cerrado”}\}$$

- **ES_{ITB}** (**Extremo superior** de intervalo básico). Es una aplicación entre el conjunto de los intervalos numéricos básicos (ITB) y el conjunto de extremos de intervalo (EXT), de forma que asocia un extremo superior a cada intervalo básico.

  $$ES_{ITB}: ITB \rightarrow EXT$$

Se debe cumplir que el extremo superior de un intervalo sea mayor o igual que el extremo inferior:

$$\forall ITB_i \in ITB \bullet V_{EXT}(EI_{ITB}(ITB_i)) \leq V_{EXT}(ES_{ITB}(ITB_i))$$

Una vez definida la estructura de un intervalo básico, se puede definir su semántica. Un intervalo numérico básico es un subconjunto de R ($\forall ITB_i \in ITB \bullet ITB_i \subseteq R$) que cumple las siguientes restricciones:

- Si el extremo inferior es abierto, todos sus valores deben ser mayores que el valor del extremo inferior:

$$\forall ITB_i \in ITB \bullet \forall x \in ITB_i \bullet T_{EXT}(EI_{ITB}(ITB_i)) = \text{"abierto"} \Rightarrow x > V_{EXT}(EI_{ITB}(ITB_i))$$

- Si el extremo inferior es cerrado, todos sus valores deben ser mayores o iguales que el valor del extremo inferior:

$$\forall ITB_i \in ITB \bullet \forall x \in ITB_i \bullet T_{EXT}(EI_{ITB}(ITB_i)) = \text{"cerrado"} \Rightarrow x \geq V_{EXT}(EI_{ITB}(ITB_i))$$

- Si el extremo superior es abierto, todos sus valores deben ser menores que el valor del extremo superior:

$$\forall ITB_i \in ITB \bullet \forall x \in ITB_i \bullet T_{EXT}(ES_{ITB}(ITB_i)) = \text{"abierto"} \Rightarrow x < V_{EXT}(ES_{ITB}(ITB_i))$$

- Si el extremo inferior es cerrado, todos sus valores deben ser menores o iguales que el valor del extremo superior:

$$\forall ITB_i \in ITB \bullet \forall x \in ITB_i \bullet T_{EXT}(ES_{ITB}(ITB_i)) = \text{"cerrado"} \Rightarrow x \leq V_{EXT}(ES_{ITB}(ITB_i))$$

Una restricción importante sobre los extremos de un intervalo básico es que, si el intervalo forma parte de una restricción de entero, entonces el valor de los dos extremos debe ser entero y se considera que el intervalo es un conjunto de valores enteros:

$$\forall TD_i \in TD \bullet R_T(TD_i) = RZ_i \in RZ \Rightarrow (IT_{RZ}(RZ_i) \in ITB \Rightarrow V_{EXT}(EI_{ITB}(IT_{RZ}(RZ_i))) \in Z \wedge V_{EXT}(EI_{ITB}(IT_{RZ}(RZ_i))) \in Z \wedge IT_{RZ}(RZ_i) \subseteq Z)$$

ITU es el **conjunto de intervalos numéricos de unión**. Cada uno de ellos se define mediante un conjunto de intervalos básicos cuya unión se calcula.

$$ITU = \{ITU_i \mid ITU_i \cong CIT_{ITU}(ITU_i)\}$$

- **CIT_{ITU}** (**Conjunto de intervalos** de intervalo de unión) es una aplicación inyectiva entre el conjunto de intervalos numéricos de unión (ITU) y las partes del conjunto de intervalos básicos (P(ITB)), de forma que a cada intervalo de unión se le asigna un conjunto de intervalos básicos.

$$CIT_{ITU}: ITU \rightarrow \Pi(ITB)$$

Se debe cumplir la condición de que los intervalos básicos que definen un intervalo de unión sean disjuntos:

$$\forall ITU_i \in ITU \cap CIT_{ITU}(ITU_i) = \varnothing$$

Cada intervalo de unión es un subconjunto de R ($\forall ITU_i \in ITU, ITU_i \subseteq R$) de forma que sus valores pertenecen a alguno de sus intervalos básicos:

$$\forall ITU_i \in ITU \bullet \forall x \in ITU_i \bullet \exists ITB_i \in ITB \bullet ITB_i \in CIT_{ITU}(ITU_i) \wedge x \in ITB_i$$

Si el intervalo de unión forma parte de una restricción de entero, entonces el valor de los dos extremos de cada uno de sus intervalos básicos debe ser entero y esos intervalos básicos son conjuntos de números enteros:

$$\forall TD_i \in TD \bullet R_T(TD_i) = RZ_i \in RZ \Rightarrow (IT_{RZ}(RZ_i) \in ITU \Rightarrow \forall ITB_i \in CIT_{ITU}(ITU_i) \bullet V_{EXT}(EI_{ITB}(ITB_i)) \in Z \wedge V_{EXT}(EI_{ITB}(ITB_i)) \in Z \wedge ITB_i \subseteq Z)$$

Queda simplemente un detalle respecto a la notación. A lo largo del presente documento se va a utilizar la notación matemática habitual para representar intervalos. Así, el intervalo con extremo inferior abierto 5 y extremo superior cerrado 15 se representará como:

(5, 15]

en vez de la notación formal introducida más arriba, que sería:

ITB((5, "abierto"), (15, "cerrado"))

4.2.1.2 Formalización de Conceptos e Instancias

El **universo de instancias** (I) es el conjunto de todas las instancias que pueden existir en cualquier instante de tiempo. Cada elemento de I, llamado **instancia**, es un elemento indivisible que existe durante un tiempo. Toda instancia se caracteriza por tener un nombre, un instante de creación y un instante de destrucción:

$$I = \{I_i \mid I_i \cong (N_I(I_i), IC_I(I_i), ID_I(I_i))\}$$

- **N_I** (**nombre** de instancia) es una aplicación entre el universo de instancias (*I*) y el tipo que representa las cadenas (*CAD*), de forma que a cada instancia se le asocia un nombre.

  $$N_I: I \rightarrow CAD$$

  El nombre de una instancia es único, es decir, dos instancias diferentes deben tener nombres diferentes, independientemente de los conceptos a los que pertenezcan:

  $$\forall I_i, I_j \in I \bullet N_I(I_i) = N_I(I_j) \Leftrightarrow I_i = I_j$$

  Esta última condición permite utilizar el nombre de una instancia como su identificador.

- **IC_I** (**instante de creación** de instancia) es una aplicación entre el universo de instancias (*I*) y el tipo que representa los números naturales (*N*), de forma que a cada instancia se le asocia un número que representa su instante de creación.

  $$IC_I: I \rightarrow N$$

- **ID_I** (**instante de destrucción** de instancia) es una aplicación entre el universo de instancias (*I*) y el tipo que representa los números naturales (*N*), de forma que a cada instancia se le asocia un número que representa su instante de destrucción.

  $$ID_I: I \rightarrow N$$

  Siempre debe cumplirse que el instante de destrucción sea mayor que el de creación:

  $$\forall I_i \in I \bullet IC_I(I_i) < ID_I(I_i)$$

Cada instancia se caracterizará por tener asignados una serie de valores (mediante atributos) y por estar relacionada con otras instancias (mediante asociaciones) tal y como se verá en apartados siguientes.

Una restricción importante es que no tiene sentido ninguna instancia que no esté relacionada al menos con un valor o bien con otra instancia. En otro caso habría que considerar esa instancia más bien como un valor de tipo cadena.

Equipo de fútbol

Algunos ejemplos de instancias:

- "Loïc" (persona)
- "Figo", "Rivaldo" (jugadores)

- "Real Madrid", "Barcelona" (equipos)

El **universo de conceptos** (C) es el conjunto de todos los conceptos que pueden definirse. Cada **concepto** C_i es un conjunto de instancias (es decir, es un subconjunto de I), de forma que todas ellas cumplen una propiedad. Todo concepto se define mediante un par formado por un nombre y una propiedad invariante.

$$C = \{C_i \subset I \mid C_i \cong (N_C(C_i), INV_C(C_i))\}$$

- **N_C** (**Nombre** de concepto) es una aplicación entre el conjunto de los conceptos (C) y el tipo que representa las cadenas (*CAD*), de forma que a cada concepto se le asocia una cadena:

  $$N_C: C \rightarrow CAD$$

  Dentro de un modelo estructural concreto, el nombre de cada concepto es único (dos conceptos diferentes del mismo modelo estructural deben tener nombres distintos):

  $$\forall EST_i \in EST \bullet \forall C_i, C_j \in C_{EST}(EST_i) \bullet N_C(C_i) = N_C(Cj) \Leftrightarrow C_i = C_j$$

  Así el nombre de un concepto podrá utilizarse como su identificador dentro del contexto de un modelo estructural.

- **INV_C** (**Invariante** de concepto) es la condición que deben cumplir todas las instancias del concepto. Es una aplicación inyectiva entre el conjunto de los conceptos (*C*) y el universo de las condiciones lógicas (*CND*), de forma que a cada concepto se le asocia su invariante:

  $$INV_C: C \rightarrow CND$$

  En todo instante de tiempo se cumple que una instancia pertenece a un concepto si y solo si la evaluación de la invariante del concepto para esa instancia en ese instante de tiempo es cierta. Esta condición se formalizará en el apartado dedicado a formalizar las condiciones lógicas y su evaluación.

Equipo de fútbol

Seguidamente se recogen los conceptos principales de este ejemplo (muchos no tienen invariante definida por no ser necesario para el problema planteado):

- **Persona**:

```
C("Persona", ...)
```

- **Jugador**:

```
C("Jugador", ...)
```

- **Equipo**:

```
C("Equipo", ...)
```

- **Alineación**:

```
C("Alineación", ...)
```

- **Táctica**:

```
C("Táctica", ...)
```

- **Puesto**:

```
C("Puesto", ...)
```

- **Fecha** (se define como invariante la condición de fecha correcta, que de momento se expresa informalmente):

```
C("Fecha",
Si mes pertenece a {4, 6, 9, 11) entonces día <=30
Si mes es 2 y año es bisiesto entonces día <=29
Si mes es 2 y año no es bisiesto entonces día <=28 )
```

Ya se ha comentado que las instancias tienen asignado un tiempo de vida determinado, especificado mediante sus instantes de creación y destrucción.

Un concepto es un conjunto de instancias, que contiene todas las instancias que han sido y son del concepto. De la misma forma que las instancias se crean y se destruyen en el tiempo, la relación de pertenencia entre un concepto y una instancia también debería recoger información temporal. Esto es muy importante dado que, como ya se comentó en la definición de clasificaciones de concepto, una instancia puede pertenecer a varios conceptos y esto puede suceder en tiempos diferentes.

Se define el **tiempo de pertenencia de una instancia a un concepto** como una función parcial del producto cartesiano entre conceptos e instancias ($C \times I$) con el producto cartesiano de los números naturales consigo mismos ($N \times N$), de forma que, si una instancia pertenece (o ha pertenecido alguna vez) a un concepto, esta función devuelve un par formado por el instante de tiempo en el que la instancia empezó a formar parte del concepto (inicio de pertenencia) y el instante en el que la instancia terminó de formar parte del concepto (fin de pertenencia):

$$TP_C: C \times I \mapsto N \times N$$

Si una instancia pertenece a un concepto, deben cumplirse las siguientes restricciones con respecto al valor del tiempo de pertenencia:

- El inicio de pertenencia debe ser menor que el final de pertenencia.
- El inicio de pertenencia debe ser mayor o igual que el tiempo de creación de la instancia.
- El fin de pertenencia debe ser menor o igual que el tiempo de destrucción de la instancia.

Expresado formalmente:

$$\forall C_i \in C \bullet \forall I_i \in I \bullet I_i \in C_i \Rightarrow (TP_C(C_i, I_i)\,[1] \leq TP_C(C_i, I_i)\,[2] \wedge TP_C(C_i, I_i)\,[1] \geq IC_I(I_i) \wedge TP_C(C_i, I_i)\,[2] \leq ID_I(I_i))$$

Se ha utilizado la notación de corchetes para indicar el acceso al primer y segundo elementos de un par ordenado. Así, si P es un par ordenado, entonces $P[1]$ es el primer elemento del par y $P[2]$ es el segundo.

Esta función de tiempo de pertenencia permite definir el **estado de un concepto en un instante de tiempo determinado** como una función entre el producto cartesiano del universo de conceptos con los números naturales y las partes del universo de instancias, de forma que dado un concepto y un instante de tiempo devuelve el conjunto de sus instancias en ese momento:

$$ST_C: C \times N \rightarrow \Pi(I)$$

Una instancia de un concepto pertenece al estado de ese concepto en un instante determinado si y sólo si el tiempo de pertenencia de esa instancia al concepto incluye el instante considerado (lo cual quiere decir que también el tiempo de vida de la instancia de concepto incluye al instante de tiempo considerado):

$$\forall t_i \in N \bullet \forall C_i \in C \bullet \forall I_i \in C_i \bullet I_i \in ST_C(C_i, t_i) \Leftrightarrow (TP_C(C_i, I_i)\,[1] \leq t_i \wedge TP_C(C_i, I_i)\,[2] \geq t_i)$$

4.2.1.3 Formalización de Atributos de Concepto

El **universo de atributos** de concepto (AT) es el conjunto formado por atributos de concepto, que se especifican mediante seis funciones: nombre, concepto, tipo, grado, límite y valor por omisión:

$$AT = \{AT_i \subset I \times V \mid AT_i \cong (N_{AT}(AT_i), C_{AT}(AT_i), T_{AT}(AT_i), G_{AT}(AT_i), L_{AT}(AT_i), VO_{AT}(AT_i))\}$$

- **N_{AT}** (**Nombre** de atributo) es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de las cadenas (*CAD*), de forma que a cada atributo se le asocia una cadena:

  $$N_{AT}: AT \rightarrow CAD$$

- **C_{AT}** (**Concepto** de atributo) es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de los conceptos (*C*), de forma que a cada atributo se le asigna un único concepto (dicho de otro modo, el atributo "pertenece" a un concepto):

  $$C_{AT}: AT \rightarrow C$$

  Para todos los atributos de un modelo estructural debe cumplirse que su concepto también pertenezca al modelo estructural.

  $$\forall EST_i \in EST \bullet \forall AT_i \in AT_{EST}(EST_i) \bullet C_{AT}(AT_i) \in C_{EST}(EST_i)$$

  El nombre de un atributo debe ser único para todos los atributos de un mismo concepto:

  $$\forall AT_i, AT_j \in AT \bullet AT_i \neq AT_j \wedge C_{AT}(AT_i) = C_{AT}(AT_j) \Rightarrow N_{AT}(AT_i) \neq N_{AT}(AT_j)$$

  De esta forma para identificar unívocamente a un atributo será necesario utilizar el nombre del concepto al que pertenece, seguido por el propio nombre del atributo.

- **T_{AT}** (**Tipo** de atributo) es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de los tipos (*T*), de forma que a cada atributo se le asocia un único tipo:

  $$T_{AT}: AT \rightarrow T$$

  Para todos los atributos de un modelo estructural debe cumplirse que su tipo también pertenezca al modelo estructural.

  $$\forall EST_i \in EST \bullet \forall AT_i \in AT_{EST}(EST_i) \bullet T_{AT}(AT_i) \in T_{EST}(EST_i)$$

  Dados el concepto y el tipo de un atributo, el atributo será en realidad un subconjunto de su producto cartesiano:

  $$\forall AT_i \in AT \bullet AT_i \subset C_{AT}(AT_i) \times T_{AT}(AT_i)$$

- **G_{AT}** (**Grado** de atributo) es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de los grados de participación (*GR*), de forma que a cada atributo se le

asocia un único grado (restricción sobre el número mínimo de valores del atributo asociados a cada instancia del concepto en un instante de tiempo):

$$G_{AT}: AT \rightarrow GR$$

GR es un conjunto formado por dos valores: GR = {"obligatorio", "opcional"}.

Si el valor del grado de un atributo es "obligatorio", indica que en cada instante de tiempo toda instancia que pertenezca al estado del concepto deberá tener al menos un valor asignado en el atributo en ese instante de tiempo. En otro caso, podrán existir instancias del concepto que no tengan valor asociado. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

- **L_{AT}** (**Límite** de atributo) es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de los límites de participación (*LI*), de forma que a cada atributo se le asocia un único límite que restringe el número máximo de valores asociados a una instancia del concepto mediante el atributo:

$$L_{AT}: AT \rightarrow LI$$

Siendo LI = {"único", "múltiple"}

Si el valor del límite de un atributo es "único", indica que en cualquier instante de tiempo toda instancia que pertenezca al estado del concepto podrá tener como máximo un valor asignado en el atributo. En otro caso, podrán existir instancias del concepto que tengan más de un valor asociado. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

La definición de límite para el número de valores de un atributo implica que existen atributos multivaluados, es decir, atributos que pueden tener varios valores asignados a una misma instancia de concepto.

- **VO_{AT} (Valor por omisión** del atributo) es una aplicación que permite asignar un valor a un atributo de concepto para que sea utilizado por todas las instancias del concepto que no tengan valor asignado en un instante dado.

La existencia de valor por omisión en un atributo es opcional, por lo tanto VO_{AT} es una aplicación entre el conjunto de atributos (*AT*) y el conjunto de los valores (*V*) unido al conjunto formado por el conjunto vacío ($\{\varnothing\}$) de forma que a cada atributo se le puede asociar un valor o no:

$$VO_{AT}: AT \rightarrow V \cup \{\varnothing\}$$

Se debe cumplir que, si un atributo tiene asignado valor por omisión, entonces ese valor debe pertenecer al tipo del atributo:

$$\forall AT_i \in AT \bullet VO_{AT}(AT_i) \neq \varnothing \Rightarrow VO_{AT}(AT_i) \in T_{AT}(AT_i)$$

Si un atributo tiene definido valor por omisión, entonces ese valor será el asignado a todas las instancias del estado del concepto mientras no se les asigne explícitamente un valor. Expresado en otros términos, si existe valor por omisión, entonces toda instancia del concepto tendrá asignado un valor: el valor por omisión o bien otro valor distinto del valor por omisión. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

Equipo de fútbol

Seguidamente se describen atributos del ejemplo, ordenados por el concepto en el que se definen:

- Concepto **Fecha**:

AT("día", "Fecha", "Día", "obligatorio", "único", ∅)

AT("mes", "Fecha", "Mes", "obligatorio", "único", ∅)

AT("año", "Fecha", "Año", "obligatorio", "único", ∅)

- Concepto **Persona**:

AT("nombre", "Persona", CAD, "obligatorio", "único", ∅)

AT("país", "Persona", "País", "obligatorio", "múltiple", ∅)

- Concepto **Jugador**:

AT("posición", "Jugador", "Posición", "obligatorio", "único", ∅)

AT("dorsal", "Jugador", "Dorsal", "obligatorio", "único", ∅)

AT("fuerza", "Jugador", "Valoración", "obligatorio", "único", ∅)

AT("velocidad", "Jugador", "Valoración", "obligatorio", "único", ∅)

AT("resistencia", "Jugador", "Valoración", "obligatorio", "único", ∅)

AT("estado", "Jugador", "Estado", "obligatorio", "único", "bien")

- Concepto **Equipo**:

AT("nombre", "Equipo", CAD, "obligatorio", "único", ∅)

AT("categoría", "Equipo", "Categoría", "obligatorio", "único", ∅)

- Concepto **Alineación**:

AT("nombre", "Alineación", CAD, "obligatorio", "único", ∅)

- Concepto **Táctica**:

AT("nombre", "Táctica", CAD, "obligatorio", "único", ∅)

AT("carácter", "Táctica", "Carácter", "obligatorio", "único", "neutro")

- Concepto **Puesto**:

AT("posición", "Puesto", "Posición", "obligatorio", "único", ∅)

AT("afines", "Puesto", "Posición", "opcional", "múltiple", ∅)

Teniendo en cuenta todas las definiciones realizadas sobre atributos, se puede indicar formalmente cómo están definidas las instancias de un atributo.

El **universo de instancias de atributo** (IAT) es un conjunto cuyos elementos son pares ordenados formados por una instancia de concepto y un valor de un tipo.

$$IAT = \{IAT_i \mid IAT_i = (I_i, V_i), I_i \in I\ V_i \in I\}$$

Para un atributo concreto sus instancias deberán contener una instancia del concepto del atributo y un valor del tipo del atributo:

$$AT_i = \{IAT_j \in IAT \mid IAT_j = (I_k, V_l), I_k \in C_{AT}(AT_i)\ V_l \in T_{AT}(AT_i)\}$$

Por lo tanto, toda instancia de un atributo pertenece al producto cartesiano de su concepto con su tipo:

$$\forall AT_i \in AT,\ \forall IAT_j \in AT_i, IAT_j \in C_{AT}(AT_i) \times T_{AT}(AT_i)$$

Equipo de fútbol

Ejemplos de instancias de atributo:

```
"Persona"."nombre" = {("Loïc", "Loïc Antonio Martínez Normand"), ...}
"Persona"."país" = {("Loïc", "España"), ("Loïc", "Francia"), ...}
"Jugador"."posición" = {("Casillas", "Portero"), ("Kluivert",
                        "Delantero"), ...}
"Jugador"."fuerza" = {("Figo", 7), ("Rivaldo", 8), ...}
"Jugador"."velocidad" = {("Figo", 8), ("Rivaldo", 9), ...}
"Jugador"."resistencia" = {("Figo", 8), ("Rivaldo", 7), ...}
"Jugador"."estado" = {("Figo", "bien"), ("Rivaldo", "lesionado"),
                      ("Casillas", "bien"), ...}
"Equipo"."nombre" = {("Real Madrid", "Real Madrid Club de Fútbol"),
                     "Barcelona", "Fútbol Club Barcelona"), ...}
"Equipo"."categoría" = {("Barcelona", "Primera"), ("Alcalá", "Tercera"),
                        ...}
"Alineación"."nombre" = {("RM 4-4-2", "4-4-2 con doble pivote"), ...}
"Fecha"."día" = {("16-10-1969", 16), ...}
"Fecha"."mes" = {("16-10-1969", 10), ...}
"Fecha"."año" = {("16-10-1969", 1969), ...}
```

De la misma forma que ocurre con las instancias de concepto, las instancias de atributo también tienen un tiempo de vida asociado.

Se define **tiempo de vida** de una instancia de atributo como una función del universo de instancias de atributos (*IAT*) en el producto cartesiano de los números naturales consigo mismos ($N \times N$) de forma que esta función devuelve un par formado por el instante de creación y el instante de destrucción de la instancia de atributo:

$$TV_{IAT}: IAT \rightarrow N \times N$$

Para toda instancia de atributo deben cumplirse las siguientes restricciones con respecto al valor del tiempo de vida:

- El instante de creación debe ser menor que el instante de destrucción.
- El instante de creación de la instancia de atributo debe ser mayor o igual que el inicio de pertenencia de la instancia de concepto al concepto correspondiente.
- El instante de destrucción de la instancia de atributo debe ser menor o igual que el fin de pertenencia de la instancia al concepto.

Expresado formalmente:

$$\forall AT_i \in AT \bullet \forall IAT_i \in AT_i \bullet TV_{IAT}(IAT_i)[1] \leq TV_{IAT}(IAT_i)[2] \wedge TV_{IAT}(IAT_i)[1] \geq TP_C(C_{AT}(AT_i), IAT_i[1])[1] \wedge TV_{IAT}(IAT_i)[2] \leq TP_C(C_{AT}(AT_i), IAT_i[1])[2]$$

Esta función de tiempo de vida permite definir el **estado de un atributo de concepto en un instante de tiempo determinado** como una función entre el producto cartesiano del universo de atributos de concepto con los números naturales y las partes del universo de instancias de atributo, de forma que dado un atributo de concepto y un instante de tiempo devuelve el conjunto de sus instancias en ese momento:

$$ST_{AT}: AT \times N \rightarrow \Pi(IAT)$$

Una instancia de un atributo pertenece al estado de ese atributo en un instante determinado si y sólo si el tiempo de vida de esa instancia incluye el instante considerado:

$$\forall t_i \in N \bullet \forall AT_i \in AT \bullet \forall IAT_i \in AT_i \bullet IAT_i \in ST_{AT}(AT_i, t_i) \Leftrightarrow (TV_{IAT}(IAT_i)[1] \leq t_i \wedge TP_{IAT}(IAT_i)[2] \geq t_i)$$

A partir de esta definición de estado de atributo de concepto y teniendo en cuenta la definición de estado de concepto, ya pueden formalizarse las restricciones de grado, límite y valor por omisión:

- Grado de participación en atributo de concepto:

  $\forall t_i \in N \bullet \forall AT_i \in AT$, *se cumple:*

  $G_{AT}(AT_i) =$ *"obligatorio"* $\Leftrightarrow \forall I_j \in ST_C(C_{AT}(AT_i), t_i) \bullet \exists V_k \in T_{AT}(AT_i) \bullet (I_j, V_k) \in ST_{AT}(AT_i, t_i)$

  $G_{AT}(AT_i) =$ *"opcional"* $\Leftarrow \exists I_j \in ST_C(C_{AT}(AT_i), t_i) \bullet \forall V_k \in T_{AT}(AT_i) \bullet (I_j, V_k) \notin ST_{AT}(AT_i, t_i)$

- Límite de participación en atributo de concepto:

  $\forall t_i \in N \bullet \forall AT_i \in AT$, *se cumple:*

  $L_{AT}(AT_i) =$ *"único"* $\Leftrightarrow \forall I_j \in ST_C(C_{AT}(AT_i), t_i) \bullet (\exists V_k \in T_{AT}(AT_i) \bullet (I_j, V_k) \in ST_{AT}(AT_i, t_i) \Rightarrow (\forall V_l \in T_{AT}(AT_i) \bullet V_l \neq V_k \Rightarrow (I_j, V_l) \notin ST_{AT}(AT_i, t_i)))$

  $L_{AT}(AT_i) =$ *"múltiple"* $\Leftarrow \exists I_j \in ST_C(C_{AT}(AT_i), t_i) \bullet \exists V_k, V_l \in T_{AT}(AT_i) \bullet V_l \neq V_k \wedge (I_j, V_k) \in ST_{AT}(AT_i, t_i) \wedge (I_j, V_l) \in ST_{AT}(AT_i, t_i)$

- Valor por omisión de atributo de concepto:

  $\forall t_i \in N \bullet \forall AT_i \in AT \bullet VO_{AT}(AT_i) \neq \varnothing \Rightarrow (\forall I_j \in ST_C(C_{AT}(AT_i), t_i) \bullet (\exists V_k \in T_{AT}(AT_i) \bullet V_k \neq VO_{AT}(AT_i) \wedge (I_j, V_k) \in ST_{AT}(AT_i, t_i)) \vee (I_j, VO_{AT}(AT_i)) \in ST_{AT}(AT_i, t_i))$

4.2.1.4 Formalización de Asociaciones

El **universo de asociaciones** (AS) es el conjunto de todas las asociaciones que pueden definirse. Cada una de ellas se define formalmente mediante una séptupla formada por su nombre, su dimensión, su origen, su destino, su completitud, su funcionalidad y su invariante:

$$AS = \{AS_i \mid AS_i \cong (N_{AS}(AS_i), DIM_{AS}(AS_i), OR_{AS}(AS_i), DES_{AS}(AS_i), CPL_{AS}(AS_i), FUN_{AS}(AS_i), INV_{AS}(AS_i)\}$$

- $\mathbf{N_{AS}}$ (**Nombre** de asociación) es una aplicación entre el conjunto de las asociaciones (AS) y el tipo que representa las cadenas de caracteres (*CAD*), de forma que a cada asociación se le asocia una cadena:

  $$N_{AS}: AS \rightarrow CAD$$

- $\mathbf{DIM_{AS}}$ (**Dimensión** de asociación) es una aplicación entre el conjunto de las asociaciones (AS) y el tipo que representa los números naturales (*N*), de forma que a cada asociación se le asocia un número natural que representa el número de participantes (conceptos o asociaciones) en la asociación:

  $$DIM_{AS}: AS \rightarrow N$$

Toda asociación se define al menos entre dos participantes, por lo que la dimensión de una asociación será siempre mayor o igual que dos:

$$\forall AS_i \in AS \bullet DIM_{AS}(AS_i) \geq 2$$

- **OR_{AS}** (**Origen** de asociación) es una aplicación entre el universo de asociaciones (AS) y la potencia n-1 (siendo $n = DIM_{AS}(AS_i)$) del conjunto de roles de asociación (RL^{n-1}), que asocia a cada asociación una tupla (conjunto ordenado) con los roles que definen el origen de la asociación.

$$OR_{AS}: AS \rightarrow RL^{n-1}, \text{ siendo } n = DIM_{AS}(AS_i)$$

Por lo tanto el origen de una asociación será un único participante (cuando la dimensión es 2) o bien una tupla de participantes (con dimensión mayor que 2).

Así, se tiene:

$$\forall AS_i \in AS \bullet OR_{AS}(AS_i) = (RL_1, RL_2, ..., RL_{n-1}), \text{ siendo } n = DIM_{AS}(AS_i)$$

RL es el **universo de roles**. Cada **rol** de una asociación es una cuaterna formada por un nombre, un participante, su grado de participación y el límite de su participación en la asociación.

$$RL = \{RL_i \mid RL_i = (N_{RL}(RL_i), P_{RL}(RL_i), G_{RL}(RL_i), L_{RL}(RL_i))\}$$

  - N_{RL} (**Nombre del rol**) es una aplicación que asocia un nombre a cada rol. Este nombre servirá para distinguir cada rol de una asociación.

$$N_{RL}: RL \rightarrow CAD$$

    Por lo tanto un rol tiene un nombre que es independiente del nombre del concepto o asociación participante. Esto es necesario para distinguir roles cuando en una asociación hay varios participantes con el mismo concepto o asociación.

    Dentro de una asociación no podrán existir dos roles (tanto en el origen como en el destino) con el mismo nombre:

$$\forall AS_i \in AS \bullet \forall RL_j, RL_k \in OR_{AS}(AS_i) \cup \{DES_{AS}(AS_i)\} \bullet$$

$$RL_j \neq RL_k \Leftrightarrow N_{RL}(RL_j) \neq N_{RL}(RL_k)$$

  - P_{RL} (**Participante del rol**) es una aplicación que asocia un tipo, un concepto o una asociación a cada rol de la asociación.

$$P_{RL}: RL \rightarrow T \cup C \cup AS$$

    Dentro de una asociación ninguno de sus participantes podrá ser ella misma:

$$\forall AS_i \in AS \bullet \forall RL_j \in OR_{AS}(AS_i) \cup \{DES_{AS}(AS_i)\} \bullet P_{RL}(RL_j) \neq AS_i$$

  - G_{RL} (**Grado de participación** del rol). Es una aplicación entre el conjunto de roles (*RL*) y el conjunto de los grados de participación (*GR*), de forma que a cada rol se le asocia un único grado de participación:

$$G_R: R \rightarrow GR$$

    Si el valor de $G_{RL}(RL_j)$ es "obligatorio", indica que todo elemento del participante asignado al rol RL_j deberá aparecer al menos una vez en las

instancias de la asociación. En otro caso, podrán existir elementos del participante que no aparezcan en las instancias de la asociación. La formalización de esta condición queda pendiente de definir el estado de una asociación.

- L_{RL} (**Límite de participación** del rol). Es una aplicación entre el conjunto de roles (*RL*) y el conjunto de los límites de participación (*LI*), de forma que a cada rol se le asocia un único límite de participación (restricción sobre el número máximo de apariciones de una instancia genérica en las instancias de la asociación):

$$L_{RL}: RL \rightarrow LI$$

Si el límite de participación es "único", indica que todo elemento del participante correspondiente aparecerá como máximo una vez en las instancias de la asociación. En otro caso, podrán existir elementos del participante que aparezcan más de una vez. La formalización de esta condición queda pendiente de definir el estado de una asociación.

Para todas las asociaciones de un modelo estructural debe cumplirse que los participantes de su origen también pertenezcan al modelo estructural.

$$\forall EST_i \in EST \bullet \forall AS_i \in AS_{EST}(EST_i) \bullet \forall RL_i \in OR_{AS}(AS_i) \bullet P_{RL}(RL_i) \in T_{EST}(EST_i) \cup C_{EST}(EST_i) \cup AS_{EST}(EST_i)$$

Como notación se va a definir $POR_{AS}(AS_i)$ como la tupla formada por los **participantes** (conceptos o asociaciones) correspondientes a los roles que forman el origen de la asociación.

$$\forall AS_i \in AS \bullet \forall i \in AS \bullet i>0 \wedge i< DIM_{AS}(AS_i) \Rightarrow POR_{AS}(AS_i)[i] = P_{RL}(OR_{AS}(AS_i)[i])$$

Una vez definido el origen de una asociación, se puede indicar una restricción para los nombres de asociación. El nombre debe ser **único** para todas las asociaciones de un modelo estructural que tengan la misma tupla de participantes como origen de asociación:

$$\forall EST_i \in EST \bullet \forall AS_i, AS_j \in AS_{EST}(EST_i) \bullet AS_i \neq AS_j \wedge POR_{AS}(AS_i) = POR_{AS}(AS_j) \Rightarrow N_{AS}(AS_i) \neq N_{AS}(ASj)$$

Teniendo en cuenta esta restricción, para identificar unívocamente a una asociación será necesario utilizar el identificador de cada uno de sus participantes en el origen, más el propio nombre de la asociación.

- **DES_{AS}** (**Destino** de asociación) es una aplicación entre el conjunto de las asociaciones y el conjunto de los roles de asociación (RL), de forma que a cada asociación se le asigna un tipo, concepto o asociación como último participante.

$$DES_{AS}: AS \rightarrow RL$$

- **CPL_{AS}** (**Completitud** de asociación) es una aplicación entre el conjunto de las asociaciones y el conjunto de los valores de completitud (CPL), de forma que a cada asociación se le asigna un valor de completitud.

$$CPL_{AS}: AS \rightarrow CPL$$

CPL es el conjunto de completitudes de asociación, que contiene dos valores, "completa", "parcial", con el significado visto en el apartado de definiciones:

$$CPL = \{\text{“completa”}, \text{“parcial”}\}$$

La formalización de esta condición queda pendiente de definir el estado de una asociación.

- **FUN_{AS}** (**Funcionalidad** de asociación) es una aplicación entre el conjunto de las asociaciones y el conjunto de los valores de funcionalidad (FUN), de forma que a cada asociación se le asigna un valor de funcionalidad.

$$FUN_{AS}: AS \rightarrow FUN$$

FUN es el conjunto de funcionalidades de asociación que indica si la asociación es función, función inyectiva, etc. mediante cinco valores con el significado visto en el apartado de definiciones:

$$FUN = \{\text{“función”}, \text{“inyectiva”}, \text{“sobreyectiva”}, \text{“biyectiva”}, \text{“no función”}\}$$

La formalización de esta condición queda pendiente de definir el estado de una asociación.

- **INV_{AS}** (**Invariante** de asociación) es la condición que deben cumplir todas las instancias de la asociación. Es una aplicación inyectiva entre el conjunto de asociaciones (*AS*) y el universo de condiciones lógicas (CND), de forma que a cada asociación se le asigna su invariante:

$$INV_{AS}: AS \rightarrow CND$$

En todo instante de tiempo se cumple que una tupla pertenece a una asociación si y solo si la evaluación de la invariante de la asociación para esa tupla en ese instante de tiempo es cierta. Esta condición se formalizará en el apartado dedicado a formalizar las condiciones lógicas y su evaluación.

Equipo de fútbol

Seguidamente se recogen las asociaciones existentes:

- Nacimiento:

```
AS("Nacimiento", 2, (("persona", "Persona", "obligatorio", "único")),
("fecha", "Fecha", "opcional", "múltiple"), "completa", "función", -)
```

- Pertenencia:

```
AS("Pertenencia", 3, (("persona","Persona","opcional","múltiple"),
("fecha comienzo", "Fecha", "opcional", "múltiple")), ("equipo",
"Equipo", "obligatorio", "múltiple"), "parcial", "sobreyectiva", "Una
persona puede aparecer varias veces pero con fechas distintas")
```

- Fin de pertenencia:

```
AS("Fin de pertenencia", 2, ("pertenencia", "(Persona,
Fecha).Pertenencia", "opcional", "único"), ("fecha fin", "Fecha",
"opcional", "múltiple"), "parcial", "función", -)
```

- Convocatoria:

```
AS("Convocatoria", 3, (("equipo", "Equipo", "obligatorio", "múltiple"),
("fecha", "Fecha", "opcional", "múltiple")), ("alineación", "Alineación",
"obligatorio", "único"), "parcial", "biyectiva", -)
```

- Táctica prevista:

```
AS("Táctica prevista", 2, (("alineación", "Alineación", "obligatorio",
"único")), ("táctica elegida", "Táctica", "opcional", "múltiple"),
"completa", "función", -)
```

- Puestos:

```
AS("Puestos", 2, (("táctica", "Táctica", "obligatorio", "múltiple")),
("puesto elegido", "Puesto", "obligatorio", "múltiple"), "completa", "no
función", "Para cada táctica habrá 11 puestos de titulares (con 1
portero) y 5 suplentes (con 1 portero)")
```

- Asignación:

```
AS("Asignación", 3, (("alineación", "Alineación", "obligatorio",
"múltiple"), ("puesto asignado", "Puesto", "opcional", "múltiple")),
("jugador convocado", "Jugador", "opcional", "múltiple"), "parcial",
"función", "El jugador deberá pertenecer al equipo asociado a la
alineación y un jugador sólo podrá aparecer una vez en cada alineación y
un puesto sólo podrá aparecer una vez en cada alineación y cada puesto
deberá pertenecer a la táctica prevista en la alineación y no puede haber
más de 3 extranjeros titulares")
```

Teniendo en cuenta todas las definiciones realizadas sobre asociaciones, se puede indicar formalmente cómo están definidas las instancias de una asociación.

Para ello lo primero es definir un término que agrupe a valores de tipo, instancias de concepto e instancias de asociación.

Se define el **universo de instancias genéricas** (IG) de un modelo estructural como la unión del universo de valores, del universo de instancias de concepto y del universo de instancias de asociación:

$$IG = V \cup I \cup IAS$$

Por lo tanto, cada instancia genérica (IG_i), elemento del universo de instancias genéricas, será un valor, una instancia de concepto o bien una instancia de asociación.

El **universo de instancias de asociación** es el conjunto de todas las instancias de todas las asociaciones. Cada una es una tupla de instancias genéricas con el mismo número de elementos que la dimensión de la asociación a la que pertenece. Cada elemento de la tupla, excepto el último, pertenece al participante correspondiente al rol que ocupa su misma posición. La última instancia de la tupla pertenece al participante destino de la asociación:

$$IAS = \{IAS_i\}$$

$$AS_i = \{IAS_j \in IAS \mid IAS_j = (IG_1, IG_2, ..., IG_n),\ n = DIM_{AS}(AS_i),\ IG_k \in P_{RL}(OR_{AS}(AS_i)[k]),\ 1 \le k \le n\text{-}1,\ IG_n \in P_{RL}(DES_{AS}(AS_i))\}$$

Equipo de fútbol

Seguidamente se recogen ejemplos de instancias de asociación:

- **Nacimiento** (en las fechas se va a poner como nombre una representación de la misma).

```
AS"Nacimiento" = {("Loïc", "16.10.1969"), ...}
```

- **Pertenencia**

```
AS"Pertenencia" = {("Rivaldo", "1.8.1998" "Barcelona"), ("Figo",
"1.8.2000", "Real Madrid"), ...}
```

- **Fin de pertenencia**

AS"Fin de Pertenencia" = {(("Rivaldo", "1.8.1998" "Barcelona"), "1.8.2002"), ...}

- **Convocatoria**

AS"Convocatoria" = {("Real Madrid", "7.9.2002", "RM Habitual"), ("Barcelona", "29.9.2002", "B Especial"), ...}

- **Táctica prevista**

AS"Táctica prevista" = {("RM Habitual", "4-4-2"), ("B Especial", "4-3-3"), ...}

- **Puestos**

AS"Puestos" = {("4-4-2", "Portero titular"), ("4-4-2", "Interior derecho titular"), ...}

- **Asignación**

AS"Asignación" = {("RM Habitual", "Portero titular", "Casillas"), ("B Especial", "Delantero centro titular", "Kluivert"), ...}

De la misma forma que ocurre con las instancias de concepto, las instancias de asociación también tienen un tiempo de vida asociado.

Se define **tiempo de vida** de una instancia de asociación como una función del universo de instancias de asociación (*IAS*) en el producto cartesiano de los números naturales consigo mismos ($N \times N$) de forma que esta función devuelve un par formado por el instante de creación y el instante de destrucción de la instancia de asociación:

$$TV_{IAS}: IAS \rightarrow N \times N$$

Para toda instancia de asociación deben cumplirse las siguientes restricciones con respecto al valor del tiempo de vida:

- El instante de creación debe ser menor que el instante de destrucción.

$$\forall IAS_i \in IAS \bullet TV_{IAS}(IAS_i)[1] \leq TV_{IAS}(IAS_i)[2]$$

- El instante de creación de la instancia de asociación debe ser mayor o igual que el máximo de los siguientes valores:
  - Inicio de pertenencia de instancias de concepto al concepto correspondiente, para cada participante de tipo concepto.
  - Inicio de pertenencia de instancias de asociación a la asociación correspondiente, para cada participante de tipo asociación.
  - 0, para cada participante que represente un tipo.
- El instante de destrucción de la instancia de asociación debe ser menor o igual que el mínimo de los siguientes valores:
  - Fin de pertenencia de instancias de concepto al concepto correspondiente, para cada participante de tipo concepto.
  - Fin de pertenencia de instancias de asociación a la asociación correspondiente, para cada participante de tipo asociación.
  - 0, para cada participante que represente un tipo.

Para expresar formalmente estas condiciones se van a definir algunas funciones auxiliares:

- **Rol que ocupa una posición de una asociación**.

$$RL_{AS}: AS \times N \mapsto RL$$

$$\forall AS_i \in AS \bullet \forall k \in N \bullet (k < DIM_{AS}(AS_i) \wedge RL_{AS}(AS_i, k) = OR_{AS}(AS_i)[i]) \vee (k = DIM_{AS}(AS_i) \wedge RL_{AS}(AS_i, k) = DES_{AS}(AS_i))$$

- **Participante correspondiente a una posición dentro de una asociación**:

$$P_{AS}: AS \times N \mapsto T \cup C \cup AS$$

$$\forall AS_i \in AS \bullet \forall k \in N \bullet (k < DIM_{AS}(AS_i) \wedge P_{AS}(AS_i, k) = POR_{AS}(AS_i)[i]) \vee (k = DIM_{AS}(AS_i) \wedge P_{AS}(AS_i, k) = P_{RL}(DES_{AS}(AS_i)))$$

- **Instante de tiempo inicial correspondiente a un participante** de una instancia de asociación:

$$IC_{IAS}: IAS \times N \to N$$

$$\forall AS_i \in AS \bullet \forall IAS_i \in AS_i \bullet \forall k \in N \bullet k \leq DIM_{AS}(AS_i) \Rightarrow (IAS_i[k] \in V \wedge IC_{IAS}(IAS_i, k) = 0) \vee (IAS_i[k] \in I \wedge IC_{IAS}(IAS_i, k) = TP_C(P_{AS}(AS_i, k), IAS_i[k])[1]) \vee (IAS_i[k] \in IAS \wedge IC_{IAS}(IAS_i, k) = TP_{AS}(P_{AS}(AS_i, k), IAS_i[k])[1])$$

- **Instante de tiempo final correspondiente a un participante** de una instancia de asociación:

$$IF_{IAS}: IAS \times N \to N$$

$$\forall AS_i \in AS \bullet \forall IAS_i \in AS_i \bullet \forall k \in N \bullet k \leq DIM_{AS}(AS_i) \Rightarrow (IAS_i[k] \in V \wedge IF_{IAS}(IAS_i, k) = \infty) \vee (IAS_i[k] \in I \wedge IF_{IAS}(IAS_i, k) = TP_C(P_{AS}(AS_i, k), IAS_i[k])[2]) \vee (IAS_i[k] \in IAS \wedge IF_{IAS}(IAS_i, k) = TP_{AS}(P_{AS}(AS_i, k), IAS_i[k])[2])$$

Ahora se puede expresar formalmente la restricción sobre el tiempo de creación y destrucción de una instancia de asociación respecto a sus elementos:

$$\forall AS_i \in AS \bullet \forall IAS_i \in IAS \bullet TV_{IAS}(IAS_i)[1] \geq max(IC_{IAS}(IAS_i, k), k \in N, 0 \leq k \leq DIM_{AS}(AS_i)) \wedge TV_{IAT}(AT_i)[2] \leq min(IF_{IAS}(IAS_i, k), k \in N, 0 \leq k \leq DIM_{AS}(AS_i))$$

Se define el **tiempo de pertenencia de una tupla a una asociación** como una función parcial del producto cartesiano entre asociaciones e instancias de asociación ($AS \times IAS$) con el producto cartesiano de los números naturales consigo mismos ($N \times N$), de forma que, si una tupla pertenece (o ha pertenecido alguna vez) a una asociación, esta función devuelve un par formado por el instante de tiempo en el que la instancia empezó a formar parte de la asociación (inicio de pertenencia) y el instante en el que la instancia terminó de formar parte de la asociación (fin de pertenencia):

$$TP_{AS}: AS \times IAS \mapsto N \times N$$

Si una instancia pertenece a una asociación, deben cumplirse las siguientes restricciones con respecto al valor del tiempo de pertenencia:

- El inicio de pertenencia debe ser menor que el final de pertenencia.
- El inicio de pertenencia debe ser mayor o igual que el tiempo de creación de la instancia de asociación.
- El fin de pertenencia debe ser menor o igual que el tiempo de destrucción de la instancia de asociación.

$$\forall AS_i \in AS \bullet \forall IAS_i \in IAS \bullet IAS_i \in AS_i \Rightarrow (TP_{AS}(AS_i, IAS_i)[1] \leq TP_{AS}(AS_i, IAS_i)[2] \wedge TP_{AS}(AS_i, IAS_i)[1] \geq TV_{IAS}(IAS_i)[1] \wedge TP_C(C_i, I_i)[2] \leq TV_{IAS}(IAS_i)[2])$$

Esta función de tiempo de pertenencia permite definir el **estado de una asociación en un instante de tiempo determinado** como una función entre el producto cartesiano del universo de asociaciones con los números naturales y las partes del universo de instancias de asociación, de forma que dado una asociación y un instante de tiempo devuelve el conjunto de sus instancias en ese momento:

$$ST_{AS}: AS \times N \rightarrow \Pi(IAS)$$

Una instancia de una asociación pertenece al estado de esa asociación en un instante determinado si y sólo si el tiempo de pertenencia de esa instancia a la asociación incluye el instante considerado (lo cual quiere decir que también el tiempo de vida de la instancia de asociación incluye al instante de tiempo considerado):

$$\forall t_i \in N \bullet \forall AS_i \in AS \bullet \forall IAS_i \in AS_i \bullet IAS_i \in ST_{AS}(AS_i) \Leftrightarrow (TP_{AS}(AS_i, IAS_i)[1] \leq t_i \wedge TP_{AS}(AS_i, IAS_i)[2] \geq t_i)$$

Una vez definidos los estados de un concepto y de una asociación, se puede definir el **estado de uno de los participantes de una asociación en un instante de tiempo** como una función del producto cartesiano de asociaciones, naturales (posiciones dentro de tupla) y naturales (tiempo) con la unión de las partes de valores, partes de instancias y partes de instancias de asociación, de la forma siguiente:

- Si el participante es un tipo, entonces será directamente ese tipo.
- Si el participante es un concepto, entonces será el estado de ese concepto.
- Si el participante es una asociación, entonces será el estado de esa asociación.

$$STP_{AS}: AS \times N \times N \rightarrow \Pi(V) \cup \Pi(I) \cup \Pi(IAS)$$

$$\forall AS_i \in AS \bullet \forall k \in N \bullet \forall t_i \in N \bullet k \leq DIM_{AS}(AS_i) \Rightarrow (P_{AS}(AS_i, k) \in T \wedge STP_{AS}(AS_i, k, t_i) = P_{AS}(AS_i, k)) \vee (P_{AS}(AS_i, k) \in C \wedge STP_{AS}(AS_i, k, t_i) = ST_C(P_{AS}(AS_i, k), t_i)) \vee (P_{AS}(AS_i, k) \in AS \wedge STP_{AS}(AS_i, k, t_i) = ST_{AS}(P_{AS}(AS_i, k), t_i))$$

Con todas las definiciones anteriores se pueden formalizar las restricciones de grado, límite, completitud y funcionalidad:

- *Grado de un rol*. Si es obligatorio entonces implica que para todo instante de tiempo y para todo elemento del estado del participante del rol debe existir una tupla perteneciente al estado de la asociación tal que ese elemento participa en la tupla en el lugar que le corresponde. En otro caso no hay ninguna limitación.

  $$\forall t_i \in N \bullet \forall AS_i \in AS \bullet \forall k \in N \bullet 1 \leq k \leq DIM_{AS}(AS_i) \wedge RL_i = RL_{AS}(AS_i, k)$$

  $$G_{RL}(RL_i) = \text{"obligatorio"} \Leftrightarrow (\forall IG_i \in STP_{AS}(AS_i, k, t_i) \bullet \exists IAS_i \in ST_{AS}(AS_i, t_i) \bullet IG_i = IAS_i[k])$$

- *Límite de un rol*. Si es único en cada instante de tiempo y para cada elemento del participante del rol como máximo puede existir una tupla con ese elemento. En otro caso no hay ninguna limitación.

  $$\forall t_i \in N \bullet \forall AS_i \in AS \bullet \forall k \in N \bullet 1 \leq k \leq DIM_{AS}(AS_i) \wedge RL_i = RL_{AS}(AS_i, k)$$

  $$L_{RL}(RL_i) = \text{"obligatorio"} \Leftrightarrow (\forall IG_i \in STP_{AS}(AS_i, k, t_i) \bullet (\exists IAS_i \in ST_{AS}(AS_i, t_i) \bullet IG_i = IAS_i[k] \Rightarrow \forall IAS_j \in ST_{AS}(AS_i, t_i) \bullet IAS_i \neq IAS_j \Rightarrow IG_i \neq IAS_j[k]))$$

- *Completitud de una asociación*. Si es completa, para todo instante de tiempo y para toda combinación de elementos de los estados de los participantes del origen debe existir al menos un elemento del estado del participante destino tal que la tupla formada por todos esos elementos pertenece al estado de la asociación. En otro caso no hay restricción.

  $\forall t_i \in N \bullet \forall AS_i \in AS \bullet n = DIM_{AS}(AS_i)$

  $CPL_{AS}(AS_i)=\text{"completa"} \Leftrightarrow (\forall IG_1 \in STP_{AS}(AS_i, 1, t_i) \bullet ... \bullet \forall IG_{n-1} \in STP_{AS}(AS_i, n-1, t_i) \bullet \exists IG_n \in STP_{AS}(AS_i, n, t_i) \bullet \exists IAS_i \in ST_{AS}(AS_i, t_i) \bullet \forall k \in N \bullet 1 \leq k \leq n \Rightarrow IG_k = IAS_i[k])$

- *Funcionalidad de una asociación*. Afecta a cualquier combinación de elementos de los estados de los participantes que aparezcan en el estado de la asociación.
  - "función": si para cualquier tupla de elementos del origen que tiene asociado un elemento del destino, ese elemento es único (es decir, no tiene dos o más elementos asociados del destino)
  - "inyectiva": si es función y, además, dadas dos tuplas diferentes de elementos del origen pertenecientes a la asociación, sus elementos del destino correspondientes son siempre diferentes
  - "sobreyectiva": si es función y, además, para todo elemento del destino existe al menos una tupla de elementos del origen relacionada
  - "biyectiva": si es función inyectiva y sobreyectiva
  - "no función": si no se cumplen ninguna de las condiciones anteriores).

4.2.1.5 Formalización de Atributos de Asociación

El **universo de atributos** de concepto (ATS) es el conjunto formado por atributos de concepto, que se especifican mediante seis funciones: nombre, asociación, tipo, grado, límite y valor por omisión:

$$ATS = \{ATS_i \subset IAS \times V \mid ATS_i \cong (N_{ATS}(ATS_i), AS_{ATS}(ATS_i), T_{ATS}(ATS_i), G_{ATS}(ATS_i), L_{ATS}(ATS_i), VO_{ATS}(ATS_i))\}$$

- **N_{ATS}** (**Nombre** de atributo de asociación) es una aplicación entre el conjunto de atributos de asociación (*ATS*) y el conjunto de las cadenas (*CAD*), de forma que a cada atributo se le asocia una cadena:

  $$N_{ATS}: ATS \rightarrow CAD$$

- **AS_{ATS}** (**Asociación** de atributo) es una aplicación entre el conjunto de atributos de asociación (*ATS*) y el conjunto de las asociaciones (*AS*), de forma que a cada atributo se le asigna una asociación (dicho de otro modo, el atributo "pertenece" a una asociación):

  $$AS_{ATS}: ATS \rightarrow AS$$

  Para todos los atributos de un modelo estructural debe cumplirse que su asociación también pertenezca al modelo estructural.

  $$\forall EST_i \in EST \bullet \forall ATS \in ATS_{EST}(EST_i) \bullet AS_{ATS}(ATS_i) \in AS_{EST}(EST_i)$$

El nombre de un atributo debe ser único para todos los atributos de una misma asociación:

$$\forall ATS_i, ATS_j \in ATS \bullet ATS_i \neq ATS_j \wedge AS_{ATS}(ATS_i) = AS_{ATS}(ATS_j) \Rightarrow N_{ATS}(ATS_i) \neq N_{ATS}(ATS_j)$$

De esta forma para identificar unívocamente a un atributo será necesario utilizar el nombre del concepto al que pertenece, seguido por el propio nombre del atributo.

- **T_{ATS}** (**Tipo** de atributo de asociación) es una aplicación entre el conjunto de atributos de asociación (*ATS*) y el conjunto de los tipos (*T*), de forma que a cada atributo se le asocia un único tipo:

$$T_{ATS}: ATS \rightarrow T$$

Para todos los atributos de asociación de un modelo estructural debe cumplirse que su tipo también pertenezca al modelo estructural.

$$\forall EST_i \in EST \bullet \forall ATS_i \in ATS_{EST}(EST_i) \bullet T_{ATS}(ATS_i) \in T_{EST}(EST_i)$$

Dados la asociación y el tipo de un atributo, el atributo será en realidad un subconjunto de su producto cartesiano:

$$\forall ATS_i \in ATS \bullet ATS_i \subset AS_{ATS}(ATS_i) \times T_{ATS}(ATS_i)$$

- **G_{ATS}** (**Grado** de atributo de asociación) es una aplicación entre el conjunto de atributos (*ATS*) y el conjunto de los grados de participación (*GR*), de forma que a cada atributo se le asocia un grado:

$$G_{ATS}: ATS \rightarrow GR$$

Si el valor del grado de un atributo es "obligatorio", indica que en cada instante de tiempo toda instancia que pertenezca al estado de la asociación deberá tener al menos un valor asignado en el atributo en ese instante de tiempo. En otro caso, podrán existir instancias de la asociación que no tengan valor asociado. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

- **L_{ATS}** (**Límite** de atributo) es una aplicación entre el conjunto de atributos de asociación (*ATS*) y el conjunto de los límites de participación (*LI*), de forma que a cada atributo se le asocia un límite:

$$L_{ATS}: ATS \rightarrow LI$$

Si el valor del límite de un atributo es "único", indica que en cualquier instante de tiempo toda instancia que pertenezca al estado de la asociación podrá tener como máximo un valor asignado en el atributo. En otro caso, podrán existir instancias de la asociación que tengan más de un valor asociado. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

La definición de límite para el número de valores de un atributo implica que existen atributos multivaluados, es decir, atributos que pueden tener varios valores asignados a una misma instancia de asociación.

- **VO_{ATS} (Valor por omisión** del atributo) es una aplicación que permite asignar un valor a un atributo de asociación para que sea utilizado por todas las instancias de la asociación que no tengan valor asignado en un instante dado.

  La existencia de valor por omisión en un atributo es opcional, por lo tanto VO_{ATS} es una aplicación entre el conjunto de atributos de asociación (*ATS*) y el conjunto de los valores (*V*) unido al conjunto formado por el conjunto vacío ($\{\varnothing\}$) de forma que a cada atributo se le puede asociar un valor o no:

  $$VO_{ATS}: ATS \rightarrow V \cup \{\varnothing\}$$

  Se debe cumplir que, si un atributo tiene asignado valor por omisión, entonces ese valor debe pertenecer al tipo del atributo:

  $$\forall ATS_i \in ATS \bullet VO_{ATS}(ATS_i) \neq \varnothing \Rightarrow VO_{ATS}(ATS_i) \in T_{ATS}(ATS_i)$$

  Si un atributo tiene definido valor por omisión, entonces ese valor será el asignado a todas las instancias del estado de la asociación mientras no se les asigne explícitamente un valor. La formalización de esta condición queda pendiente de definir el tiempo de vida de una instancia de atributo.

Equipo de fútbol

En la asociación "Pertenencia" entre equipo, fecha y persona, se define un atributo "**es socio**" que indica si la persona, además de pertenecer contractualmente al equipo es socio del mismo. Este atributo es de tipo boleando, tiene grado obligatorio, límite único y valor por omisión falso (normalmente la persona no es socio del equipo al que pertenece).

```
ATS("es socio", "Pertenencia", B, "obligatorio", "único", f)
```

Teniendo en cuenta todas las definiciones realizadas sobre atributos, se puede indicar formalmente cómo están definidas las instancias de un atributo de asociación.

El **universo de instancias de atributo de asociación** (IATS) es un conjunto cuyos elementos son pares ordenados formados por una instancia de asociación y un valor de un tipo.

$$IATS = \{IATS_i \mid IATS_i = (IAS_i, V_i), IAS_i \in IAS\ V_i \in I\}$$

Para un atributo concreto sus instancias deberán contener una instancia de la asociación del atributo y un valor del tipo del atributo:

$$ATS_i = \{IATS_j \in IATS \mid IATS_j = (IAS_k, V_l), IAS_k \in AS_{ATS}(ATS_i)\ V_l \in T_{ATS}(ATS_i)\}$$

Por lo tanto, toda instancia de un atributo pertenece al producto cartesiano de su concepto con su tipo:

$$\forall ATS_i \in ATS,\ \forall IATS_j \in ATS_i, IATS_j \in AS_{ATS}(ATS_i) \times T_{ATS}(ATS_i)$$

Equipo de fútbol

Seguidamente se muestran algunas instancias del atributo "es socio" de la asociación "Pertenencia":

```
(("Persona", "Fecha")."Pertenencia")."es socio" = {(("Rivaldo",
"1.8.1998" "Barcelona"), f), (("Figo", "1.8.2000", "Real Madrid"), f),
(("Raúl", "1.8.1995", "Real Madrid"), v),  ...}
```

De la misma forma que ocurre con las instancias de concepto, las instancias de atributo de asociación también tienen un tiempo de vida asociado.

Se define **tiempo de vida** de una instancia de atributo como una función del universo de instancias de atributos de asociación (*IATS*) en el producto cartesiano de los números naturales consigo mismos ($N \times N$) de forma que esta función devuelve un par formado por el instante de creación y el instante de destrucción de la instancia de atributo de asociación:

$$TV_{IATS}: IATS \rightarrow N \times N$$

Para toda instancia de atributo deben cumplirse las siguientes restricciones con respecto al valor del tiempo de vida:

- El instante de creación debe ser menor que el instante de destrucción.
- El instante de creación de la instancia de atributo debe ser mayor o igual que el inicio de pertenencia de la instancia de asociación a la asociación correspondiente.
- El instante de destrucción de la instancia de atributo debe ser menor o igual que el fin de pertenencia de la instancia a la asociación.

Expresado formalmente:

$$\forall ATS_i \in ATS \bullet \forall IATS_i \in ATS_i \bullet TV_{IATS}(IATS_i)[1] \leq TV_{IATS}(IATS_i)[2] \wedge TV_{IATS}(IATS_i)[1] \geq TP_{AS}(AS_{ATS}(ATS_i), IATS_i[1])[1] \wedge TV_{IATS}(IATS_i)[2] \leq TP_{AS}(AS_{ATS}(ATS_i), IATS_i[1])[2]$$

Esta función de tiempo de vida permite definir el **estado de un atributo de concepto en un instante de tiempo determinado** como una función entre el producto cartesiano del universo de atributos de asociación con los números naturales y las partes del universo de instancias de atributo de asociación, de forma que dado un atributo de asociación y un instante de tiempo devuelve el conjunto de sus instancias en ese momento:

$$ST_{ATS}: ATS \times N \rightarrow \Pi(IATS)$$

Una instancia de un atributo pertenece al estado de ese atributo en un instante determinado si y sólo si el tiempo de vida de esa instancia incluye el instante considerado:

$$\forall t_i \in N \bullet \forall ATS_i \in ATS \bullet \forall IATS_i \in ATS_i \bullet IATS_i \in ST_{ATS}(ATS_i, t_i) \Leftrightarrow (TV_{IATS}(IATS_i)[1] \leq t_i \wedge TP_{IATS}(IATS_i)[2] \geq t_i)$$

A partir de esta definición de estado de atributo de concepto y teniendo en cuenta la definición de estado de concepto, ya pueden formalizarse las restricciones de grado, límite y valor por omisión:

- Grado de participación en atributo de asociación:

  $\forall t_i \in N \bullet \forall ATS_i \in ATS$, *se cumple:*

  $G_{ATS}(ATS_i) =$ *"obligatorio"* $\Leftrightarrow \forall IAS_j \in ST_{AS}(AS_{ATS}(ATS_i), t_i) \bullet \exists V_k \in T_{ATS}(ATS_i) \bullet (IAS_j, V_k) \in ST_{ATS}(ATS_i, t_i)$

  $G_{ATS}(ATS_i) =$ *"opcional"* $\Leftarrow \exists IAS_j \in ST_{AS}(AS_{ATS}(ATS_i), t_i) \bullet \forall V_k \in T_{ATS}(ATS_i) \bullet (IAS_j, V_k) \notin ST_{ATS}(ATS_i, t_i)$

- Límite de participación en atributo de concepto:

  $\forall t_i \in N \bullet \forall ATS_i \in ATS$, *se cumple:*

  $L_{ATS}(ATS_i)$ = *"único"* $\Leftrightarrow \forall IAS_j \in ST_{AS}(AS_{ATS}(ATS_i), t_i) \bullet (\exists V_k \in T_{ATS}(ATS_i) \bullet (IAS_j, V_k) \in ST_{ATS}(ATS_i, t_i) \Rightarrow (\forall V_l \in T_{ATS}(ATS_i) \bullet V_l \neq V_k \Rightarrow (IAS_j, V_l) \notin ST_{ATS}(ATS_i, t_i)))$

  $L_{ATS}(ATS_i)$ = *"múltiple"* $\Leftarrow \exists IAS_j \in ST_{AS}(AS_{ATS}(ATS_i), t_i) \bullet \exists V_k, V_l \in T_{ATS}(ATS_i) \bullet V_l \neq V_k \wedge (IAS_j, V_k) \in ST_{ATS}(ATS_i, t_i) \wedge (IAS_j, V_l) \in ST_{ATS}(ATS_i, t_i)$

- Valor por omisión de atributo de concepto:

  $\forall t_i \in N \bullet \forall ATS_i \in ATS \bullet VO_{ATS}(ATS_i) \neq \varnothing \Rightarrow (\forall IAS_j \in ST_{AS}(AS_{ATS}(ATS_i), t_i) \bullet (\exists V_k \in T_{ATS}(ATS_i) \bullet V_k \neq VO_{ATS}(ATS_i) \wedge (IAS_j, V_k) \in ST_{ATS}(ATS_i, t_i)) \vee (IAS_j, VO_{ATS}(ATS_i)) \in ST_{ATS}(ATS_i, t_i))$

4.2.1.6 Formalización de Clasificaciones de Concepto

La formalización de las clasificaciones se ha divido por claridad en una serie de apartados, empezando por su definición. Después se definirán formalmente las especializaciones, que permiten restringir las propiedades de los subconceptos. Posteriormente se tratará la forma en la que la herencia afecta a restricciones que han aparecido hasta ahora y, por último, se definirán formalmente los antecesores, sucesores y familiares de un concepto.

4.2.1.6.1 Clasificaciones

El **universo de clasificaciones** (CL) contiene todas las posibles clasificaciones de concepto, cada una de las cuales es una quíntupla formada por el nombre de la clasificación, el superconcepto, el conjunto de subconceptos, la indicación de si es disjunta o no y la indicación de si es completa o no:

$$CL = \{CL_i \mid CL_i = (N_{CL}(CL_i), S_{CL}(CL_i), SB_{CL}(CL_i), DIS_{CL}(CL_i), COM_{CL}(CL_i))\}$$

- $\mathbf{N_{CL}}$ (**Nombre** de clasificación) es una aplicación entre el conjunto de clasificaciones (*CL*) y la unión entre el conjunto de las cadenas (*CAD*) y el conjunto cuyo único elemento es el conjunto vacío ($\{\varnothing\}$), de forma que a cada clasificación se le podrá asociar o no una cadena:

  $$N_{CL}: CL \rightarrow CAD \cup \{\varnothing\}$$

- $\mathbf{S_{CL}}$ (**Superconcepto** de clasificación) es una aplicación entre el conjunto de clasificaciones (*CL*) y el conjunto de los conceptos (*C*), de forma que a cada clasificación se le asocia un único concepto, que es su superconcepto:

  $$S_{CL}: CL \rightarrow C$$

  Para toda clasificación de un modelo estructural su superconcepto también debe pertenecer al modelo estructural.

  $$\forall EST_i \in EST \bullet \forall CL_i \in CL_{EST}(EST_i) \bullet S_{CL}(CL_i) \in C_{EST}(EST_i)$$

  Para todo concepto, si existen dos o más clasificaciones de las que es superconcepto, esas clasificaciones deben tener nombre asignado y ese nombre debe ser distinto en todas ellas:

$$\forall C_i \in C \bullet \exists CL_j \in CL \bullet \exists CL_k \in CL \bullet CL_j \neq CL_k \wedge C_i = S_{CL}(CL_j) \wedge C_i = S_{CL}(CL_k) \Rightarrow N_{CL}(CL_j) \neq \varnothing \wedge N_{CL}(CL_k) \neq \varnothing \wedge N_{CL}(CL_j) \neq N_{CL}(CL_k)$$

- **SB_{CL}** (**Subconceptos** de clasificación) es una aplicación inyectiva entre el conjunto de clasificaciones (*CL*) y las partes del conjunto de las definiciones de subconceptos (*Π(SB)*), de forma que a cada clasificación se le asocia un conjunto de definiciones de subconceptos:

  $$SB_{CL}: CL \rightarrow \Pi(SB)$$

  SB es el **universo de definiciones de subconcepto**. Cada **definición de subconcepto** está asociada a un concepto (el subconcepto) y define cómo cambia ese concepto respecto al superconcepto de la clasificación. Cada uno de los cambios se denomina especialización.

  $$SB = \{SB_i \mid SB_i = (C_{SB}(SB_i), ESP_{SB}(SB_i))\}$$

  - **C_{SB}** (**Concepto de definición** de subconcepto) es una aplicación entre el conjunto de definiciones de subconceptos (*SB*) y el universo de conceptos (*C*), de forma que a cada definición de subconcepto se le asocia un concepto (el subconcepto de esa definición).

    $$C_{SB}: SB \rightarrow C$$

  - **ESP_{SB}** (**Especializaciones de definición** de subconcepto) es una aplicación inyectiva entre el conjunto de definiciones de subconceptos (*SB*) y las partes del conjunto de especializaciones de subconcepto (*Π(ESB)*), de forma que a cada definición de subconcepto se le asocia un conjunto de especializaciones.

    $$ESP_{SB}: SB \rightarrow \Pi(ESB)$$

    La explicación detallada de las especializaciones es considerablemente extensa, por lo que se deja para más adelante (apartado "Especializaciones"), con el fin de no interrumpir la definición completa de clasificación.

  Para toda clasificación de un modelo estructural los conceptos asociados a sus definiciones de subconcepto también deben pertenecer al modelo estructural.

  $$\forall EST_i \in EST \bullet \forall CL_i \in CL_{EST}(EST_i) \bullet \forall SB_i \in SB_{CL}(CL_i) \bullet C_{SB}(SB_i) \in C_{EST}(EST_i)$$

  En relación con los subconceptos se deben cumplir dos condiciones:

  3. *Regla de inclusión:* para todo instante de tiempo el estado de cada subconcepto es un subconjunto del estado del superconcepto.

     $$\forall t_i \in N \bullet \forall CL_i \in CL \bullet \forall SB_j \in SB_{CL}(CL_i) \bullet C_k = S_{CL}(CL_i) \wedge C_l = C_{SB}(SB_j) \Rightarrow ST_C(C_l, t_i) \subset ST_C(C_k, t_i)$$

  4. *Regla de no circularidad*: ningún subconcepto debe ser igual al superconcepto o pertenecer al conjunto de sus antecesores (esta regla se podría derivar automáticamente de la regla de inclusión).

     $$\forall CL_i \in CL \bullet C_j = S_{CL}(CL_i) \Rightarrow \forall SB_k \in SB_{CL}(CL_i) \bullet C_{SB}(SB_k) \neq C_j \wedge C_{SB}(SB_k) \notin ANT_C(C_j)$$

     En esta regla se ha utilizado la aplicación **antecesores de concepto** (ANT_C) que representa todos los "padres", directos o no, de un concepto. Esta función se

definirá formalmente más adelante, en el apartado "Antecesores, Sucesores y Familiares".

- **DIS_{CL}** (Clasificación **disjunta**) es una aplicación entre el conjunto de clasificaciones (*CL*) y el conjunto de los valores booleanos (*B*), de forma que a cada clasificación se le asocia un valor booleano que indica si es disjunta o no:

$$DIS_{CL}: CL \rightarrow B$$

Si la clasificación no es disjunta (valor *f*), indica el estado de los subconceptos pueden solaparse, es decir, que pueden tener instancias en común. En otro caso, los estados de los subconceptos serán disjuntos.

$$\forall t_i \in N \bullet \forall CL_i \in CL \text{ se cumple:}$$

$$DIS_{CL}(CL_i) = v \Leftrightarrow \cap ST_C(C_{SB}(SB_j), t_i) = \varnothing, \ \forall SB_j \in SB_{CL}(CL_i)$$

$$DIS_{CL}(CL_i) = f \Leftrightarrow \cap ST_C(C_{SB}(SB_j), t_i) \neq \varnothing, \ \forall SB_j \in SB_{CL}(CL_i)$$

- **COM_{CL}** (Clasificación **completa**) es una aplicación entre el conjunto de clasificaciones (*CL*) y el conjunto de los valores booleanos (*B*) de forma que a cada clasificación se le asocia un valor booleano diciendo si es completa o no:

$$COM_{CL}: CL \rightarrow B$$

Si la clasificación es completa (valor *v*) indica que en todo instante de tiempo la unión de los estados de los subconceptos es exactamente igual al estado del superconcepto. En otro caso, la unión de los estados de los subconceptos será un subconjunto del estado del superconcepto.

$$\forall t_i \in N \bullet \forall CL_i \in CL \text{ se cumple:}$$

$$COM_{CL}(CL_i) = v \Leftrightarrow \cup ST_C(C_{SB}(SB_j), t_i) = ST_C(S_{CL}(CL_i), t_i), \ \forall SB_j \in SB_{CL}(CL_i)$$

$$COM_{CL}(CL_i) = f \Leftrightarrow \cup ST_C(C_{SB}(SB_j), t_i) \subset ST_C(S_{CL}(CL_i), t_i), \ \forall SB_j \in SB_{CL}(CL_i)$$

Equipo de fútbol

- Hay **tres tipos de personas**, en función de su **ocupación**: Técnicos, Directivos o Jugadores.

```
C("Técnico", ...)
C("Directivo", ...)
CL("por ocupación", "Persona", {("Técnico", ø), ("Directivo", ø),
("Jugador", ø)}, f, f)
```

- Hay **tres tipos de jugadores**, en función de su **país** de origen: nacionales, comunitarios, o extranjeros.

```
C("Nacional", ...)
C("Comunitario", ...)
C("Extranjero", ...)
CL("por país", "Jugador", {("Nacional", {... }), ("Comunitario", {...}),
("Extranjero", {...)}, v, v)
```

- Hay **dos tipos de jugadores** en función de su **posición** en el campo: porteros y jugadores de campo. Los porteros tienen como nuevos atributos evaluados sus reflejos, juego con el pie, paradas por alto,

paradas por bajo y salidas. Los jugadores de campo tienen como nuevos atributos su capacidad de cortar balones, de centrar, de regatear, de rematar con el pie y de rematar con la cabeza.

```
C("Jugador Campo", ...)
C("Portero", ...)
CL("por posición", "Jugador", {("Portero", {...}), ("Jugador campo", {...})}, v, v)
AT("reflejos", "Portero", "Valoración", "obligatorio", "único", NULO)
AT("pie", "Portero", "Valoración", "obligatorio", "único", NULO)
AT("alto", "Portero", "Valoración", "obligatorio", "único", NULO)
AT("bajo", "Portero", "Valoración", "obligatorio", "único", NULO)
AT("salidas", "Portero", "Valoración", "obligatorio", "único", NULO)
AT("robos", "Jugador Campo", "Valoración", "obligatorio", "único", NULO)
AT("centros", "Jugador Campo", "Valoración", "obligatorio", "único", NULO)
AT("regate", "Jugador Campo", "Valoración", "obligatorio", "único", NULO)
AT("remate pie", "Jugador Campo", "Valoración", "obligatorio", "único", NULO)
AT("remate cabeza", "Jugador Campo", "Valoración", "obligatorio", "único", NULO)
```

- Hay **dos tipos de puestos** dentro de una táctica: titulares o suplentes.

```
C("Titular", ...)
C("Suplente", ...)
CL("por titularidad", "Puesto", {("Titular", {...}), ("Suplente", {...})}, v, v)
```

En cuanto a las instancias en los conceptos "Persona", "Jugador", "Técnico", "Directivo", "Portero" y "Jugador campo", y teniendo en cuenta las clasificaciones anteriores, se podrán tener los siguientes casos:

- Una persona que sea jugador y técnico a la vez:

```
"Viali"∈"Jugador" y "Viali"∈"Técnico"
```

- Un jugador que sea jugador de campo y extranjero:

```
"Rivaldo"∈"Jugador Campo" y "Rivaldo"∈"Extranjero"
```

- Un jugador que sea portero y nacional:

```
"Casillas"∈"Portero" y "Casillas"∈"Nacional"
```

En cambio no podrá ocurrir que un jugador sea portero y de campo simultáneamente, así como tampoco podrá ser nacional y extranjero a la vez.

Después de formalizar las clasificaciones de concepto se puede volver sobre la herencia simple, la herencia múltiple y la pertenencia de una instancia a varios conceptos.

Para las afirmaciones siguientes se va a suponer que $CL_i \in CL$ es una clasificación, $C_s = S_{CL}(CL_i) \in C$ es el superconcepto de la clasificación y $C_b \in C$ es uno de los subconceptos de la clasificación (es decir, $\exists SB_j \in SB_{CL}(CL_i)$ *tal que* $C_b = C_{SB}(SB_j)$).

Sea $I_k \in ST_C(C_b, t_i)$ una instancia del subconcepto entonces, dado que $I_k \in ST_C(C_s, t_i)$:

- I_k puede aparecer en las instancias de asociación (n-tuplas) de cualquier asociación en la que participe C_s. Por lo tanto **C_b hereda la participación en asociaciones** definida para C_s.
- I_k puede aparecer en las instancias de cualquier atributo definido para C_s. Por lo tanto **C_b hereda las definiciones de atributos de C_s.**

Esta herencia de asociaciones y atributos se realiza de forma que los subconceptos pueden especializar la participación de asociaciones y la definición de atributos. Esto es así ya que un subconcepto es más especializado que el superconcepto y se pueden restringir más esas definiciones, tal y como se verá al definir formalmente los tipos de especialización en el apartado correspondiente.

Formalmente se dice que existe **herencia múltiple** si existe al menos un concepto tal que su conjunto de padres tiene más de un elemento:

$$herencia\ múltiple \Leftrightarrow \exists C_i \in C \text{ tal que } |PAD_C(C_i)|>1$$

En ese caso se debe cumplir que la intersección de todos los padres de ese concepto no sea nula:

$$\forall C_i \in C\ \ \forall C_j \in PAD_C(C_i)\ \ \cap ST_C(C_j, t_i) \neq \varnothing,$$

Con la definición dada de clasificación, puede ocurrir que una instancia pertenezca a varios conceptos de forma simultánea, sin que estos conceptos tengan una relación superconcepto – subconcepto directa.

Existen dos posibilidades:

- Cuando se tiene una **clasificación "superpuesta"** una instancia puede pertenecer a varios de los subconceptos. (Por ejemplo la clasificación de los animales en cazadores y presas)
- Cuando se tienen **varias clasificaciones para el mismo superconcepto**, suele ocurrir que una instancia pertenezca a subconceptos de cada una de las clasificaciones. (Por ejemplo, al clasificar personas por edad, y por sexo, una instancia puede ser joven [edad] y mujer [sexo] al mismo tiempo).

La única restricción que existe es que una instancia sólo podrá pertenecer a dos o más conceptos si estos conceptos tienen un antecesor común. Expresado formalmente, para toda instancia, sólo podrán existir dos conceptos a los cuales pertenece si y sólo si la intersección de los antecesores de ambos conceptos no es vacía:

$$\forall I_i \in I \text{ se cumple } \exists C_j \in C\ \exists C_k \in C\ I_i \in ST_C(C_j, t_i) \wedge I_i \in ST_C(C_k, t_i) \Leftrightarrow ANT_C(C_j) \cap ANT_C(C_k) \neq \varnothing$$

4.2.1.6.2 Especializaciones

ESB es el **universo de especializaciones de subconcepto**. Se definen seis tipos de especialización: sobre el grado y límite de asociación o bien sobre el grado, límite, tipo y valor por omisión de atributo.

$$ESB = EGAS \cup ELAS \cup EGAT \cup ELAT \cup ETAT \cup EVAT$$

Para las definiciones formales de los tipos de especialización se van a utilizar las siguientes declaraciones:

- Sea $CL_i \in CL$ una clasificación
- Sea $C_k = S_{CL}(CL_i)$ el superconcepto de la clasificación
- Sea $SB_j \in SB_{CL}(CL_i)$ una de las definiciones de subconcepto de la clasificación
- Sea $C_l = C_{SB}(SB_j)$ el concepto correspondiente a esa definición de subconcepto

Seguidamente se va a proceder a definir cada uno de los seis tipos de especialización.

1. **Especialización de grado de participación en asociación** (EGAS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa el superconcepto, el grado de participación del subconcepto se vuelve más restrictivo (es decir, se convierte en "obligatorio").

   Una especialización de este tipo se especifica mediante un par ordenado en el que se indican la asociación y el rol cuyo grado se modifica.

   $$EGAS = \{ EGAS_i \mid EGAS_i \cong (AS_{EG}(EGAS_i), RL_{EG}(EGAS_i))\}$$

   Seguidamente se detallan los elementos de una especialización de grado de asociación:

   - **AS_{EG} (Asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de asociación (*EGAS*) y el conjunto de asociaciones (*AS*), de forma que a cada especialización se le asigna una única asociación.

     $$AS_{EG}: EGAS \rightarrow AS$$

   - **RL_{EG} (Rol** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de asociación (*EGAS*) y el conjunto de roles (*RL*), de forma que a cada especialización se le asigna un rol.

     $$RL_{EG}: EGAS \rightarrow RL$$

   Para toda especialización debe cumplirse que su asociación (AS_{EG}) sea una asociación en la que participe (dentro del origen o como destino) el superconcepto de la clasificación, precisamente en el rol asignado a la especialización (RL_{EG}).

   La expresión formal de la condición anterior es la siguiente: sea $EGAS_i \in ESP_{SB}(SB_j)$, sea $AS_x = AS_{EG}(EGAS_i)$ y sea $RL_y = RL_{EG}(EGAS_i)$. $EGAS_i$ es válida si y sólo si el rol RL_y existe entre sus participantes (orígenes o destino) y su participantes es el concepto C_k (el superconcepto):

   $$EGAS_i \text{ válida} \Leftrightarrow (\exists RL_z \in OR_{AS}(AS_x) \cup \{DES_{AS}(AS_x)\} , RL_z = RL_y \wedge$$
   $$P_{RL}(RL_z) = C_k)$$

   La existencia de la especialización implica que para cada instante de tiempo, todas las instancias del estado del subconcepto C_l deberán aparecer al menos una vez en la asociación en la posición del rol correspondiente.

   $$EGAS_i = (AS_x, RL_y) \Rightarrow \forall t_i \in N \bullet \forall I_z \in ST_C(C_l, t_i) \bullet \exists k \in N \bullet RL_Y = RL_{AS}(AS_x, k)$$
   $$\Rightarrow \exists IAS_i \in ST_{AS}(AS_x, t_i) \bullet \wedge \; I_z = IAS_i[k])$$

Equipo de fútbol

En la clasificación de Persona por su ocupación (Jugador, Directivo o Técnico) se puede definir una especialización de grado de asociación: todo jugador debe pertenecer obligatoriamente a un equipo para ser considerado como tal. Para ello hay que especializar el grado del rol "persona" de la asociación "Pertenencia":

```
EGAS("(Persona, Fecha).Pertenencia","persona")
```

Dado que el nombre del rol es único dentro de la asociación "Pertenencia", se ha utilizado ese nombre como representación del rol.

La clasificación "por ocupación" de Persona quedaría como sigue:

```
CL("por ocupación", "Persona", {("Técnico", ø), ("Directivo", ø),
("Jugador", {EGAS("(Persona, Fecha).Pertenencia", "persona"})}, f, f)
```

2. **Especialización de límite de participación en asociación** (ELAS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa el superconcepto, el límite de participación del subconcepto se vuelve más restrictivo (es decir, se convierte en "único").

   Una especialización de este tipo se especifica mediante un par ordenado en el que se indican la asociación y el rol cuyo límite se modifica.

   $$ELAS = \{ ELAS_i \mid ELAS_i \cong (AS_{EL}(ELAS_i), RL_{EL}(ELAS_i))\}$$

   Seguidamente se detallan los elementos de una especialización de límite de asociación:

   - **AS_{EL} (Asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de asociación (*ELAS*) y el conjunto de asociaciones (*AS*), de forma que a cada especialización se le asigna una asociación.

     $$AS_{EL}: ELAS \rightarrow AS$$

   - **RL_{EL} (Rol** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de asociación (*ELAS*) y el conjunto de roles (*RL*), de forma que a cada especialización se le asigna un rol.

     $$RL_{EL}: ELAS \rightarrow RL$$

   Para toda especialización debe cumplirse que su asociación (AS_{EL}) sea una asociación en la que participe (como parte del origen o como destino) el superconcepto de la clasificación, precisamente en el rol asignado a la especialización (RL_{EL}).

   La expresión formal de la condición anterior es la siguiente: sea $ELAS_i \in ESP_{SB}(SB_j)$, sea $AS_x = AS_{EG}(ELAS_i)$ y sea $RL_y = RL_{EL}(ELAS_i)$. $ELAS_i$ es válida si y sólo si existe un rol entre sus participantes (orígenes o destino) igual a RL_y y su participante es el concepto C_k (el superconcepto):

   $$ELAS_i \text{ válida} \Leftrightarrow (\exists RL_z \in OR_{AS}(AS_x) \cup \{DES_{AS}(AS_x)\}\ ,\ RL_z = RL_y \wedge P_{RL}(RL_z) = C_k)$$

   La existencia de la especialización implica que para todo instante de tiempo todas las instancias del subconcepto C_l podrán aparecer como máximo una vez en la asociación. Esta condición se formaliza de manera similar a la especialización de límite de atributo.

3. **Especialización de grado de atributo** (EGAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su grado se vuelve más restrictivo en el subconcepto (es decir se convierte en "obligatorio").

   Una especialización de este tipo se especifica únicamente mediante el atributo cuyo grado se modifica.

   $$EGAT = \{ EGAT_i \mid EGAT_i \cong (AT_{EG}(EGAT_i))\}$$

   Una especialización de grado de atributo tiene sólo un elemento:

   - **AT_{EG} (Atributo** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de atributo (*EGAT*) y el conjunto de atributos (AT), de forma que a cada especialización se le asigna un único atributo.

   $$AT_{EG}: EGAT \rightarrow AT$$

   Para toda especialización debe cumplirse que su atributo (AT_{EG}) esté definido en el superconcepto de la clasificación, bien directamente, o bien en alguno de sus antecesores (superconceptos suyos en alguna clasificación).

   La expresión formal de la condición anterior es la siguiente: sea $EGAT_i \in ESP_{SB}(SB_j)$ y sea $AT_x = AT_{EG}(EGAT_i)$. $EGAT_i$ es válida si y sólo si el superconcepto (C_k) es el concepto del atributo especializado, o bien, si el concepto del atributo especializado es uno de los antecesores del superconcepto:

   $$EGAT_i \text{ válida} \Leftrightarrow (C_{AT}(AT_x) = C_k \wedge C_{AT}(AT_x) \in ANT_C(C_k))$$

   La existencia de la especialización implica que todas las instancias del subconcepto C_l deberán tener al menos un valor definido para el atributo:

   $$EGAT_i \Rightarrow \forall t_i \in N \bullet \forall I_y \in ST_C(C_l, t_i) \bullet \exists V_z \in T_{AT}(AT_x) \bullet (I_y, V_z) \in ST_{AT}(AT_x, t_i)$$

4. **Especialización de límite de atributo** (ELAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su límite se vuelve más restrictivo en el subconcepto (es decir se convierte en "único").

   Una especialización de este tipo se especifica únicamente mediante el atributo cuyo límite se modifica.

   $$ELAT = \{ ELAT_i \mid ELAT_i = (AT_{EL}(ELAT_i))\}$$

   Una especialización de grado de atributo tiene sólo un elemento:

   - **AT_{EL} (Atributo** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de atributo (*ELAT*) y el conjunto de atributos (*AT*), de forma que a cada especialización se le asigna un único atributo.

   $$AT_{EL}: ELAT \rightarrow AT$$

   Para toda especialización debe cumplirse que su atributo (AT_{EL}) esté definido en el superconcepto de la clasificación, bien directamente, o bien en alguno de sus antecesores (superconceptos suyos en alguna clasificación).

   La expresión formal de la condición anterior es la siguiente: sea $ELAT_i \in ESP_{SB}(SB_j)$ y sea $AT_x = AT_{EL}(ELAT_i)$. $ELAT_i$ es válida si y sólo si el superconcepto (C_k) es el concepto del atributo especializado, o bien, si el concepto del atributo especializado es uno de los antecesores del superconcepto:

   $$ELAT_i \text{ válida} \Leftrightarrow (C_{AT}(AT_x) = C_k \vee C_{AT}(AT_x) \in ANT_C(C_k))$$

La existencia de la especialización implica que todas las instancias del subconcepto C_l tendrán como máximo un valor definido para el atributo:

$$ELAT_i \Rightarrow \forall t_i \in N \bullet \forall I_y \in ST_C(C_l, t_i) \bullet \forall V_i, V_j \in T_{AT}(AT_x) \bullet V_i \neq V_j \wedge$$

$$(I_y, V_i) \in ST_{AT}(AT_x, t_i) \Rightarrow (I_y, V_j) \notin ST_{AT}(AT_x, t_i)$$

5. **Especialización de tipo de atributo** (ETAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, su tipo se vuelve más restrictivo en el subconcepto. Para ello el tipo definido en la especialización del atributo debe ser un tipo derivado (directa o indirectamente) del tipo definido en el atributo del superconcepto.

   Una especialización de este tipo se especifica mediante el atributo cuyo tipo se restringe y el nuevo tipo.

   $$ETAT = \{ ETAT_i \mid ETAT_i = (AT_{ET}(ETAT_i), T_{ET}(ETAT_i))\}$$

   Una especialización de grado de atributo tiene dos elementos:

   - **AT_{ET} (Atributo** de la especialización). Es una aplicación entre el conjunto de especializaciones de tipo de atributo (*ETAT*) y el conjunto de atributos (*AT*), de forma que a cada especialización se le asigna un único atributo.

     $$AT_{ET}: ETAT \rightarrow AT$$

   - **T_{ET} (Tipo** de la especialización). Es una aplicación entre el conjunto de especializaciones de tipo de atributo (*ETAT*) y el conjunto de tipos (*T*), de forma que a cada especialización se le asigna un único tipo.

     $$T_{ET}: ETAT \rightarrow T$$

   Para toda especialización debe cumplirse que su atributo (AT_{ET}) esté definido en el superconcepto de la clasificación, bien directamente, o bien en alguno de sus antecesores (superconceptos suyos en alguna clasificación). Además, el tipo de la especialización debe ser un tipo derivado del tipo definido en el superconcepto.

   La expresión formal de la condición anterior es la siguiente: sea $ETAT_i \in ESP_{SB}(SB_j)$, sea $AT_x = AT_{ET}(ETAT_i)$ y sea $T_y = T_{ET}(ETAT_i)$. $ETAT_i$ es válida si y sólo si el superconcepto (C_k) es el concepto del atributo especializado, o bien, si el concepto del atributo especializado es uno de los antecesores del superconcepto y, además, el tipo de la especialización es derivado del tipo del atributo:

   $$ETAT_i \text{ válida} \Leftrightarrow (C_{AT}(AT_x) = C_k \vee C_{AT}(AT_x) \in ANT_C(C_k)) \wedge T_{AT}(AT_x) \in ANT_T(T_y)$$

   La existencia de la especialización implica que los valores asociados a las instancias del subconcepto C_l deben pertenecer al nuevo tipo:

   $$ETAT_i \Rightarrow \forall t_i \in N \bullet \forall I_y \in ST_C(C_l, t_i) \bullet \forall V_z \in T_{AT}(AT_x) \bullet (I_y, V_z) \in ST_{AT}(AT_x, t_i) \text{ ç}$$

   $$\Rightarrow V_z \in T_y$$

Equipo de fútbol

Hay especializaciones de tipo de atributo en las dos clasificaciones del concepto Jugador.

En la clasificación por posición se especializa el atributo "posición" para los dos subconceptos (Portero y Jugador Campo):

```
Portero → ETAT("Posición", "Posición portero")

Jugador Campo → ETAT("Posición", "Posición campo")
```

La clasificación "por posición" queda como sigue:

```
CL("posición", "Jugador", {("Portero", {ETAT("Posición", "Posición
portero")}), ("Jugador campo", {ETAT("Posición", "Posición campo"})},
"exclusiva", "exhaustiva")
```

En la clasificación por país se especializa el atributo "país" para los tres subconceptos (Nacional, Comunitario y Extranjero):

```
Nacional → ETAT("País", "País nacional")

Comunitario → ETAT("País", "País comunitario")

Extranjero → ETAT("País", "País extranjero")
```

La clasificación "por país" queda como sigue:

```
CL("por país", "Jugador", {("Nacional", {ETAT("País", "País nacional")}),
("Comunitario", {ETAT("País", "País comunitario")}), ("Extranjero",
{ETAT("País", "País extranjero")})}, "exclusiva", "exhaustiva")
```

6. **Especialización de valor por omisión de atributo** (EVAT). Cada especialización de este tipo indica que, dado un atributo del superconcepto, el valor por omisión de ese atributo cambia para las instancias del subconcepto considerado.

   Una especialización de este tipo se especifica mediante el atributo especializado y su nuevo valor por omisión.

   $$EVAT = \{ EVAT_i \mid EVAT_i = (AT_{EV}(EVAT_i), V_{EV}(EVAT_i))\}$$

   Una especialización de grado de atributo tiene dos elementos:

   - **AT_{EV} (Atributo** de la especialización). Es una aplicación entre el conjunto de especializaciones de valor por omisión de atributo (*EVAT*) y el conjunto de atributos (*AT*), de forma que a cada especialización se le asigna un atributo.

     $$AT_{EV}: EVAT \rightarrow AT$$

   - **V_{EV} (Valor por omisión** de la especialización). Es una aplicación entre el conjunto de especializaciones de valor por omisión de atributo (*EVAT*) y el conjunto de valores (*V*), de forma que a cada especialización se le asigna un único valor.

     $$V_{EV}: EVAT \rightarrow V$$

   Para toda especialización debe cumplirse que su atributo (AT_{EV}) esté definido en el superconcepto de la clasificación, bien directamente, o bien en alguno de sus antecesores (superconceptos suyos en alguna clasificación). Además, el valor por omisión de la especialización debe pertenecer al tipo del atributo.

   La expresión formal de la condición anterior es la siguiente: sea $EVAT_i \in ESP_{SB}(SB_j)$, sea $AT_x = AT_{EV}(EVAT_i)$ y sea $V_y = V_{EV}(EVAT_i)$. $ETAT_i$ es válida si y sólo si el superconcepto (C_k) es el concepto del atributo especializado, o bien, si el concepto del atributo especializado es uno de los antecesores del superconcepto y, además, el valor por omisión de la especialización pertenece al tipo del atributo:

   $$EVAT_i \text{ válida} \Leftrightarrow (C_{AT}(AT_x) = C_k \vee C_{AT}(AT_x) \in ANT_C(C_k)) \wedge V_y \in T_{AT}(AT_x)$$

La existencia de la especialización implica que el valor por omisión para las instancias del subconcepto C_l es el valor correspondiente a la especialización.

Equipo de fútbol

Se podría definir una especialización de valor por omisión en los "Puestos", con un atributo booleano "titular". En el subconcepto "titular" ese atributo tiene valor "cierto", mientras que en el subconcepto "suplente" ese atributo tiene valor "falso".

```
Titular → EVAT("titular", "cierto")

Suplente → EVAT("titular", "falso")
```

La clasificación de puestos quedaría como sigue:

```
CL("por titularidad", "Puesto", {("Titular", { EVAT("titular",
"cierto")}), ("Suplente", { EVAT("titular", "falso")})}, "exclusiva",
"superpuesta")
```

Respecto a las especializaciones definidas en un subconcepto, debe cumplirse que **el conjunto de especializaciones asignadas a una definición de subconcepto sea consistente**. Para ello:

- Para cada asociación en la participa el superconcepto no deben existir más de una especialización de grado y de límite para cada rol en el que participe el superconcepto.
- Para cada atributo del superconcepto sólo debe existir una especialización de cada tipo: grado, límite, tipo del atributo y valor por omisión.
- Sólo se podrá especializar aquello que no esté restringido. Por ejemplo, no se podrá especializar el grado de un rol que ya es "obligatorio".

4.2.1.6.3 Restricciones afectadas por clasificaciones de concepto

Alguna de las restricciones que han ido apareciendo anteriormente se ven afectadas por la definición de clasificaciones:

- **El nombre de atributo será único en toda jerarquía de herencia del concepto en el que se define**. Si se define un atributo en un concepto determinado, no deberá existir otro atributo con el mismo nombre ni en ese concepto ni en cualquiera de los familiares del concepto.

$$\forall AT_i, ATj \in AT, AT_i \neq AT_j \wedge (C_{AT}(AT_j) = C_{AT}(AT_i) \vee C_{AT}(AT_j) \in FAM_C(C_{AT}(AT_i))) \Rightarrow N_{AT}(AT_i) \neq N_{AT}(AT_j)$$

  Para expresar formalmente esta condición se ha utilizado la aplicación **familiares** de concepto (FAM_C), que asigna a cada concepto el conjunto de todos los conceptos que tienen algún tipo de relación de clasificación con el dado y que se define formalmente más abajo.

  Esta función se define calculando la unión de (a) los padres del concepto, (b) los hijos del concepto, (c) los familiares de los padres y (d) los familiares de los hijos.

- **El nombre de asociación será único en toda la jerarquía del concepto o conceptos que definen su origen**: no deberá existir otra asociación con el mismo nombre que tenga por origen alguno de los familiares del concepto.

4.2.1.6.4 Antecesores, Sucesores y Familiares de concepto

En este apartado se van a definir formalmente algunas funciones útiles relacionadas con las clasificaciones que se han utilizado a lo largo de la descripción de las clasificaciones de concepto:

- *Padres* de un concepto: representa el conjunto de todos los superconceptos directos de un concepto dado.
- *Antecesores* de un concepto: representa el conjunto de todos los superconceptos, directos o no, de un concepto dado.
- *Hijos* de un concepto: representa el conjunto de todos los subconceptos directos de un concepto dado.
- *Sucesores* de un concepto: representa el conjunto de todos los subconceptos, directos o no, de un concepto dado.
- *Familiares* de un concepto: representa el conjunto de todos los conceptos que tienen alguna relación mediante clasificaciones con el concepto dado. Este conjunto incluye los sucesores del concepto, los antecesores del mismo y los familiares de ambos.

Todas estas funciones son aplicaciones entre el conjunto de los conceptos (C) y las partes del conjunto de conceptos ($\Pi(C)$).

La aplicación **padres de concepto** (PAD_C) es una aplicación que asocia un concepto al conjunto (que puede ser vacío) de todos sus superconceptos *directos*, para toda clasificación en la que el concepto dado es un subconcepto:

$$PAD_C: C \rightarrow \Pi(C)$$

$$PAD_C(C_i) = \{C_j \in C \mid \exists CL_k \in CL \text{ con } C_j = S_{CL}(CL_k) \wedge \exists SB_l \in SB_{CL}(CL_k) \text{ con } C_i = C_{SB}(SB_l)\}$$

La aplicación **antecesores de concepto** (ANT_C) devuelve el conjunto de superconceptos, directos o no, de un concepto dado: padres del concepto, padres de los padres del concepto, padres de los padres de los padres, y así hasta que no haya más clasificaciones.

$$ANT_C: C \rightarrow \Pi(C)$$

Se puede definir de forma recursiva $ANT_C(C_i)$ como el conjunto de elementos que son padres directos de C_i o bien antecesores de los padres directos de C_i:

$$ANT_C(C_i) = \{C_j \in C \mid C_j \in PAD_C(C_i) \vee \exists C_k \in PAD_C(C_i) \text{ con } C_j \in ANT_C(C_k)\}$$

La aplicación **hijos de concepto** (HIJ_C) es una aplicación que asocia un concepto al conjunto (que puede ser vacío) de todos sus subconceptos *directos*, para toda clasificación en la que el concepto dado es superconcepto:

$$HIJ_C: C \rightarrow \Pi(C)$$

$$HIJ_C(C_i) = \{C_j \in C \mid \exists CL_k \in CL \text{ con } C_i = S_{CL}(CL_k) \wedge \exists SB_l \in SB_{CL}(CL_k) \text{ con } C_j = C_{SB}(SB_l)\}$$

La aplicación **sucesores de concepto** (SUC_C) devuelve el conjunto de subconceptos, directos o no, de un concepto dado: hijos del concepto, hijos de los hijos del concepto, hijos de los hijos de los hijos, y así hasta que no haya más clasificaciones.

$$SUC_C: C \rightarrow \Pi(C)$$

Se puede definir de forma recursiva $SUC_C(C_i)$ como el conjunto de elementos que son hijos directos de C_i o bien sucesores de los hijos directos de C_i:

$$SUC_C(C_i) = \{C_j \in C \mid C_j \in HIJ_C(C_i) \vee \exists C_k \in HIJ_C(C_i) \text{ con } C_j \in SUC_C(C_k)\}$$

La aplicación **familiares de concepto** (FAM_C) devuelve el conjunto de todos los conceptos que tienen algún tipo de relación de clasificación con el concepto dado: padres, familiares de padres, hijos y familiares de los hijos.

$$FAM_C: C \rightarrow \Pi(C)$$

Se puede definir de forma recursiva $FAM_C(C_i)$ como el conjunto de elementos (distintos de C_i) que son padres o hijos directos de C_i o bien son familiares de los padres o hijos directos de C_i:

$$FAM_C(C_i) = \{C_j \in C \mid C_j \neq C_i \wedge (C_j \in PAD_C(C_i) \vee C_j \in HIJ_C(C_i) \vee \exists C_k \in PAD_C(C_i) \text{ con } C_j \in FAM_C(C_k) \vee \exists C_l \in HIJ_C(C_i) \text{ con } C_j \in FAM_C(C_l))\}$$

Para aclarar estas definiciones se va a mostrar un ejemplo gráfico. La Figura 4.33 muestra un grafo complejo de clasificaciones en el que existe herencia múltiple (en dos casos: "K" es subconcepto de "D", "E", mientras que "L" es subconcepto de "E", "F".

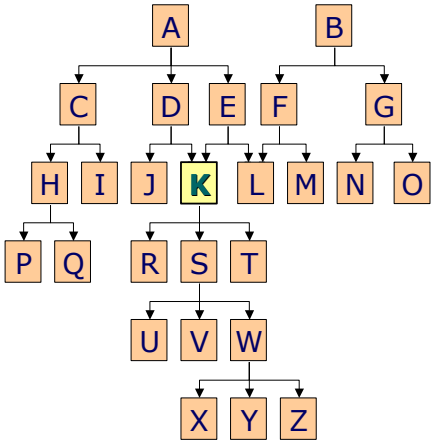


Figura 4.33 Ejemplo de grafo complejo de clasificaciones

Si se calculan las funciones anteriores para el concepto "K", se obtienen los siguientes resultados:

- PAD_C("K") = {"D", "E"}
- ANT_C("K") = {"D", "E", "A"}
- HIJ_C("K") = {"R", "S", "T"}
- SUC_C("K") = {"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
- FAM_C("K") = {"D", "E", "R", "S", "T"} ∪ FAM_C("D") ∪ FAM_C("E") ∪ FAM_C("R") ∪ FAM_C("S") ∪ FAM_C("T") = {<<todos los conceptos >>}

4.2.1.7 Formalización de clasificaciones de asociación

La formalización de las clasificaciones de asociación se ha divido por claridad en una serie de apartados, empezando por la formalización de los elementos principales. Después se definirán formalmente las especializaciones, que permiten restringir las propiedades de las subasociaciones. Posteriormente se tratará la forma en la que la herencia afecta a restricciones que han aparecido hasta ahora y, por último, se definirán formalmente los antecesores, sucesores y familiares de una asociación.

4.2.1.7.1 Clasificaciones

El **universo de clasificaciones de asociación** (CLS) es el conjunto de todas las clasificaciones de asociación, cada una de las cuales se especifica mediante una quíntupla formada por el nombre de la clasificación, la superasociación, el conjunto de subasociaciones, la indicación de si es disjunta o no y la indicación de si es completa o no:

$$CLS = \{CLS_i \mid CLS_i = (N_{CLS}(CLS_i), SA_{CLS}(CLS_i), SBA_{CLS}(CLS_i), DIS_{CLS}(CLS_i), COM_{CLS}(CLS_i))\}$$

- **N_{CLS}** (**Nombre** de clasificación de asociación) es una aplicación entre el conjunto de clasificaciones de asociación (*CL*) y la unión entre el conjunto de las cadenas (*CAD*) y el conjunto cuyo único elemento es el conjunto vacío ($\{\varnothing\}$), de forma que a cada clasificación de asociación se le podrá asociar o no una cadena:

$$N_{CLS}: CLS \rightarrow CAD \cup \{\varnothing\}$$

- **SA_{CLS}** (**Superasociación** de clasificación) es una aplicación entre el conjunto de clasificaciones de asociación (*CLS*) y el conjunto de las asociaciones (*AS*), de forma que a cada clasificación se le asocia una asociación, que es su superasociación:

$$SA_{CLS}: CLS \rightarrow AS$$

  Para toda clasificación de asociación de un modelo estructural su superasociación también debe pertenecer al modelo estructural.

$$\forall EST_i \in EST \bullet \forall CLS_i \in CLS_{EST}(EST_i) \bullet SA_{CL}(CLS_i) \in AS_{EST}(EST_i)$$

  Para toda asociación, si existen dos o más clasificaciones de las que es superasociación, esas clasificaciones deben tener nombre asignado y ese nombre debe ser distinto en todas ellas:

$$\forall AS_i \in AS \bullet \exists CLS_j \in CLS \bullet \exists CLS_k \in CLS \bullet CLS_j \neq CLS_k \wedge AS_i = SA_{CLS}(CLS_j) \wedge AS_i = SA_{CLS}(CLS_k) \Rightarrow N_{CLS}(CLS_j) \neq \varnothing \wedge N_{CLS}(CLS_k) \neq \varnothing \wedge N_{CLS}(CLS_j) \neq N_{CLS}(CLS_k)$$

- **SBA_{CLS}** (**Subasociaciones** de clasificación) es una aplicación inyectiva entre el conjunto de clasificaciones de asociación (*CLS*) y las partes del conjunto de las definiciones de subasociaciones (*P(SBA)*), de forma que a cada clasificación se le asocia un conjunto de definiciones de subasociaciones:

$$SBA_{CLS}: CLS \rightarrow P(SBA)$$

  SBA es el **universo de definiciones de subasociaciones**. Cada **definición de subasociación** está asociada a una asociación (la subasociación) y define cómo

cambia esa asociación respecto a la superasociación de la clasificación. Cada uno de los cambios se denomina especialización.

$$SBA = \{SBA_i \mid SBA_i \cong (AS_{SBA}(SBA_i), ESP_{SBA}(SBA_i))\}$$

Los elementos de una definición de subasociación son:

- **AS_{SBS}** (**Asociación de definición** de subasociación) es una aplicación entre el conjunto de definiciones de subasociaciones (*SBA*) y el universo de asociaciones (*AS*), de forma que a cada definición de subasociación se le asocia una asociación (la subasociación de esa definición).

$$AS_{SBA}: SBA \rightarrow AS$$

- **ESP_{SBA}** (**Especializaciones de definición** de subasociación) es una aplicación inyectiva entre el conjunto de definiciones de subasociaciones (*SBA*) y las partes del conjunto de especializaciones de subasociación (*P(ESBA)*), de forma que a cada definición de subasociación se le asocia un conjunto de especializaciones.

$$ESP_{SBA}: SBA \rightarrow P(ESBA)$$

  La explicación detallada de las especializaciones de asociación es considerablemente extensa, por lo que se deja para más adelante (apartado "Especializaciones"), con el fin de no interrumpir la definición completa de clasificación de asociación.

Para toda clasificación de asociación de un modelo estructural las asociaciones asociadas a sus definiciones de subasociación también deben pertenecer al modelo estructural.

$$\forall EST_i \in EST \bullet \forall CLS_i \in CLS_{EST}(EST_i) \bullet \forall SBA_i \in SBA_{CLS}(CLS_i) \bullet AS_{SBA}(SBA_i) \in AS_{EST}(EST_i)$$

En relación con las subasociaciones se deben cumplir las siguientes condiciones:

1. *Regla de inclusión:* cada subasociación es un subconjunto de la superasociación.

$$\forall t_i \in N \bullet \forall CLS_i \in CLS,\ \forall SBA_j \in SBA_{CLS}(CLS_i), AS_k = SA_{CLS}(CLS_i) \wedge AS_l = AS_{SBA}(SBA_j) \Rightarrow ST_{AS}(AS_l, t_i) \subset ST_{AS}(AS_k, t_i)$$

2. *Regla de no circularidad*: ninguna subasociación debe ser igual a la subasociación o pertenecer al conjunto de sus antecesores (esta regla se podría derivar automáticamente de la regla de inclusión).

$$\forall CLS_i \in CLS \bullet AS_j = AS_{CLS}(CLS_i) \Rightarrow \forall SBA_k \in SBA_{CLS}(CLS_i) \bullet AS_{SBA}(SBA_k) \neq AS_j \wedge AS_{SBA}(SBA_k) \notin ANT_{AS}(AS_j)$$

   En esta regla se ha utilizado la aplicación **antecesores de asociación** (ANT_{AS}) que representa todos los "padres", directos o no, de una asociación. Esta función se definirá formalmente más adelante, en el apartado "Antecesores, Sucesores y Familiares de asociación".

3. *Regla de compatibilidad de roles*: los roles de las subasociaciones deben ser compatibles con los roles de la superasociación (esta regla también se puede derivar de la regla de inclusión). Para ello se deben cumplir tres condiciones:

a. La dimensión de las subasociaciones debe ser igual a la dimensión de la superasociación.

$$\forall CLS_i \in CLS \bullet \forall SBA_j \in SBA_{CLS}(CLS_i) \bullet DIM_{AS}(SA_{CLS}(CLS_i)) = DIM_{AS}(AS_{SBA}(SBA_j))$$

b. Los roles de las subasociaciones deben tener el mismo participante o un sucesor del participante del rol correspondiente en la superasociación. Esto se debe cumplir tanto en los roles del origen como en el destino.

   Se procede a descomponer esta condición, para que su expresión formal sea más sencilla.

   Se tienen las siguientes definiciones previas:

$$\forall CLS_i \in CLS \bullet \forall SBA_j \in SBA_{CLS}(CLS_i) \bullet sea\ AS_s = SA_{CLS}(CLS_i) \bullet sea\ AS_b = AS_{SBA}(SBA_j) \bullet n = DIM_{AS}(AS_s)$$

   En primer lugar, se debe cumplir que el participante del destino de cada subasociación sea igual o sucesor del participante del destino de la superasociación:

   Sea $P_s = P_{RL}(DES_{AS}(AS_s)) \bullet$ *sea* $P_b = P_{RL}(DES_{AS}(AS_b))$, *entonces:*

   $P_s \in T \Rightarrow P_b \in T \wedge (P_b = P_s \vee P_b \in SUC_T(P_s))$

   $P_s \in C \Rightarrow P_b \in C \wedge (P_b = P_s \vee P_b \in SUC_C(P_s))$

   $P_s \in AS \Rightarrow P_b \in AS \wedge (P_b = P_s \vee P_b \in SUC_{AS}(P_s))$

   En segundo lugar, se debe cumplir la misma condición para cada uno de los orígenes:

   $\forall j\ 1 \leq j \leq n\text{-}1 \bullet$ *Sea* $P_s = P_{RL}(OR_{AS}(AS_s)[j])$, *sea* $P_b = P_{RL}(OR_{AS}(AS_b)[j])$, *entonces:*

   $P_s \in T \Rightarrow P_b \in T \wedge (P_b = P_s \vee P_b \in SUC_T(P_s))$

   $P_s \in C \Rightarrow P_b \in C \wedge (P_b = P_s \vee P_b \in SUC_C(P_s))$

   $P_s \in AS \Rightarrow P_b \in AS \wedge (P_b = P_s \vee P_b \in SUC_{AS}(P_s))$

   En estas condiciones se ha utilizado la aplicación **sucesores de asociación** (SUC_{AS}) que representa todos los "hijos", directos o no, de una asociación. Esta función se definirá formalmente más adelante, en el apartado "Antecesores, Sucesores y Familiares de asociación".

c. El grado y límite de los roles de las subasociaciones deben ser iguales o más restrictivos que los roles de la superasociación. Respecto al grado de participación, esto quiere decir que si ya es "obligatorio" en algún rol de la superasociación, entonces debe ser también "obligatorio" en los roles correspondientes de las subasociaciones. Respecto al límite, si es "único" en algún rol de la superasociación, deberá serlo en los roles correspondientes de las subasociaciones.

   Para expresar formalmente esta condición se procede de nuevo por partes. Definiciones previas:

$$\forall CLS_i \in CLS \bullet \forall SBA_j \in SBA_{CLS}(CLS_i) \bullet sea\ AS_s = SA_{CLS}(CLS_i) \bullet sea\ AS_b = AS_{SBA}(SBA_j) \bullet sea\ n = DIM_{AS}(AS_s)$$

En primer lugar, las restricciones sobre grado y límite se deben cumplir en el destino de la asociación:

Sea $RL_s = DES_{AS}(AS_s) \bullet$ *sea* $RL_b = DES_{AS}(AS_b)$*, entonces:*
$G_{RL}(RL_s) =$ *"obligatorio"* $\Rightarrow G_{RL}(RL_b) =$ *"obligatorio"*
$L_{RL}(RL_s) =$ *"único"* $\Rightarrow L_{RL}(RL_b) =$ *"único"*

En segundo lugar, las restricciones sobre grado y límite deben cumplirse en cada uno de los orígenes:

$\forall j\ 1 \leq j \leq n\text{-}1 \bullet$ *Sea* $RL_s = OR_{AS}(AS_s)[j]$*, sea* $RL_b = OR_{AS}(AS_b)[j]$*, entonces:*
$G_{RL}(RL_s) =$ *"obligatorio"* $\Rightarrow G_{RL}(RL_b) =$ *"obligatorio"*
$L_{RL}(RL_s) =$ *"único"* $\Rightarrow L_{RL}(RL_b) =$ *"único"*

Las tres reglas anteriores (inclusión, no circularidad y compatibilidad de roles) sirven para asegurar que una clasificación de asociación es válida.

- **DIS$_{CLS}$** (Clasificación de asociación **disjunta**) es una aplicación entre el conjunto de clasificaciones de asociación (*CLS*) y el conjunto de los valores booleanos (*B*), de forma que a cada clasificación de asociación se le asocia un valor booleano que indica si es disjunta o no:

$$DIS_{CLS}: CLS \rightarrow B$$

Si la clasificación no es disjunta (valor *f*), indica que las subasociaciones pueden solaparse, es decir, que pueden tener instancias en común. En otro caso, los conjuntos de instancias de las subasociaciones serán disjuntos.

$\forall t_i \in N \bullet \forall CLS_i \in CLS$ *se cumple:*

$$DIS_{CLS}(CLS_i) = v \Leftrightarrow \cap ST_{AS}(AS_{SBA}(SBA_j), t_i) = \varnothing,\ \forall SBA_j \in SBA_{CLS}(CLS_i)$$

$$DIS_{CLS}(CLS_i) = f \Leftrightarrow \cap ST_{AS}(AS_{SBA}(SBA_j), t_i) \neq \varnothing,\ \forall SBA_j \in SBA_{CLS}(CLS_i)$$

- **COM$_{CLS}$** (Clasificación de asociación **completa**) es una aplicación entre el conjunto de clasificaciones de asociación (*CLS*) y el conjunto de los valores booleanos (*B*) de forma que a cada clasificación se le asocia un valor booleano diciendo si es completa o no:

$$COM_{CLS}: CLS \rightarrow B$$

Si la clasificación es completa (valor *v*) indica que la unión de los conjuntos de instancias de las subasociaciones es exactamente igual al conjunto de instancias de la superasociación. En otro caso, la unión de los conjuntos de instancias de las subasociaciones será un subconjunto del conjunto de instancias de la superasociación.

$\forall t_i \in N \bullet \forall CLS_i \in CLS$ *se cumple:*

$$COM_{CLS}(CLS_i) = v \Leftrightarrow \cup ST_{AS}(AS_{SBA}(SBA_j), t_i) = ST_{AS}(SA_{CLS}(CLS_i), t_i),\ \forall SBA_j \in SBA_{CLS}(CLS_i)$$

$$COM_{CLS}(CLS_i) = f \Leftrightarrow \cup ST_{AS}(AS_{SBA}(SB_j), t_i) \subset ST_{AS}(S_{CLS}(CLS_i), t_i), \forall SBA_j \in SBA_{CLS}(CLS_i)$$

Equipo de fútbol

Seguidamente se muestran las clasificaciones de asociación y definiciones adicionales relacionadas con cada una de ellas:

- Hay **tres tipos de pertenencia de personas a equipos**, en función de su **papel**: Fichaje, Entrenador y Presidente. Esta clasificación es no disjunta e incompleta. La asociación "Fichaje" tiene dos atributos: el importe de su ficha y el importe de su cláusula de rescisión. La asociación "Entrenador" tiene dos atributos: el sueldo y el objetivo planteado al contratar al entrenador. La asociación "Presidente" tiene un atributo para indicar si el presidente es, además, propietario del equipo (este atributo tiene valor por omisión falso).

```
AS("Fichaje", 3, (("jugador","Jugador","obligatorio","múltiple"), ("fecha comienzo", "Fecha", "opcional", "múltiple")), ("equipo", "Equipo", "obligatorio", "múltiple"), "Un jugador puede aparecer varias veces pero con fechas distintas")

ATS("ficha", "Fichaje", NUM, "obligatorio", "único", ∅)

ATS("rescisión", "Fichaje", NUM, "obligatorio", "único", ∅)

AS("Entrenador", 3, (("entrenador","Técnico","opcional","múltiple"), ("fecha comienzo", "Fecha", "opcional", "múltiple")), ("equipo", "Equipo", "obligatorio", "múltiple"), "Un equipo sólo puede tener un entrenador en una fecha dada")

ATS("sueldo", "Entrenador", NUM, "obligatorio", "único", ∅)

ATS("objetivo", "Entrenador", CAD, "obligatorio", "único", ∅)

AS("Presidente", 3, (("presidente","Directivo","opcional","múltiple"), ("fecha comienzo", "Fecha", "opcional", "múltiple")), ("equipo", "Equipo", "obligatorio", "múltiple"), "Un equipo sólo puede tener un presidente en una fecha dada")

ATS("propietario", "Presidente", B, "obligatorio", "único", f)

CLS("por papel", "Pertenencia", {("Fichaje", {...}), ("Entrenador", {...}), ("Presidente", {...})}, f, f)
```

- Hay **dos tipos de asociaciones entre tácticas y puestos**, en función de su **titularidad** de origen: titulares y suplentes. Esta clasificación es disjunta y completa.

```
AS("Titulares", 2, (("táctica", "Táctica", "obligatorio", "múltiple")), ("puesto elegido", "Titular", "opcional", "múltiple"), "Para cada táctica habrá 11 puestos de titulares (con 1 portero)")

AS("Suplentes", 2, (("táctica", "Táctica", "obligatorio", "múltiple")), ("puesto elegido", "Suplente", "opcional", "múltiple"), "Para cada táctica habrá 5 suplentes (con 1 portero)")

CL("por titularidad", "Puestos", {("Titulares", {... }), ("Suplentes", {...})}, v, v)
```

En cuanto a las instancias en las asociaciones "Pertenencia", "Fichaje", "Entrenador" y "Presidente", y teniendo en cuenta la clasificación anterior, se podrán tener los siguientes casos:

- Si una persona es jugador y entrenador a la vez, la tupla correspondiente pertenecerá simultáneamente a las asociaciones "Fichaje" y "Entrenador":

```
("Vialli", "1.5.1990", "Arsenal")∈"Fichaje" y ("Vialli", "1.5.1990", "Arsenal")∈"Entrenador"
```

- Una persona puede ser directivo de un equipo pero no ser el presidente:

```
("Valdano", "1.1.2000", "Real Madrid")∈"Pertenencia" y ("Valdano",
"1.1.2000", "Real Madrid")∉"Presidente"
```

Después de formalizar las clasificaciones de concepto se puede volver sobre la herencia simple, la herencia múltiple y la pertenencia de una instancia a varios conceptos.

Para las afirmaciones siguientes se va a suponer que $CLS_i \in CLS$ es una clasificación de asociación, $AS_s = SA_{CLS}(CLS_i) \in AS$ es la superasociación de la clasificación y $AS_b \in AS$ es una de las subasociaciones de la clasificación (es decir, $\exists SBA_j \in SBA_{CLS}(CLS_i)$ *tal que* $AS_b = AS_{SBA}(SBA_j)$).

Sea $IAS_k \in AS_b$ una instancia de la subasociación entonces, dado que $IAS_k \in AS_s$:

- IAS_k puede aparecer en las instancias de asociación (n-tuplas) de cualquier asociación en la que participe AS_s. Por lo tanto **AS_b hereda la participación en asociaciones** definida para AS_s.
- IAS_k puede aparecer en las instancias de cualquier atributo de asociación definido para AS_s. Por lo tanto **AS_b hereda las definiciones de atributos de AS_s**.

Esta herencia de asociaciones y atributos de asociación se realiza de forma que las subasociaciones pueden especializar la participación de asociaciones y la definición de atributos de asociación. Esto es así ya que una subasociación es más especializada que la superasociación y se pueden restringir más esas definiciones, tal y como se verá al definir formalmente los tipos de especialización en el apartado correspondiente.

Formalmente se dice que existe **herencia múltiple** si existe al menos una asociación tal que su conjunto de padres tiene más de un elemento:

$$herencia\ múltiple \Leftrightarrow \exists AS_i \in AS \text{ tal que } |PAD_{AS}(AS_i)| > 1$$

En ese caso se debe cumplir que la intersección de todos los padres de ese concepto no sea nula:

$$\forall AS_i \in AS \ \ \forall AS_j \in PAD_{AS}(AS_i) \ \cap AS_j \neq \varnothing,$$

Con la definición dada de clasificación de asociación, puede ocurrir que una instancia de asociación (una tupla) pertenezca a varias asociaciones de forma simultánea, sin que estas asociaciones tengan una relación superasociación – subasociación directa.

Existen dos posibilidades:

- Cuando se tiene una **clasificación "superpuesta"** (no disjunta) una instancia de asociación puede pertenecer a varias de las subasociaciones.
- Cuando se tienen **varias clasificaciones para la misma superasociación**, suele ocurrir que una instancia pertenezca a subasociaciones de cada una de las clasificaciones.

La única restricción que existe es que una instancia de asociación sólo podrá pertenecer a dos o más asociaciones si estas asociaciones tienen un antecesor común. Expresado formalmente, para toda instancia de asociación, sólo podrán existir dos asociaciones a las cuales pertenece si y sólo si la intersección de los antecesores de ambas asociaciones no es vacía:

$$\forall IAS_i \in IAS \text{ se cumple } \exists AS_j \in AS\ \exists AS_k \in AS\ IAS_i \in AS_j \wedge AS_i \in AS_k \Leftrightarrow ANT_{AS}(AS_j) \cap ANT_{AS}(AS_k) \neq \varnothing$$

4.2.1.7.2 Especializaciones

ESBA es el **universo de especializaciones de subasociación**. Se definen seis tipos de especialización: sobre el grado y límite de asociación o bien sobre el grado, límite, tipo y valor por omisión de atributo de asociación.

$$ESBA = EGASS \cup ELASS \cup EGATS \cup ELATS \cup ETATS \cup EVATS$$

Para las definiciones formales de los tipos de especialización se van a utilizar las siguientes declaraciones:

- Sea $CLS_i \in CLS$ una clasificación de asociación
- Sea $AS_k = SA_{CLS}(CLS_i)$ la superasociación de la clasificación
- Sea $SBA_j \in SBA_{CLS}(CLS_i)$ una de las definiciones de subasociación de la clasificación
- Sea $AS_l = AS_{SBA}(SBA_j)$ la asociación correspondiente a esa definición de subasociación

Seguidamente se va a proceder a definir cada uno de los seis tipos de especialización.

1. **Especialización de grado de participación en asociación** (EGASS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa la superasociación, el grado de participación de la subasociación se vuelve más restrictivo (es decir, se convierte en "obligatorio").

   Una especialización de este tipo se especifica mediante un par ordenado en el que se indican la asociación y el rol cuyo grado se modifica.

   $$EGASS = \{ EGASS_i \mid EGASS_i \cong (AS_{EGS}(EGASS_i), RL_{EGS}(EGASS_i)) \}$$

   Seguidamente se detallan los elementos de una especialización de grado de asociación:

   - **AS_{EGS} (Asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de asociación de subasociación (*EGASS*) y el conjunto de asociaciones (*AS*), de forma que a cada especialización se le asigna una única asociación.

     $$AS_{EGS}: EGASS \rightarrow AS$$

   - **RL_{EGS} (Rol** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de asociación de subasociación (*EGASS*) y el conjunto de roles (*RL*), de forma que a cada especialización se le asigna un rol.

     $$RL_{EGS}: EGASS \rightarrow RL$$

   Para toda especialización debe cumplirse que su asociación (AS_{EGS}) sea una asociación en la que participe (dentro del origen o como destino) la superasociación de la clasificación, precisamente en el rol asignado a la especialización (RL_{EGS}).

   La expresión formal de la condición anterior es la siguiente: sea $EGASS_i \in ESP_{SBA}(SBA_j)$, sea $AS_x = AS_{EGS}(EGASS_i)$ y sea $RL_y = RL_{EGS}(EGASS_i)$. $EGASS_i$ es

válida si y sólo si el rol RL_y existe entre sus participantes (orígenes o destino) y su participante es la asociación AS_k (la superasociación):

$$EGASS_i \text{ válida} \Leftrightarrow (\exists RL_z \in OR_{AS}(AS_x) \cup \{DES_{AS}(AS_x)\} , RL_z = RL_y \wedge$$

$$P_{RL}(RL_z) = AS_k)$$

La existencia de la especialización implica que todas las instancias de la subasociación AS_l deberán aparecer al menos una vez en la asociación en la posición del rol correspondiente.

$$EGASS_i = (AS_x, RL_y) \Rightarrow \forall t_i \in N \bullet \forall IAS_z \in ST_{AS}(AS_l, t_i) \bullet \exists k \in N \bullet RL_Y = RL_{AS}(AS_x,k) \Rightarrow \exists IAS_i \in ST_{AS}(AS_x, t_i) \bullet \wedge IAS_z = IAS_i[k])$$

2. **Especialización de límite de participación en asociación** (ELASS). Cada especialización de este tipo indica que, dado un rol determinado de una asociación en la que participa la superasociación, el límite de participación de la subasociación se vuelve más restrictivo (es decir, se convierte en "único").

   Una especialización de este tipo se especifica mediante un par ordenado en el que se indican la asociación y el rol cuyo límite se modifica.

$$ELASS = \{ ELASS_i \mid ELASS_i \cong (AS_{ELS}(ELASS_i), RL_{ELS}(ELASS_i))\}$$

   Seguidamente se detallan los elementos de una especialización de límite de asociación:

   - **AS_{ELS} (Asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de asociación de subasociación (*ELASS*) y el conjunto de asociaciones (*AS*), de forma que a cada especialización se le asigna una asociación.

$$AS_{ELS}: ELASS \rightarrow AS$$

   - **RL_{ELS} (Rol** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de asociación de subasociación (*ELASS*) y el conjunto de roles (*RL*), de forma que a cada especialización se le asigna un rol.

$$RL_{ELS}: ELASS \rightarrow RL$$

   Para toda especialización debe cumplirse que su asociación (AS_{ELS}) sea una asociación en la que participe (como parte del origen o como destino) la superasociación de la clasificación, precisamente en el rol asignado a la especialización (RL_{ELS}).

   La expresión formal de la condición anterior es la siguiente: sea $ELASS_i \in ESP_{SBA}(SBA_j)$, sea $AS_x = AS_{EGS}(ELASS_i)$ y sea $RL_y = RL_{ELS}(ELASS_i)$. $ELASS_i$ es válida si y sólo si existe un rol entre sus participantes (orígenes o destino) igual a RL_y y su participante es la asociación AS_k (la superasociación):

$$ELASS_i \text{ válida} \Leftrightarrow (\exists RL_z \in OR_{AS}(AS_x) \cup \{DES_{AS}(AS_x)\} , RL_z = RL_y \wedge$$

$$P_{RL}(RL_z) = AS_k)$$

   La existencia de la especialización implica que todas las instancias del subconcepto AS_l podrán aparecer como máximo una vez en la asociación. Esta condición se formaliza de manera similar a la especialización de límite de atributo

3. **Especialización de grado de atributo de asociación** (EGATS). Cada especialización de este tipo indica que, dado un atributo de la superasociación, su grado se vuelve más restrictivo en la subasociación (es decir se convierte en "obligatorio").

   Una especialización de este tipo se especifica únicamente mediante el atributo cuyo grado se modifica.

   $$EGATS = \{ EGATS_i \mid EGATS_i \cong (ATS_{EGS}(EGATS_i))\}$$

   Una especialización de grado de atributo tiene sólo un elemento:

   - **ATS_{EGS} (Atributo de asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de grado de atributo de asociación (*EGATS*) y el conjunto de atributos de asociación (ATS), de forma que a cada especialización se le asigna un único atributo.

   $$ATS_{EGS}: EGATS \rightarrow ATS$$

   Para toda especialización debe cumplirse que su atributo (ATS_{EGS}) esté definido en la superasociación de la clasificación, bien directamente, o bien en alguno de sus antecesores (superasociaciones suyas en alguna clasificación).

   La expresión formal de la condición anterior es la siguiente: sea $EGATS_i \in ESP_{SBA}(SBA_j)$ y sea $ATS_x = ATS_{EGS}(EGATS_i)$. $EGATS_i$ es válida si y sólo si la superasociación (AS_k) es la asociación del atributo especializado, o bien, si la asociación del atributo especializado es uno de los antecesores de la superasociación:

   $$EGATS_i \text{ válida} \Leftrightarrow (AS_{ATS}(ATS_x) = AS_k \wedge AS_{ATS}(ATS_x) \in ANT_{AS}(AS_k))$$

   La existencia de la especialización implica que todas las instancias de la subasociación AS_l deberán tener al menos un valor definido para el atributo:

   $$EGATS_i \Rightarrow \forall t_i \in N \bullet \forall IAS_y \in ST_{AS}(AS_l, t_i) \bullet \exists V_z \in T_{ATS}(ATS_x) \bullet (IAS_y, V_z) \in ST_{ATS}(ATS_x, t_i)$$

4. **Especialización de límite de atributo de asociación** (ELATS). Cada especialización de este tipo indica que, dado un atributo de la superasociación, su límite se vuelve más restrictivo en la subasociación (es decir se convierte en "único").

   Una especialización de este tipo se especifica únicamente mediante el atributo cuyo límite se modifica.

   $$ELATS = \{ ELATS_i \mid ELATS_i \cong (ATS_{ELS}(ELATS_i))\}$$

   Una especialización de grado de atributo tiene sólo un elemento:

   - **ATS_{ELS} (Atributo de asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de límite de atributo de asociación (*ELATS*) y el conjunto de atributos de asociación (*ATS*), de forma que a cada especialización se le asigna un único atributo.

   $$ATS_{ELS}: ELATS \rightarrow ATS$$

Para toda especialización debe cumplirse que su atributo (ATS_{ELS}) esté definido en la superasociación de la clasificación, bien directamente, o bien en alguno de sus antecesores (superasociaciones suyos en alguna clasificación).

La expresión formal de la condición anterior es la siguiente: sea $ELATS_i \in ESP_{SBA}(SBA_j)$ y sea $ATS_x = ATS_{ELS}(ELATS_i)$. $ELATS_i$ es válida si y sólo si la superasociación (C_k) es la asociación del atributo especializado, o bien, si la asociación del atributo especializado es uno de los antecesores de la superasociación:

$$ELATS_i \text{ válida} \Leftrightarrow (AS_{ATS}(ATS_x) = AS_k \vee AS_{ATS}(ATS_x) \in ANT_{AS}(AS_k))$$

La existencia de la especialización implica que todas las instancias de la subasociación AS_l tendrán como máximo un valor definido para el atributo:

$$ELATS_i \Rightarrow \forall t_i \in N \bullet \forall IAS_y \in ST_{AS}(AS_l, t_i) \bullet \forall V_i, V_j \in T_{ATS}(ATS_x) \bullet V_i \neq V_j \wedge (IAS_y, V_i) \in ST_{ATS}(ATS_x, t_i) \Rightarrow (IAS_y, V_j) \notin ST_{ATS}(ATS_x, t_i)$$

5. **Especialización de tipo de atributo de asociación** (ETATS). Cada especialización de este tipo indica que, dado un atributo de la superasociación, su tipo se vuelve más restrictivo en la subasociación. Para ello el tipo definido en la especialización del atributo debe ser un tipo derivado (directa o indirectamente) del tipo definido en el atributo de la subasociación.

   Una especialización de este tipo se especifica mediante el atributo cuyo tipo se restringe y el nuevo tipo.

   $$ETATS = \{ ETATS_i \mid ETATS_i = (ATS_{ETS}(ETATS_i), T_{ETS}(ETATS_i))\}$$

   Una especialización de grado de atributo tiene dos elementos:

   - **ATS_{ETS} (Atributo de asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de tipo de atributo de asociación (*ETATS*) y el conjunto de atributos de asociación (*ATS*), de forma que a cada especialización se le asigna un único atributo.

     $$ATS_{ETS}: ETATS \rightarrow ATS$$

   - **T_{ETS} (Tipo** de la especialización). Es una aplicación entre el conjunto de especializaciones de tipo de atributo de asociación (*ETATS*) y el conjunto de tipos (*T*), de forma que a cada especialización se le asigna un único tipo.

     $$T_{ETS}: ETATS \rightarrow T$$

   Para toda especialización debe cumplirse que su atributo (ATS_{ETS}) esté definido en la superasociación de la clasificación, bien directamente, o bien en alguno de sus antecesores. Además, el tipo de la especialización debe ser un tipo derivado del tipo definido en la superasociación.

   La expresión formal de la condición anterior es la siguiente: sea $ETATS_i \in ESP_{SBA}(SBA_j)$, sea $ATS_x = ATS_{ETS}(ETATS_i)$ y sea $T_y = T_{ETS}(ETATS_i)$. $ETATS_i$ es válida si y sólo si la superasociación (AS_k) es la asociación del atributo especializado, o bien, si la asociación del atributo especializado es uno de los antecesores de la superasociación y, además, el tipo de la especialización es derivado del tipo del atributo:

$$ETATS_i \text{ válida} \Leftrightarrow (AS_{ATS}(ATS_x) = AS_k \vee AS_{ATS}(ATS_x) \in ANT_{AS}(AS_k)) \wedge T_{ATS}(ATS_x) \in ANT_T(T_y)$$

La existencia de la especialización implica que los valores asociados a las instancias de la subasociación AS_l deben pertenecer al nuevo tipo:

$$ETATS_i \Rightarrow \forall t_i \in N \bullet \forall IAS_y \in ST_{AS}(AS_l, t_i) \bullet \forall V_z \in T_{ATS}(ATS_x) \bullet (IAS_y, V_z) \in ST_{ATS}(ATS_x, t_i) \Rightarrow V_z \in T_y$$

6. **Especialización de valor por omisión de atributo de asociación** (EVATS). Cada especialización de este tipo indica que, dado un atributo de la superasociación, el valor por omisión de ese atributo cambia para las instancias de la subasociación considerada. Una especialización de este tipo se especifica mediante el atributo especializado y su nuevo valor por omisión.

$$EVATS = \{ EVATS_i \mid EVATS_i = (ATS_{EVS}(EVATS_i), V_{EVS}(EVATS_i))\}$$

Una especialización de grado de atributo tiene dos elementos:

- **ATS_{EVS} (Atributo de asociación** de la especialización). Es una aplicación entre el conjunto de especializaciones de valor por omisión de atributo de asociación (*EVATS*) y el conjunto de atributos de asociación (*ATS*), de forma que a cada especialización se le asigna un atributo.

$$ATS_{EVS}: EVATS \rightarrow ATS$$

- **V_{EVS} (Valor por omisión** de la especialización). Es una aplicación entre el conjunto de especializaciones de valor por omisión de atributo de asociación (*EVAT*) y el conjunto de valores (*V*), de forma que a cada especialización se le asigna un único valor.

$$V_{EVS}: EVATS \rightarrow V$$

Para toda especialización debe cumplirse que su atributo (ATS_{EVS}) esté definido en la superasociación de la clasificación, bien directamente, o bien en alguno de sus antecesores. Además, el valor por omisión de la especialización debe pertenecer al tipo del atributo.

La expresión formal de la condición anterior es la siguiente: sea $EVATS_i \in ESP_{SBA}(SBA_j)$, sea $ATS_x = ATS_{EVS}(EVATS_i)$ y sea $V_y = V_{EVS}(EVATS_i)$. $EVATS_i$ es válida si y sólo si la superasociación (AS_k) es la asociación del atributo especializado, o bien, si la asociación del atributo especializado es uno de los antecesores de la superasociación y, además, el valor por omisión de la especialización pertenece al tipo del atributo:

$$EVATS_i \text{ válida} \Leftrightarrow (AS_{ATS}(ATS_x) = AS_k \vee AS_{ATS}(ATS_x) \in ANT_{AS}(AS_k)) \wedge V_y \in T_{ATS}(ATS_x)$$

La existencia de la especialización implica que el valor por omisión para las instancias de la subasociación AS_l es el valor correspondiente a la especialización.

Respecto a las especializaciones definidas en una subasociación, debe cumplirse que **el conjunto de especializaciones asignadas a una definición de subasociación sea consistente**. Para ello:

- Para cada asociación en la participa la superasociación no deben existir más de una especialización de grado y de límite para cada rol en el que participe la superasociación.
- Para cada atributo de la superasociación sólo debe existir una especialización de cada tipo: grado, límite, tipo del atributo y valor por omisión.
- Sólo se podrá especializar aquello que no esté restringido. Por ejemplo, no se podrá especializar el grado de un rol que ya es "obligatorio".

4.2.1.7.3 Restricciones afectadas por clasificaciones de asociación

Las restricciones sobre nombre de atributo y nombre de asociación se ven afectadas por la existencia de clasificaciones de asociación.

- **El nombre de atributo de asociación será único en toda jerarquía de herencia de la asociación en la que se define**. Si se define un atributo en una asociación determinada, no deberá existir otro atributo con el mismo nombre ni en esa asociación ni en cualquiera de los familiares de la asociación.

$$\forall ATS_i, ATS_j \in ATS, ATS_i \neq ATSj \wedge (AS_{ATS}(ATS_j) = AS_{ATS}(ATS_i) \vee AS_{ATS}(ATS_j) \in FAM_{AS}(AS_{ATS}(ATS_i))) \Rightarrow N_{ATS}(ATS_i) \neq N_{ATS}(ATS_j)$$

  Para expresar formalmente esta condición se ha utilizado la aplicación **familiares** de asociación (FAM_{AS}), que asigna a cada asociación el conjunto de todas las asociaciones que tienen algún tipo de relación de clasificación con la dada y que se define formalmente más abajo.

  Esta función se define calculando la unión de (a) los padres de la asociación, (b) los hijos de la asociación, (c) los familiares de los padres y (d) los familiares de los hijos.

- **El nombre de asociación será único en toda la jerarquía de la asociación o asociaciones que definen su origen**: no deberá existir otra asociación con el mismo nombre que tenga por origen alguno de los familiares de la asociación.

4.2.1.7.4 Antecesores, Sucesores y Familiares de asociación

En este apartado se van a definir formalmente algunas funciones útiles relacionadas con las clasificaciones que se han utilizado a lo largo de la descripción de las clasificaciones de concepto:

- *Padres* de una asociación: representa el conjunto de todas las superasociaciones directas de una asociación dada.
- *Antecesores* de una asociación: representa el conjunto de todos las superasociaciones, directas o no, de una asociación dada.
- *Hijos* de una asociación: representa el conjunto de todos las subasociaciones directas de una asociación dada.
- *Sucesores* de un asociación: representa el conjunto de todas las subasociaciones, directas o no, de una asociación dada.
- *Familiares* de una asociación: representa el conjunto de todas las asociaciones que tienen alguna relación mediante clasificaciones con la asociación dada. Este

conjunto incluye los hijos de la asociación, los padres de la misma y los familiares de ambos.

Todas estas funciones son aplicaciones entre el conjunto de las asociaciones (*AS*) y las partes del conjunto de asociaciones ($\Pi(AS)$).

La aplicación **padres de asociación** (PAD_{AS}) es una aplicación que asocia una asociación al conjunto (que puede ser vacío) de todos sus superasociaciones *directas*, para toda clasificación en la que la asociación dada es una subasociación:

$$PAD_{AS}: AS \rightarrow \Pi(AS)$$

$$PAD_{AS}(AS_i) = \{AS_j \in AS \mid \exists\, CLS_k \in CLS \text{ con } AS_j = SA_{CLS}(CLS_k) \wedge \exists\, SBA_l \in SBA_{CLS}(CLS_k) \text{ con } AS_i = AS_{SBA}(SBA_l)\}$$

La aplicación **antecesores de asociación** (ANT_{AS}) devuelve el conjunto de superasociaciones, directas o no, de una asociación dada: padres de la asociación, padres de los padres de la asociación, padres de los padres de los padres, y así hasta que no haya más clasificaciones.

$$ANT_{AS}: AS \rightarrow \Pi(AS)$$

Se puede definir de forma recursiva $ANT_{AS}(AS_i)$ como el conjunto de elementos que son padres directos de AS_i o bien antecesores de los padres directos de AS_i:

$$ANT_{AS}(AS_i) = \{AS_j \in AS \mid AS_j \in PAD_{AS}(AS_i) \vee \exists\, AS_k \in PAD_{AS}(AS_i) \text{ con } AS_j \in ANT_{AS}(AS_k)\}$$

La aplicación **hijos de asociación** (HIJ_{AS}) es una aplicación que asocia una asociación al conjunto (que puede ser vacío) de todas sus subasociaciones *directas*, para toda clasificación en la que la asociación dada es superasociación:

$$HIJ_{AS}: AS \rightarrow \Pi(AS)$$

$$HIJ_{AS}(AS_i) = \{AS_j \in AS \mid \exists\, CLS_k \in CLS \text{ con } AS_i = SA_{CLS}(CLS_k) \wedge \exists\, SBA_l \in SBA_{CLS}(CLS_k) \text{ con } AS_j = AS_{SBA}(SBA_l)\}$$

La aplicación **sucesores de asociación** (SUC_{AS}) devuelve el conjunto de subasociaciones, directas o no, de una asociación dada: hijos de la asociación, hijos de los hijos de la asociación, hijos de los hijos de los hijos, y así hasta que no haya más clasificaciones.

$$SUC_{AS}: AS \rightarrow \Pi(AS)$$

Se puede definir de forma recursiva $SUC_{AS}(AS_i)$ como el conjunto de elementos que son hijos directos de AS_i o bien sucesores de los hijos directos de AS_i:

$$SUC_{AS}(AS_i) = \{AS_j \in AS \mid AS_j \in HIJ_{AS}(AS_i) \vee \exists\, AS_k \in HIJ_{AS}(AS_i) \text{ con } AS_j \in SUC_{AS}(AS_k)\}$$

La aplicación **familiares de asociación** (FAM_{AS}) devuelve el conjunto de todas las asociaciones que tienen algún tipo de relación de clasificación con la asociación dada: padres, familiares de padres, hijos y familiares de los hijos.

$$FAM_{AS}: AS \rightarrow \Pi(AS)$$

Se puede definir de forma recursiva $FAM_{AS}(AS_i)$ como el conjunto de elementos (distintos de AS_i) que son padres o hijos directos de AS_i o bien son familiares de los padres o hijos directos de AS_i:

$$FAM_C(C_i) = \{AS_j \in AS \mid AS_j \neq AS_i \wedge (AS_j \in PAD_{AS}(AS_i) \vee AS_j \in HIJ_{AS}(AS_i) \vee \exists AS_k \in PAD_{AS}(AS_i) \text{ con } AS_j \in FAM_{AS}(AS_k) \vee \exists AS_l \in HIJ_{AS}(AS_i) \text{ con } AS_j \in FAM_{AS}(AS_l))\}$$

4.2.1.8 Formalización de estados

A lo largo de la formalización de los elementos que definen el modelo estructural se han definido cuatro funciones que definen el estado de algunos elementos del modelo estructural en un instante de tiempo, como conjuntos de las instancias que contienen en ese momento. Estas funciones son:

- *Estado de un concepto* en un instante de tiempo (ST_C).
- *Estado de un atributo de concepto* en un instante de tiempo (ST_{AT}).
- *Estado de una asociación* en un instante de tiempo (ST_{AS}).
- *Estado de un atributo de asociación* en un instante de tiempo (ST_{ATS}).

En este apartado se va a formalizar la constitución de estados del sistema, usando como punto de partida las cuatro funciones anteriores.

En primer lugar se van a definir conjuntos formados por todos los estados de elementos del modelo estructural de un mismo tipo:

- Se define el **estado de todos los conceptos de un modelo estructural** en un instante (**STC_{EST}**) como una aplicación entre el producto cartesiano de los modelos estructurales con los números naturales y el universo de estados de concepto (STC).

  $$STC_{EST}: EST \times N \rightarrow STC$$

  Se define el **universo de estados de concepto** (STC) como un conjunto de elementos que son conjuntos de conjuntos de instancias de concepto (es decir, son subconjuntos de $\Pi(I)$).

  $$STC = \{STC_i \subset \Pi(I)\}$$

  Para todo modelo estructural y para todo instante de tiempo, el resultado de esta función es el conjunto de todos los estados de todos los conceptos del modelo estructural en ese instante.

  $$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall C_i \in C_{EST}(EST_i) \bullet ST_C(C_i, t_i) \in STC_{EST}(EST_i, t_i)$$

- Se define el **estado de todos los atributos de concepto de un modelo estructural** en un instante (**$STAT_{EST}$**) como una aplicación entre el producto cartesiano de los modelos estructurales con los números naturales y el universo de estados de atributos de concepto (STAT).

  $$STAT_{EST}: EST \times N \rightarrow STAT$$

  Se define el **universo de estados de atributo de concepto** (STAT) como un conjunto de elementos que son conjuntos de conjuntos de instancias de atributos (es decir, son subconjuntos de $\Pi(IAT)$).

  $$STAT = \{STAT_i \subset \Pi(IAT)\}$$

  Para todo modelo estructural y para todo instante de tiempo, el resultado de esta función es el conjunto de todos los estados de todos los atributos de concepto del modelo estructural en ese instante.

$$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall AT_i \in AT_{EST}(EST_i) \bullet ST_{AT}(AT_i, t_i) \in STAT_{EST}(EST_i, t_i)$$

- Se define el **estado de todas las asociaciones de un modelo estructural** en un instante (**$STAS_{EST}$**) como una aplicación entre el producto cartesiano de los modelos estructurales con los números naturales y el universo de estados de asociación (STAS).

  $$STAS_{EST}: EST \times N \rightarrow STAS$$

  Se define el **universo de estados de asociación** (STAS) como un conjunto de elementos que son conjuntos de conjuntos de instancias de asociación (es decir, son subconjuntos de $\Pi(IAS)$).

  $$STAS = \{STAS_i \subset \Pi(IAS)\}$$

  Para todo modelo estructural y para todo instante de tiempo, el resultado de esta función es el conjunto de todos los estados de todas las asociaciones del modelo estructural en ese instante.

  $$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall AS_i \in AS_{EST}(EST_i) \bullet ST_{AS}(AS_i, t_i) \in STAS_{EST}(EST_i, t_i)$$

- Se define el **estado de todos los atributos de asociación de un modelo estructural** en un instante (**$STATS_{EST}$**) como una aplicación entre el producto cartesiano de los modelos estructurales con los números naturales y el universo de estados de atributos de asociación (STATS).

  $$STATS_{EST}: EST \times N \rightarrow STATS$$

  Se define el **universo de estados de atributo de asociación** (STATS) como un conjunto de elementos que son conjuntos de conjuntos de instancias de atributos de asociación(es decir, son subconjuntos de $\Pi(IATS)$).

  $$STATS = \{STATS_i \subset \Pi(IATS)\}$$

  Para todo modelo estructural y para todo instante de tiempo, el resultado de esta función es el conjunto de todos los estados de todos los atributos de asociación del modelo estructural en ese instante.

  $$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall ATS_i \in ATS_{EST}(EST_i) \bullet ST_{ATS}(ATS_i, t_i) \in STATS_{EST}(EST_i, t_i)$$

Partiendo de estas cuatro funciones se puede definir el **estado de un modelo estructural** en un instante de tiempo (ST_{EST}) como una aplicación entre el producto cartesiano de los modelos estructurales con los números naturales y el universo de estados de modelo estructural ST.

$$ST_{EST}: EST \times N \rightarrow ST$$

El **universo de estados de modelo estructural** (ST) es un conjunto de elementos, cada uno de los cuales es una cuaterna formada por un elemento del universo de estados de concepto, un elemento del universo de atributos de concepto, un elemento del universo de estados de asociación y un elemento del universo de estados de atributo de asociación.

$$ST = \{ST_i = (STC_i, STAT_i, STAS_i, STATS_i) \mid STC_i \in STC \wedge STAT_i \in STAT \wedge STAS_i \in STAS \wedge STATS_i \in STATS\}$$

Es decir, cada elemento de ST es una cuaterna de conjuntos y cada uno de esos conjuntos contiene a su vez conjuntos de instancias.

Para todo modelo estructural y todo instante de tiempo, su estado es la cuaterna formada por sus estados de conceptos, de atributos de concepto, de asociaciones y de atributos de asociación en ese mismo instante de tiempo:

$$\forall t_i \in N \bullet \forall EST_i \in EST \bullet ST_{EST}(EST_i, t_i) = (STC_{EST}(EST_i, t_i), STAT_{EST}(EST_i, t_i), STAS_{EST}(EST_i, t_i), STATS_{EST}(EST_i, t_i))$$

4.2.2 Formalización del Modelo de Comportamiento

El **universo de modelos de comportamiento** (**COM**) es un conjunto de modelos de comportamiento (COM_i), cada uno de los cuales se representa mediante un par: tareas y métodos de tarea.

$$COM = \{COM_i \mid COM_i \cong (TA_{COM}(COM_i), M_{COM}(COM_i))\}$$

Los elementos que definen un modelo estructural son:

- **TA_{COM}** (**Tareas** del modelo de comportamiento) es una función entre el conjunto los modelos de comportamiento (*COM*) y las partes del conjunto de las tareas ($\Pi(TA)$), de forma que a cada modelo de comportamiento se le asocia un conjunto de tareas:

  $$TA_{COM}: COM \rightarrow \Pi(TA)$$

- **M_{COM}** (**Métodos** del modelo de comportamiento) es una función entre el conjunto los modelos de comportamiento (*COM*) y las partes del conjunto de los métodos de tarea ($\Pi(M)$), de forma que a cada modelo estructural se le asocia un conjunto de métodos:

  $$M_{COM}: COM \rightarrow \Pi(M)$$

Sobre estos elementos se definen una serie de restricciones (por ejemplo, la tarea principal de un método debe pertenecer al modelo de comportamiento en el que se define el método) que se formalizarán según se vaya formalizando cada uno de los elementos en los apartados siguientes.

Además de las tareas y métodos, también se van a formalizar los operadores y funciones de consulta, que son elementos predefinidos que pueden utilizarse como parte de la especificación de control de los métodos.

4.2.2.1 Formalización de Tareas

El **universo de tareas** (TA) es un conjunto formado tareas, que son correspondencias del universo de estados en sí mismo y que se especifican mediante cinco funciones: nombre, entrada, precondición, salida y postcondición:

$$TA = \{TA_i \subset ST \times ST \mid TA_i \cong (N_{TA}(TA_i), ENT_{TA}(TA_i), PRE_{TA}(TA_i), SAL_{TA}(TA_i), POS_{TA}(TA_i))\}$$

- **N_{TA}** (**Nombre** de tarea) es una aplicación inyectiva entre el conjunto de tareas (*TA*) y el conjunto de cadenas (*CAD*), de forma que a cada tarea se le asocia un nombre:

  $$N_{TA}: TA \rightarrow CAD$$

  El nombre de una tarea debe ser único dentro de un modelo de comportamiento (dos tareas diferentes deben tener nombres distintos):

$$\forall COM_i \in COM \bullet \forall TA_i, TA_j \in TA_{COM}(COM_i) \bullet N_{TA}(TA_i) = N_{TA}(TA_j) \Leftrightarrow TA_i = TA_j$$

Así el nombre de una tarea podrá utilizarse como su identificador dentro del contexto de un modelo de comportamiento.

- **ENT_{TA}** (**Entrada** de tarea) es una aplicación entre el conjunto de tareas (*TA*) y las partes del conjunto de variables ($\Pi(VAR)$), de forma que a cada tarea se le asocia un conjunto de variables que necesitan ser asignadas para poder evaluar la precondición de la tarea.

$$ENT_{TA}: TA \rightarrow \Pi(VAR)$$

La entrada de una tarea está formada un conjunto de variables libres (sin cuantificar):

$$\forall TA_i \in TA \bullet \forall VAR_j \in ENT_{TA}(TA_i) \bullet CUA_{VAR}(VAR_j) = \varnothing$$

Para toda conceptualización, las variables de la entrada de todas las tareas de su modelo de comportamiento deben tomar valores en tipos, conceptos o asociaciones del modelo estructural de esa conceptualización.

$$\forall CNC_i \in CNC \bullet \forall TA_i \in TA_{COM}(COM_{CNC}(CNC_i)) \bullet \forall VAR_j \in ENT_{TA}(TA_i) \bullet CJTO_{VAR}(VAR_j) \in T_{EST}(EST_{CNC}(CNC_i)) \cup C_{EST}(EST_{CNC}(CNC_i)) \cup AS_{EST}(EST_{CNC}(CNC_i))$$

Las variables se formalizan dentro del apartado dedicado a la formalización de condiciones lógicas.

- **PRE_{TA}** (**Precondición** de tarea) es una aplicación inyectiva entre el conjunto de tareas (*TA*) y el conjunto de condiciones lógicas sobre un estado (*CND*), de forma que a cada tarea se le asocia una condición unaria que deberá ser cierta en el estado inicial (incluyendo la entrada) para poder llevar a cabo la tarea.

$$PRE_{TA}: TA \rightarrow CND$$

La precondición de una tarea tendrá como variables libres los elementos de la entrada de la tarea y no podrá tener ninguna variable libre más.

Para toda conceptualización, las precondición de todas las tareas de su modelo de comportamiento únicamente podrán hacer referencia a los elementos del modelo estructural de esa conceptualización.

- **SAL_{TA}** (**Salida** de tarea) es una aplicación entre el conjunto de tareas (*TA*) y las partes del conjunto de variables ($\Pi(VAR)$), de forma que a cada tarea se le asocia un conjunto de variables cuyo valor será conocido después de haber realizado la tarea.

$$SAL_{TA}: TA \rightarrow \Pi(VAR)$$

La salida de una tarea está formada un conjunto de variables libres (sin cuantificar):

$$\forall TA_i \in TA \bullet \forall VAR_j \in SAL_{TA}(TA_i) \bullet CUA_{VAR}(VAR_j) = \varnothing$$

Para toda conceptualización, las variables de la salida de todas las tareas de su modelo de comportamiento deben tomar valores en tipos, conceptos o asociaciones del modelo estructural de esa conceptualización.

$$\forall CNC_i \in CNC \bullet \forall TA_i \in TA_{COM}(COM_{CNC}(CNC_i)) \bullet \forall VAR_j \in SAL_{TA}(TA_i) \bullet CJTO_{VAR}(VAR_j) \in T_{EST}(EST_{CNC}(CNC_i)) \cup C_{EST}(EST_{CNC}(CNC_i)) \cup AS_{EST}(EST_{CNC}(CNC_i))$$

- **POS_{TA}** (**Postcondición** de tarea) es una aplicación inyectiva entre el conjunto de tareas (*TA*) y el conjunto de condiciones lógicas binarias (*CNB*), de forma que a cada tarea se le asocia una condición que deberá ser cierta comparando los estados inicial y final de la tarea.

  $$POS_{TA}: TA \rightarrow CNB$$

  La postcondición de una tarea tendrá como variables libres los elementos de la entrada y la salida de la tarea y no podrá tener ninguna variable libre más.

Se define el **universo de instancias de tarea (ITA)** como el conjunto de todas las instancias de tarea posibles, que son pares ordenados formado por dos estados.

$$ITA = \{ITA_i = (ST_i, ST_j) \mid ST_i \in ST \wedge ST_j \in ST\}$$

Una instancia de tarea representa la resolución de una tarea partiendo del estado concreto del modelo estructural en un instante determinado. El primer estado se denomina **estado inicial** de la instancia de tarea y el segundo estado se denomina **estado final** de la instancia de tarea.

Una vez definidos los elementos que permiten especificar una tarea, es el momento de definir formalmente la **condición de pertenencia de dos estados a una tarea**, que dice lo siguiente:

Para toda conceptualización CNC_i, para toda tarea TA_i de su modelo de comportamiento y para todo par instantes de tiempo t_i, t_f ($t_i < t_f$), el par ($ST_{EST}(EST_{CNC}(CNC_i), t_i)$, $ST_{EST}(EST_{CNC}(CNC_i), t_f)$ pertenece a la tarea TA_i si y sólo si:

- existe un conjunto de asignación de variables para la entrada ASS_e de forma que todas sus variables sean variables de la entrada de la tarea y su valor pertenezca al estado del modelo estructural en el estado inicial (teniendo en cuenta que las variables pueden representar elementos de tipos, conceptos y asociaciones o incluso subconjuntos de tipos, conceptos y asociaciones),
- y la evaluación de la precondición en el instante t_i usando el conjunto de asignaciones de variables ASS_e es cierta,
- y existe un conjunto de asignaciones de variables para la salida ASS_s de forma que todas sus variables sean variables de la salida de la tarea y su valor pertenezca al estado del modelo estructural en el estado final,
- y la evaluación de la postcondición en los instantes de tiempo t_i, t_f con la unión entre las asignaciones de entrada y de salida ($ASS_e \cup ASS_s$) es cierta.

Las asignaciones de variables están formalizadas en el apartado dedicado a la formalización de condiciones lógicas.

Expresado formalmente:

$$\forall CNC_i \in CNC \bullet \forall TA_i \in TA_{COM}(COM_{CNC}(CNC_i)) \bullet \forall t_i, t_f \in N \bullet (ST_{EST}(EST_{CNC}(CNC_i), t_i), ST_{EST}(EST_{CNC}(CNC_i), t_f) \in TA_i \Leftrightarrow$$

$$(t_i > t_f \wedge \exists ASS_e \in \Pi(ASV) \bullet (\forall ASV_x \in ASS_e\ VAR_{ASV}(ASV_x) \in ENT_{TA}(TA_i)$$
$$\wedge$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in C \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \in ST_C(CJTO_{VAR}(VAR_{ASV}(ASV_x), t_i)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in C \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \subset ST_C(CJTO_{VAR}(VAR_{ASV}(ASV_x), t_i)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in T \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \in CJTO_{VAR}(VAR_{ASV}(ASV_x)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in T \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \subset CJTO_{VAR}(VAR_{ASV}(ASV_x)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in AS \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \in ST_{AS}(CJTO_{VAR}(VAR_{ASV}(ASV_x), t_i)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_x)) \in AS \wedge TP_{VAR}(VAR_{ASV}(ASV_x)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_x)) \subset ST_{AS}(CJTO_{VAR}(VAR_{ASV}(ASV_x), t_i)))$$
$$\wedge$$
$$EVAL_{CND}(EST_i, PRE_{TA}(TA_i), t_i, ASS_e) = v$$
$$\wedge$$
$$\exists ASS_s \in \Pi(ASV) \bullet (\forall ASV_y \in ASS_s\ VAR_{ASV}(ASV_y) \in SAL_{TA}(TA_i)$$
$$\wedge$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in C \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \in ST_C(CJTO_{VAR}(VAR_{ASV}(ASV_y), t_f)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in C \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \subset ST_C(CJTO_{VAR}(VAR_{ASV}(ASV_y), t_f)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in T \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \in CJTO_{VAR}(VAR_{ASV}(ASV_y)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in T \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \subset CJTO_{VAR}(VAR_{ASV}(ASV_y)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in AS \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“simple”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \in ST_{AS}(CJTO_{VAR}(VAR_{ASV}(ASV_y), t_f)))$$
$$\vee$$
$$(CJTO_{VAR}(VAR_{ASV}(ASV_y)) \in AS \wedge TP_{VAR}(VAR_{ASV}(ASV_y)) = \text{“conjunto”} \Rightarrow$$
$$VAL_{VAR}(VAR_{ASV}(ASV_y)) \subset ST_{AS}(CJTO_{VAR}(VAR_{ASV}(ASV_y), t_f)))$$
$$\wedge$$
$$EVAL_{CNB}(EST_i, POS_{TA}(TA_i), t_i, t_f, ASS_e \cup ASS_s) = v)$$

Teniendo en cuenta la condición anterior, la **resolución de una tarea** TA_i consiste en el siguiente planteamiento de problema:

- DADO
  - una tarea TA_i,
  - un instante de tiempo inicial t_i,
  - una asignación de variables para la entrada ASS_e,
  - tales que la evaluación de la precondición de la tarea en ese instante y con esa asignación de variables es cierta
- HACER
  - definir el instante final t_f y los elementos de su estado correspondiente,
  - encontrar dentro del estado final los valores de la asignación de variables ASS_s,
  - tales que la evaluación de la postcondición de la tarea en los instantes inicial y final y con la unión entre las asignaciones ASS_e y ASS_s sea cierta.

Equipo de fútbol

Se muestran tres ejemplos de tareas.

Mejorar la fuerza de un jugador en un incremento determinado, proporcionando el jugador y el incremento de fuerza deseado.

```
TA("Mejorar fuerza jugador",

 {VAR("jug", ∅, C"Jugador", "simple"),
  VAR("inc", ∅, T"Real", "simple")},

 CND(
  existe f1 de T"Valoración"
    (jug, f1) pertenece a AT"Jugador"."fuerza" and
    f1 < 10 and inc > 0 and f1 + inc <= 10
  fin condición
 ),

 ∅,

 CNB(
  existe f1 de T"Valoración"
    (jug, f1) pertenece a AT"Jugador"."fuerza" en [1] and
   not (jug, f1) pertenece a AT"Jugador"."fuerza" en [2] and
    (jug, f1 + inc) pertenece a AT"Jugador"."fuerza" en [2]
  fin condición
 ),
)
```

Crear un jugador de campo desde cero, proporcionando todos sus datos (nombre completo, país de origen, fecha de nacimiento, posición, dorsal, fuerza, velocidad, resistencia, nombre del equipo al que pertenece, fecha de comienzo del fichaje, importe de la ficha, cláusula de rescisión, valoración de robos, centros, regate, remate con el pie y remate con la cabeza).

```
TA("Crear jugador campo",

  {VAR("nombre",∅,T"CAD","simple"),VAR("país",∅,T"País","simple"),
   VAR("dn",∅,T"Día","simple"),VAR("mn",∅,T"Mes","simple"),
   VAR("an",∅,T"Año","simple"),VAR("pos",∅,T"Posición","simple"),
   VAR("dorsal",∅,T"Dorsal","simple"),
   VAR("fuerza",∅,T"Valoración","simple"),
   VAR("vel",∅,T"Valoración","simple"),
   VAR("res",∅,T"Valoración","simple"),
```

```
    VAR("nequipo",∅,T"CAD","simple"),VAR("df",∅,T"Día",simple),
    VAR("mf",∅,T"Mes",simple),VAR("af",∅,T"Año","simple"),
    VAR("ficha",∅,T"Z","simple"),VAR("rescisión",∅,T"Z","simple"),
    VAR("robos", ∅, T"Valoración", "simple"),
    VAR("centros", ∅, T"Valoración", "simple"),
    VAR("regate", ∅, T"Valoración", "simple"),
    VAR("rpie", ∅, T"Valoración", "simple"),
    VAR("rcabeza", ∅, T"Valoración", "simple")}

  CND(
  existe equipo de C"Equipo"
    pos pertenece a T"Posición campo" and
    (equipo, nequipo) pertenece a AT"Equipo"."nombre" and
    para todo jug de C"Jugador"
     not (jug, nombre) pertenece a AT"Persona"."nombre" and
     si
      existe f de C"Fecha"
       (jug,f,equipo) pertenece a AS("Jugador","Fecha")."Fichaje"
      fin condición
     entonces
      not (jug, dorsal) pertenece a AT"Jugador"."dorsal" and
     fin si
    fin condición
   fin condición
  ),

  {VAR("jug" , ∅, C"Jugador campo", "simple"),
   VAR("fn",∅, C"Fecha", "simple"), VAR("ff",∅, C"Fecha", "simple")},

  CND(
   existe equipo de C"Equipo"
    (equipo, nequipo) pertenece a AT"Equipo"."nombre" en [2] and
    (jug,nombre) pertenece a AT"Persona"."nombre" en [2] and
    (jug,país) pertenece a AT"Persona"."país" en [2] and
    si
     país pertenece a T"País nacional"
    entonces
     jug pertenece a C"Nacional" en [2]
    fin si and
    si
     país pertenece a T"País comunitario"
    entonces
     jug pertenece a C"Comunitario" en [2]
    fin si and
    si
     país pertenece a T"País extranjero"
    entonces
     jug pertenece a C"Extranjero" en [2]
    fin si and
    (fn,dn) pertenece a AT"Fecha"."día" en [2] and
    (fn,mn) pertenece a AT"Fecha"."mes" en [2] and
    (fn,an) pertenece a AT"Fecha"."año" en [2] and
    (jug,fn) pertenece a AS("Persona")."Nacimiento" en [2] and
    (jug,pos) pertenece a AT"Jugador"."posición" en [2] and
    (jug,dorsal) pertenece a AT"Jugador"."dorsal" en [2] and
    (jug,fuerza) pertenece a AT"Jugador"."fuerza" en [2] and
    (jug,vel) pertenece a AT"Jugador"."velocidad" en [2] and
    (jug,res) pertenece a AT"Jugador"."resistencia" en [2] and
    (ff,df) pertenece a AT"Fecha"."día" en [2] and
    (ff,mf) pertenece a AT"Fecha"."mes" en [2] and
    (ff,af) pertenece a AT"Fecha"."año" en [2] and
    (jug,ff,equipo) pertenece a AS("Jugador,"Fecha")."Fichaje" en [2]
     and ((jug,ff,equipo),ficha) pertenece a
      ATS("Jugador, "Fecha")."Fichaje"."ficha" en [2] and
    ((jug,ff,equipo),rescisión) pertenece a
      ATS("Jugador, "Fecha")."Fichaje"."rescisión" en [2] and
```

```
      (jug,robos) pertenece a AT"Jugador campo"."robos" en [2] and
      (jug,centros) pertenece a AT"Jugador campo"."centros" en [2] and
      (jug,regate) pertenece a AT"Jugador campo"."regate" en [2] and
      (jug,rpie) pertenece a AT"Jugador campo"."remate pie" en [2] and
      (jug,rcabeza) pertenece a AT"Jugador campo"."remate cabeza" en [2]
     fin condición
    )
   )
```

Crear la alineación de un equipo para un partido concreto.

```
   TA("Crear alineación de equipo",

     {VAR("eq",∅, C"Equipo", "simple"), VAR("da", ∅, T"Día", "simple"),
      VAR("ma", ∅, T"Mes", "simple"), VAR("aa", ∅, T"Año", "simple")
    },

    CND(
     para todo al de C"Alineación"
      para todo f de C"Fecha"
       si
        (eq,f,al) pertenece a AS("Equipo","Fecha")."Convocatoria"
       entonces
        not (f,da) pertenece a AT"Fecha"."día" or
        not (f,ma) pertenece a AT"Fecha"."mes" or
        not (f,aa) pertenece a AT"Fecha"."año"
       fin si
      fin condición
     fin condición
    ),

    {
    VAR("aln",∅,C"Alineación","simple"),VAR("t",∅,C"Táctica","simple"),
    VAR("fa", ∅, C"Fecha", "simple")
    },

    CND(
     (fa,da) pertenece a AT"Fecha"."día" en [2] and
     (fa,ma) pertenece a AT"Fecha"."mes" en [2] and
     (fa,aa) pertenece a AT"Fecha"."año" en [2] and
     (eq,fa,aln) pertenece a
         AS("Equipo","Fecha")."Convocatoria" en[2] and
     (aln,t) pertenece a AS("Alineación")."Táctica prevista" en [2] and
     para todo puesto de C"Puesto"
      si
       (t,puesto) pertenece a AS("Táctica")."Puestos" en [2]
      entonces
       existe jug de C"Jugador"
        (aln, puesto, jug) pertenece a
           AS("Alineación","Puesto")."Asignación" en [2] and
       existe pos de T"Jugador"
        (jug,pos) pertenece a AT"Jugador"."posición" en [2] and
        ((puesto,pos") pertenece a AT"Puesto"."posición" en [2] or
         (puesto",pos") pertenece a AT"Puesto"."afines") en [2]
       fin condición
      fin condición
     fin si
    fin condición
    )
   )
```

4.2.2.2 Formalización de Métodos de Tarea

La formalización de los métodos de tarea se ha dividido en tres apartados para facilitar su lectura. En el primero se formalizarán los métodos, en el segundo se formalizarán los

elementos que componen una especificación de control y en el último apartado se describirá la semántica de ejecución de métodos.

4.2.2.2.1 Formalización de métodos

El **universo de métodos de tarea (M)** como el conjunto de todos los métodos de tarea, cada uno de los cuales se especifica una cuaterna formada por el nombre, la tarea que permite resolver, las subtareas utilizadas por el método y la especificación del control ejercido por el método.

$$M = \{M_i \mid M_i \cong (N_M(M_i), TA_M(M_i), STA_M(M_i), CN_M(M_i))\}$$

- $\mathbf{N_M}$ (**Nombre** de método) es una aplicación inyectiva entre el conjunto de métodos (*M*) y el conjunto de cadenas (*CAD*), de forma que a cada método se le asocia un único nombre:

  $$N_M: M \rightarrow CAD$$

  El nombre de un método debe ser único dentro de un modelo de comportamiento (dos métodos diferentes deben tener nombres distintos):

  $$\forall COM_i \in COM \bullet \forall M_i, M_j \in M_{COM}(COM_i) \bullet N_M(M_i) = N_M(M_j) \Leftrightarrow M_i = M_j$$

  Así el nombre de un método podrá utilizarse como su identificador dentro del contexto de un modelo de comportamiento.

- $\mathbf{TA_M}$ (**Tarea** de método) es una aplicación inyectiva entre el conjunto de métodos (*M*) y el conjunto de tareas (*TA*), de forma que a cada método se le asocia la tarea que puede resolver. Esta tarea se denomina tarea principal del método.:

  $$TA_M: M \rightarrow TA$$

  Pueden existir varios métodos distintos para una misma tarea.

  Para toda conceptualización y para todo método de tarea de su modelo de comportamiento, la tarea principal del método también debe pertenecer a ese modelo de comportamiento.

  $$\forall CNC_i \in CNC \bullet \forall M_i \in M_{COM}(COM_{CNC}(CNC_i)) \bullet TA_M(M_i) \in T_{COM}(COM_{CNC}(CNC_i))$$

- $\mathbf{STA_M}$ (**Subtareas** de método) es una aplicación inyectiva entre el conjunto de métodos (*M*) y las partes del conjunto de tareas ($\Pi(TA)$), de forma que a cada método se le asocia un conjunto de tareas que son subtareas de la tarea principal del método:

  $$STA_M: M \rightarrow \Pi(TA)$$

  Para toda conceptualización y para todo método de tarea de su modelo de comportamiento, el conjunto de sus subtareas debe estar incluido en el conjunto de tareas de ese modelo de comportamiento.

  $$\forall CNC_i \in CNC \bullet \forall M_i \in M_{COM}(COM_{CNC}(CNC_i)) \bullet STA_M(M_i) \subseteq T_{COM}(COM_{CNC}(CNC_i))$$

- $\mathbf{CN_M}$ (**Control** de método) es una aplicación inyectiva entre el conjunto de métodos (*M*) y el universo de especificaciones de control de método (*CN*), de forma que a cada método se le asocia una especificación del control de la ejecución de las subtareas.

$$CN_M: M \rightarrow CN$$

CN es el universo de especificaciones de control, que permite especificar control imperativo, dirigido por eventos y heurístico.

Equipo de fútbol

Se muestra a continuación un **método para resolver la tarea "Crear alineación equipo"**. El método consiste en elegir en primer lugar una táctica y después asignar los mejores jugadores posibles a los puestos de esa táctica. Elementos de este método:

```
M(
  "Partiendo de táctica",
  "Crear alineación equipo",
  {"Elegir táctica", "Asignar jugadores"},
  CN(
    INICIO,
    RT("Elegir táctica", (eq), (t)),
    RT("Asignar jugadores", (eq, t, da, ma, aa), (aln, fa)),
    FIN
  )
)
```

4.2.2.2.2 Formalización del control

El **universo de especificaciones de control** (CN) está formado por especificaciones de control que son tuplas que contienen al menos un nodo de control.

$$CN = \{CN_i \subset \bigcup_{\substack{k \in N \\ k>1}} NC^k \}$$

En otras palabras una especificación de control está incluida en la unión de todas las potencias del universo de control.

La notación que se utilizará para acceder a las especificaciones de control es la siguiente:

- Número de elementos de una secuencia de control: $|CN_i|$
- Acceso al nodo k-ésimo de una secuencia de control: $CN_i[k]$, con $k \in N$, $1 \leq k \leq |CN_i|$

El **universo de nodos de control** (NC) es un conjunto de elementos que pueden ser de los tipos siguientes:

- Fin de una secuencia de control
- Declaración de una variable que podrá usarse en el resto de la secuencia
- Resolución de una tarea, pasándole como parámetros los valores de la entrada y recogiendo como resultado los valores de la salida.
- Ejecución de un operador
- Condición simple (*if – then*)
- Condición doble (*if – then - else*)
- Condición múltiple (*case*)
- Bucle mientras (*while*)

- Bucle repetir (*repeat*)
- Bucle para (*for*)
- Bucle para todo (*for all*)
- División en caminos paralelos
- Espera a evento (cambio de estado o condición sobre el tiempo)
- Control heurístico

$$NC = NF \cup NV \cup NT \cup NO \cup NCS \cup NCD \cup NCM \cup NBM \cup NBR \cup NBF \cup NBFA \cup NP \cup NEV \cup NH$$

En este apartado se van a formalizar cada uno de los nodos de control y en el siguiente se definirá su semántica.

- El **conjunto de nodos de control de fin** (NF) sólo contiene un único elemento (nf), que representa el final de la ejecución de una secuencia de control:

$$NF = \{nf\}$$

- El **universo de nodos de declaración de variable** (NV) contiene elementos que se especifican mediante una función que les asigna una declaración de variable que tendrá efecto en el resto de nodos de la secuencia.

$$NV = \{ NV_i \mid NV_i \cong VAR_{NV}(NV_i) \}$$

  - **VAR_{NV}** (**Variable** de nodo) es una función entre el universo de nodos de variable y el universo de declaraciones de variable:

$$VAR_{NV}: NV \rightarrow VAR$$

    Las variables declaradas en un nodo de control de variable no deberán estar cuantificadas:

$$\forall NV_i \in NV \bullet CUA_{VAR}(VAR_{NV}(NV_i)) = \varnothing$$

- El **universo de nodos de resolución de tarea** (NT), contiene elementos que se especifican mediante una terna de funciones: la tarea que se va a resolver, los parámetros que se le pasan como entrada y las variables que recogen la salida de la tarea.

$$NT = \{ NT_i \mid NT_i \cong (TA_{NT}(NT_i), ENT_{NT}(NT_i), SAL_{NT}(NT_i)) \}$$

  - **TA_{NT}** (**Tarea** de nodo) es una función entre el universo de nodos de tarea y el universo de tareas:

$$TA_{NT}: NT \rightarrow TA$$

    Para todo modelo de comportamiento y para todo noto de tarea que aparezca en la especificación de control de un método de ese modelo de comportamiento, su tarea debe pertenecer al conjunto de subtareas del método:

$$\forall COM_i \in COM \bullet \forall M_i \in M_{COM}(COM_i) \bullet \forall NT_i \in NC_{CN}(CN_M(M_i)) \bullet TA_{NT}(NT_i) \in TA_{COM}(COM_i)$$

    Para formalizar la condición anterior se ha utilizado la función NC_{CN}, que devuelve el **conjunto de nodos de control que aparece en una especificación**

de control (incluyendo los nodos que aparecen en las especificaciones de secuencia de otros nodos). Esta función se formalizará cuando termine la especificación de todos los nodos de control.

- **ENT_{NT}** (**Entrada** de nodo) es una función entre el universo de nodos de tarea y las partes del universo de asignaciones de variable, de forma que a cada nodo de tarea le asigna un conjunto de asignaciones de valores a las variables de la entrada de la tarea:

$$ENT_{NT}: NT \rightarrow \Pi(ASV)$$

Para todo nodo de tarea, cada una de las asignaciones de variable se debe corresponder con variables de la entrada de la tarea asociada al nodo:

$$\forall NT_i \in NT \bullet \forall ASV_i \in ENT_{NT}(NT_i) \bullet VAR_{ASV}(ASV_i) \in ENT_{TA}(TA_{NT}(NT_i))$$

Para todo nodo de tarea, los valores de cada una de las asignaciones de la entrada podrán ser:

- Valores concretos del tipo adecuado.
- Expresiones que devuelvan valores del tipo adecuado.
- Funciones de consulta que devuelvan elementos del tipo adecuado.
- Variables de la tarea principal del método al que pertenece el nodo.
- Variables definidas en nodos anteriores en la secuencia.

- **SAL_{NT}** (**Salida** de nodo) es una función entre el universo de nodos de tarea y las partes del universo de asignaciones de variable, de forma que a cada nodo de tarea le asigna un conjunto de asignaciones de valores a las variables de la salida de la tarea (que representa la recogida de valores de la salida):

$$SAL_{NT}: NT \rightarrow \Pi(ASV)$$

Para todo nodo de tarea, cada una de las asignaciones de variable se debe corresponder con variables de la salida de la tarea asociada al nodo:

$$\forall NT_i \in NT \bullet \forall ASV_i \in SAL_{NT}(NT_i) \bullet VAR_{ASV}(ASV_i) \in SAL_{TA}(TA_{NT}(NT_i))$$

Para todo nodo de tarea, los valores de cada una de las asignaciones de la salida serán variables donde se recojan los valores de la salida de la tarea correspondiente.

- El **universo de nodos de ejecución de operador** (NO), contiene elementos que se especifican mediante una terna de funciones: el operador que se va a ejecutar, los parámetros que se le pasan como entrada y las variables que recogen la salida del operador.

$$NO = \{ NO_i \mid NO_i \cong (OP_{NO}(NO_i), ENT_{NO}(NO_i), SAL_{NO}(NO_i)) \}$$

- **OP_{NO}** (**Operador** de nodo) es una función entre el universo de nodos de operador y el universo de operadores:

$$OP_{NO}: NO \rightarrow OP$$

- **ENT_{NO}** (**Entrada** de nodo) es una función entre el universo de nodos de operador y las partes del universo de asignaciones de variable, de forma que a cada nodo de operador le asigna un conjunto de asignaciones de valores a las variables de la entrada de del operador:

$$ENT_{NO}: NO \rightarrow \Pi(ASV)$$

Para todo nodo de operador, cada una de las asignaciones de variable se debe corresponder con variables de la entrada del operador asociado al nodo:

$$\forall NO_i \in NO \bullet \forall ASV_i \in ENT_{NO}(NO_i) \bullet VAR_{ASV}(ASV_i) \in ENT_{OP}(OP_{NO}(NO_i))$$

Para todo nodo de operador, los valores de cada una de las asignaciones de la entrada podrán ser:

- Valores concretos del tipo adecuado.
- Expresiones que devuelvan valores del tipo adecuado.
- Funciones de consulta que devuelvan elementos del tipo adecuado.
- Variables de la tarea principal del método al que pertenece el nodo.
- Variables definidas en nodos anteriores en la secuencia.

- **SAL_{NO}** (**Salida** de nodo) es una función entre el universo de nodos de operador y las partes del universo de asignaciones de variable, de forma que a cada nodo de operador le asigna un conjunto de asignaciones de valores a las variables de la salida del operador (si el operador tiene salida), con lo que se representa la recogida de valores de la salida:

$$SAL_{NO}: NO \rightarrow \Pi(ASV)$$

Para todo nodo de operador, cada una de las asignaciones de variable se debe corresponder con variables de la salida de del operador asociado al nodo:

$$\forall NO_i \in NO \bullet \forall ASV_i \in SAL_{NO}(NO_i) \bullet VAR_{ASV}(ASV_i) \in SAL_{OP}(OP_{NO}(NO_i))$$

Para todo nodo de operador, los "valores" de cada una de las asignaciones de la salida serán variables donde se recojan los valores de la salida del operador correspondiente.

- El **universo de nodos de condición simple** (NCS), contiene elementos que se especifican mediante un par de funciones: la condición que debe evaluarse y la especificación de control que se ejecuta si se cumple la condición.

$$NCS = \{ NCS_i \mid NCS_i \cong (CND_{NCS}(NCS_i), CN_{NCS}(NCS_i)) \}$$

- **CND_{NCS}** (**Condición** de nodo) es una función entre el universo de nodos de condición simple y el universo de condiciones lógicas:

$$CND_{NCS}: NCS \rightarrow CND$$

- **CN_{NCS}** (**Control** de nodo) es una función entre el universo de nodos de condición simple y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse si se cumple la condición:

$$CN_{NCS}: NCS \rightarrow CN$$

Obsérvese cómo un nodo de condición simple contiene una secuencia completa de control. Esto mismo ocurrirá en más nodos de control.

- El **universo de nodos de condición doble** (NCD), contiene elementos que se especifican mediante una terna de funciones: la condición que debe evaluarse, la especificación de control que se ejecuta si se cumple la condición y la especificación de control que se ejecuta en caso contrario.

  $NCD = \{ NCD_i \mid NCD_i \cong (CND_{NCD}(NCD_i), CNV_{NCD}(NCD_i), CNF_{NCD}(NCD_i)) \}$

  - **CND_{NCD}** (**Condición** de nodo) es una función entre el universo de nodos de condición doble y el universo de condiciones lógicas:

    $$CND_{NCD}: NCD \rightarrow CND$$

  - **CNV_{NCD}** (**Control si verdadero** de nodo) es una función entre el universo de nodos de condición doble y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse si se cumple la condición:

    $$CNV_{NCD}: NCD \rightarrow CN$$

  - **CNF_{NCD}** (**Control si falso** de nodo) es una función entre el universo de nodos de condición doble y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse si no se cumple la condición:

    $$CNF_{NCD}: NCD \rightarrow CN$$

- El **universo de nodos de condición múltiple** (NCM), contiene elementos que se especifican mediante una terna de funciones: la expresión que debe evaluarse, las especificaciones de control asignadas a valores concretos de la expresión y la especificación de control que se ejecuta cuando la expresión devuelve un valor no considerado.

  $NCM = \{ NCM_i \mid NCM_i \cong (EXP_{NCM}(NCM_i), CNS_{NCM}(NCM_i), CNF_{NCM}(NCM_i)) \}$

  - **EXP_{NCM}** (**Expresión** de nodo) es una función entre el universo de nodos de condición múltiple y el universo de expresiones de valor:

    $$EXP_{NCM}: NCM \rightarrow EXPV$$

  - **CNS_{NCM}** (**Controles asignados a valores** de nodo) es una función entre el universo de nodos de condición múltiple y las partes del producto cartesiano del universo de valores y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse para determinados valores de la expresión evaluada:

    $$CNS_{NCM}: NCM \rightarrow \Pi(V \times CN)$$

  - **CNF_{NCM}** (**Control si falso** de nodo) es una función entre el universo de nodos de condición múltiple y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse si el valor de la expresión no es ninguno de los considerados en CNS_{NCM}:

    $$CNF_{NCM}: NCM \rightarrow CN$$

- El **universo de nodos de bucle mientras** (NBM), contiene elementos que se especifican mediante un par de funciones: la condición que debe evaluarse y la especificación de control que se ejecuta mientras se cumpla la condición.

$$NBM = \{ NBM_i \mid NBM_i \cong (CND_{NBM}(NBM_i), CN_{NBM}(NBM_i)) \}$$

  - **CND_{NBM}** (**Condición** de nodo) es una función entre el universo de nodos de bucle mientras y el universo de condiciones lógicas:

$$CND_{NBM}: NBM \rightarrow CND$$

  - **CN_{NBM}** (**Control** de nodo) es una función entre el universo de nodos de bucle mientras y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse mientras se cumpla la condición:

$$CN_{NBM}: NBM \rightarrow CN$$

- El **universo de nodos de bucle repetir** (NBR), contiene elementos que se especifican mediante un par de funciones: la condición que debe evaluarse y la especificación de control que se ejecuta hasta que se cumpla la condición.

$$NBR = \{ NBR_i \mid NBR_i \cong (CND_{NBR}(NBR_i), CN_{NBR}(NBR_i)) \}$$

  - **CND_{NBR}** (**Condición** de nodo) es una función entre el universo de nodos de bucle repetir y el universo de condiciones lógicas:

$$CND_{NBR}: NBR \rightarrow CND$$

  - **CN_{NBR}** (**Control** de nodo) es una función entre el universo de nodos de bucle repetir y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse hasta que se cumpla la condición:

$$CN_{NBR}: NBR \rightarrow CN$$

- El **universo de nodos de bucle para** (NBF), contiene elementos que se especifican mediante una quíntupla de funciones: la variable del bucle, el valor inicial, el valor final, el incremento y la especificación de control que se ejecuta para todos los valores de la variable entre el valor inicial y el final, incrementándose en función del incremento correspondiente.

$$NBF = \{ NBF_i \mid NBF_i \cong (VAR_{NBF}(NBF_i), LIN_{NBF}(NBF_i), LSU_{NBF}(NBF_i), INC_{NBF}(NBF_i), CN_{NBF}(NBF_i),) \}$$

  - **VAR_{NBF}** (**Variable** de nodo) es una función entre el universo de nodos de bucle para y el universo de declaraciones de variable:

$$VAR_{NBF}: NBF \rightarrow VAR$$

    La variable deberá estar sin cuantificar y deberá representar valores reales (o enteros).

  - **LIN_{NBF}** (**Límite inferior** de nodo) es una función entre el universo de nodos de bucle para y el conjunto de los números reales:

$$LIN_{NBF}: NBF \rightarrow R$$

  - **LSU_{NBF}** (**Límite superior** de nodo) es una función entre el universo de nodos de bucle para y el conjunto de los números reales:

$$LSU_{NBF}: NBF \rightarrow R$$

- **INC_{NBF}** (**Incremento** de nodo) es una función entre el universo de nodos de bucle para y el conjunto de los números reales:

$$INC_{NBF}: NBF \rightarrow R$$

- **CN_{NBF}** (**Control** de nodo) es una función entre el universo de nodos de bucle para y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse para los valores especificados de la variable del bucle:

$$CN_{NBF}: NBF \rightarrow CN$$

- El **universo de nodos de bucle para todo** (NBFA), contiene elementos que se especifican mediante un par de funciones: la variable del bucle y la especificación de control que se ejecuta para todos los valores posibles de la variable.

$$NBFA = \{ NBFA_i \mid NBFA_i \cong (VAR_{NBFA}(NBFA_i), CN_{NBFA}(NBFA_i),) \}$$

- **VAR_{NBFA}** (**Variable** de nodo) es una función entre el universo de nodos de bucle para todo y el universo de declaraciones de variable:

$$VAR_{NBFA}: NBFA \rightarrow VAR$$

La variable deberá estar sin cuantificar.

- **CN_{NBFA}** (**Control** de nodo) es una función entre el universo de nodos de bucle para y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse para todos los valores posibles de la variable del bucle:

$$CN_{NBFA}: NBFA \rightarrow CN$$

- El **universo de nodos de control paralelo** (NP), contiene elementos que se especifican mediante un par de funciones: las especificaciones de control que deben ejecutarse en paralelo y la forma de terminar la ejecución de caminos paralelos.

$$NP = \{ NP_i \mid NP_i \cong (CNS_{NP}(NP_i), PAR_{NP}(NP_i)) \}$$

- **CNS_{NP}** (**Controles paralelos** de nodo) es una función entre el universo de nodos de condición múltiple y las partes del universo de valores y el universo de especificaciones de control de método, que representa las secuencia de control que deberán ejecutarse en paralelo:

$$CNS_{NP}: NP \rightarrow \Pi(CN)$$

- **PAR_{NP}** (**Parada** de nodo) es una función entre el universo de nodos de condición múltiple y el universo paradas de ejecución paralela, que representa la forma de terminar la ejecución de los caminos paralelos.

$$PAS_{NP}: NP \rightarrow PAR$$

PAR = {“termina uno”, “terminan todos”, “fin”}

  - “termina uno”: la ejecución de caminos paralelos se interrumpe cuando termina uno de los caminos.

- "terminan todos": la ejecución de caminos paralelos sigue hasta que terminan todos los caminos.
- "fin": la ejecución de caminos paralelos únicamente termina cuando uno de los caminos ejecuta FIN.

- El **universo de nodos de espera a evento** (NEV), contiene elementos que se especifican mediante un par de funciones: la condición que refleja el evento esperado y la especificación de control que se ejecuta cuando se cumpla la condición (es decir, cuando se lance el evento).

$$NEV = \{ NEV_i \mid NEV_i \cong (CND_{NEV}(NEV_i), CN_{NEV}(NEV_i)) \}$$

  - **CND_{NEV}** (**Condición** de nodo) es una función entre el universo de nodos de evento y el universo de condiciones lógicas:

$$CND_{NEV}: NEV \rightarrow CND$$

  - **CN_{NEV}** (**Control** de nodo) es una función entre el universo de nodos de evento y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse cuando que se cumpla la condición:

$$CN_{NEV}: NEV \rightarrow CN$$

- El **universo de nodos de control heurístico** (NH), contiene elementos que se especifican mediante una terna de funciones: una resolución de tarea que representa la evaluación de la situación actual, una expresión sobre los resultados de esa tarea, las secuencias de control asignadas a distintos valores de la expresión y la especificación de control que se ejecuta cuando no se ha producido ninguno de los valores esperados.

$$NH = \{ NH_i \mid NH_i \cong (NT_{NH}(NH_i), EXP_{NH}(NH_i), CNS_{NH}(NH_i), CNF_{NH}(NH_i)) \}$$

  - **NT_{NH}** (**Nodo de resolución de tarea** de nodo) es una función entre el universo de nodos de control heurístico y el universo de nodos de resolución de tareas:

$$NT_{NH}: NH \rightarrow NT$$

  - **EXP_{NH}** (**Expresión** de nodo) es una función entre el universo de nodos de control heurístico y el universo de expresiones de valor:

$$EXP_{NH}: NH \rightarrow EXPV$$

  - **CNS_{NH}** (**Controles asignados a valores** de nodo) es una función entre el universo de nodos de control heurístico y las partes del producto cartesiano del universo de valores y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse para determinados valores de la expresión evaluada:

$$CNS_{NH}: NH \rightarrow \Pi(V \times CN)$$

  - **CNF_{NH}**(**Control si falso** de nodo) es una función entre el universo de nodos de control heurístico y el universo de especificaciones de control de método, que representa la secuencia de control que deberá ejecutarse si el valor de la expresión no es ninguno de los considerados en CNS_{NH}:

$$CNF_{NH}: NH \rightarrow CN$$

4.2.2.2.3 Semántica del control

Seguidamente se recoge la semántica de cada uno de los nodos de control:

- Fin de una secuencia de control (NF): termina por completo la ejecución de la secuencia de control, independientemente del nivel de profundidad en el que se encuentre.
- Declaración de una variable (NV): define una variable que podrá usarse en el resto de nodos de la secuencia.
- Resolución de una tarea (NT): resuelve una tarea, produciéndose el cambio de estado correspondiente si se verifica la precondición de la tarea (en cuyo caso el tiempo actual se incrementa).
- Ejecución de un operador (NO): resuelve una tarea, produciéndose el cambio elemental de estado correspondiente (el tiempo actual se incrementa).
- Condición simple (*if – then*) (NCS): si se cumple la condición se ejecuta la secuencia de control especificada.
- Condición doble (*if – then - else*) (NCD): si se cumple la condición se ejecuta la secuencia de control "si verdadero". En otro caso se ejecuta la secuencia de control "si falso".
- Condición múltiple (*case*) (NCM): se evalúa la expresión y se ejecuta la secuencia de control correspondiente al valor de la expresión (o la secuencia de control "si falso" cuando el valor no fue considerado).
- Bucle mientras (*while*) (NBM): se ejecuta la secuencia de control correspondiente mientras se cumpla la condición.
- Bucle repetir (*repeat*) (NBR): se ejecuta la secuencia de control correspondiente hasta que se cumpla la condición.
- Bucle para (*for*) (NBF): se ejecuta la secuencia de control correspondiente para los valores de la variable entre el límite inferior y el superior.
- Bucle para todo (*for all*) (NBFA): se ejecuta la secuencia de control correspondiente para todos los valores posibles de la variable.
- División en caminos paralelos (NP): se ejecutan en paralelo todas las secuencia de control. La forma de terminar dependerá del parámetro PAR_{NP}.
- Espera a evento (cambio de estado o condición sobre el tiempo): el control se bloquea hasta que ocurre el evento, momento en el que se ejecuta la secuencia de control correspondiente.
- Control heurístico: se resuelve la tarea, se evalúa la expresión y se ejecuta la secuencia de control correspondiente al valor de la expresión (o la secuencia de control "si falso" cuando el valor no fue considerado).

4.2.2.3 Formalización de Operadores

Los operadores son tareas predefinidas e independientes del dominio, que realizan cambios elementales de estado y que se pueden utilizar en cualquier problema.

Dado que son independientes del dominio, tienen parámetros "plantilla", a los que se podrán asignar elementos dependientes del dominio.

Por ejemplo, el operador "Crear instancia de concepto" tiene como segunda entrada una "plantilla", a la que se puede asignar cualquier concepto de cualquier modelo estructural.

4.2.2.4 Formalización de Funciones de Consulta

Las funciones de consulta permiten obtener vistas parciales del estado del modelo estructural en un instante determinado. Las funciones de consulta están predefinidas y son independientes de dominio.

El resultado de las funciones de consulta puede utilizarse dentro de expresiones, o bien asignarse a variables.

Dado que son independientes del dominio, tienen parámetros "plantilla", a los que se podrán asignar elementos dependientes del dominio.

4.2.3 Formalización de las Condiciones Lógicas

La formalización de las condiciones lógicas y su evaluación se va a dividir en seis apartados. En primer lugar se formalizarán en detalle las condiciones lógicas unarias. y luego las binarias. A continuación se formalizará la evaluación de las condiciones unarias y binarias. Para terminar se formalizarán las restricciones de las invariantes de concepto y asociación.

4.2.3.1 Formalización de condiciones lógicas unarias

Esta formalización se va a dividir en cinco partes, para abarcar los cinco elementos de las condiciones lógicas unarias: condiciones, variables, fórmulas, átomos y expresiones.

4.2.3.1.1 Condiciones

CND es el **universo de las condiciones lógicas** (expresiones lógicas con variables). Estas condiciones lógicas pueden ser de 6 tipos que se irán definiendo a continuación.

$$CND = CNDV \cup CNDC \cup CNDD \cup CNDI \cup CNDN \cup CNDF$$

CNDV es el conjunto de **condiciones lógicas con variable**. Cada elemento de este conjunto se especifica mediante una variable y la condición lógica a la que afecta:

$$CNDV = \{CNDV_i \mid CNDV_i \cong (VAR_{CV}(CNDV_i), CND_{CV}(CNDV_i))\}$$

- **VAR_{CV}** (**Variable** de la condición). Es una aplicación entre el conjunto de las condiciones lógicas con variable (*CNDV*) y el conjunto de las definiciones de variables (*VAR*), de forma que a cada condición lógica con variable se le asigna una definición de variable (éstas se formalizarán en el apartado siguiente).

$$VAR_{CV}: CNDV \rightarrow VAR$$

- **CND_{CV}** (**Condición** de la condición con variable). Es una aplicación entre el conjunto de las condiciones lógicas con variable (*CNDV*) y el conjunto de las condiciones lógicas de cualquier tipo (*CND*), de forma que a cada condición lógica con variable se le asigna una condición que se verá afectada por la variable.

$$CND_{CV}: CNDV \rightarrow CND$$

CNDC es el conjunto de **condiciones lógicas de conjunción**. Cada elemento de este conjunto representa la conjunción lógica (AND) de dos condiciones lógicas de cualquier tipo. Cada condición lógica de conjunción se define mediante las dos condiciones lógicas cuya conjunción representa:

$$CNDC = \{CNDC_i \mid CNDC_i \cong CNDS_{CC}(CNDC_i)\}$$

- **CNDS$_{CC}$** (**Condiciones** de la condición de conjunción). Es una aplicación entre el conjunto de las condiciones lógicas de conjunción (*CNDC*) y el producto cartesiano del conjunto de las condiciones lógicas (de cualquier tipo) consigo mismo (*CND*×*CND*), de forma que a cada condición lógica de conjunción se le asignan dos condiciones.

$$CNDS_{CC}: CNDC \rightarrow CND \times CND$$

CNDD es el conjunto de **condiciones lógicas de disyunción**. Cada elemento de este conjunto representa la disyunción lógica (OR) de dos condiciones lógicas de cualquier tipo. Cada condición lógica de disyunción se define mediante las dos condiciones lógicas cuya disyunción representa:

$$CNDD = \{CNDD_i \mid CNDD_i \cong CNDS_{CD}(CNDD_i)\}$$

- **CNDS$_{CD}$** (**Condiciones** de la condición de disyunción). Es una aplicación entre el conjunto de las condiciones lógicas de disyunción (*CNDD*) y el producto cartesiano del conjunto de las condiciones lógicas (de cualquier tipo) consigo mismo (*CND*×*CND*), de forma que a cada condición lógica de conjunción se le asignan dos condiciones.

$$CNDS_{CD}: CNDD \rightarrow CND \times CND$$

CNDI es el conjunto de **condiciones lógicas de implicación**. Cada elemento de este conjunto representa la implicación lógica (→) de dos condiciones lógicas de cualquier tipo. Cada condición lógica de implicación se define mediante las dos condiciones lógicas cuya implicación representa:

$$CNDI = \{CNDI_i \mid CNDI_i \cong CNDS_{CI}(CNDI_i)\}$$

- **CNDS$_{CI}$** (**Condiciones** de la condición de implicación). Es una aplicación entre el conjunto de las condiciones lógicas de implicación (*CNDI*) y el producto cartesiano del conjunto de las condiciones lógicas (de cualquier tipo) consigo mismo (*CND*×*CND*), de forma que a cada condición lógica de implicación se le asignan dos condiciones.

$$CNDS_{CI}: CNDI \rightarrow CND \times CND$$

CNDN es el conjunto de **condiciones lógicas de negación**. Cada elemento de este conjunto representa la negación lógica (NOT) de una condición lógica de cualquier tipo. Cada condición lógica de negación se define mediante la condición lógica cuya negación representa:

$$CNDN = \{CNDN_i \mid CNDN_i \cong CND_{CN}(CNDN_i)\}$$

- **CND_{CN}** (**Condición** de la condición de negación). Es una aplicación entre el conjunto de las condiciones lógicas de negación (*CNDN*) y el conjunto de las condiciones lógicas de cualquier tipo (*CND*), de forma que a cada condición lógica de negación se le asigna una condición.

$$CND_{CN}: CNDN \rightarrow CND$$

CNDF es el conjunto de **condiciones lógicas de fórmula**. Cada elemento de este conjunto representa una fórmula (condición lógica sin definición de variables). Cada condición lógica de fórmula se define mediante la fórmula que representa:

$$CNDF = \{CNDF_i \mid CNDF_i \cong FORM_{CF}(CNDF_i)\}$$

Una condición lógica de fórmula se define, por tanto, con un único elemento:

- **$FORM_{CF}$** (**Fórmula** de la condición de fórmula). Es una aplicación entre el conjunto de las condiciones lógicas de fórmula (*CNDF*) y el conjunto de las fórmulas (*FORM*), de forma que a cada condición lógica de fórmula se le asigna una fórmula.

$$FORM_{CF}: CNDF \rightarrow FORM$$

Equipo de fútbol

Seguidamente se muestra un ejemplo de condición completa, cuyos elementos irán mostrándose en los apartados siguientes. El ejemplo es la invariante del concepto Fecha: para toda fecha debe cumplirse:

1. Si el mes es abril, junio, septiembre o noviembre, el día debe ser menor que 31.
2. Si el mes es febrero y el año es bisiesto (múltiplo de 4 pero no múltiplo de 100 excepto los múltiplos de 400), el día debe ser menor que 30.
3. Si el mes es febrero y el año no es bisiesto, el día debe ser menor que 29.

En cualquier otro mes el día siempre será válido, dado que el tipo día sólo admite valores entre 1 y 31.

La formulación lógica de esta condición es la siguiente:

"Para toda fecha "f", la existencia de un día "d", un mes "m" y un año "a" tales que "d" es el día de "f", "m" es el mes de "f" y "a" es el año de "f" implica que:

- "m" es abril (4), junio(6), septiembre (9) o noviembre (11) y el día es menor que 31 o bien
- "m" es febrero, "a" es múltiplo de 4 y no es múltiplo de 100 o lo es de 400 y el día es menor que 30 o bien
- "m" es febrero, "a" no es múltiplo de 4 o es múltiplo de 100 y no lo es de 400 y el día es menor que 29"

```
CNDV(VAR("f", ∀, "Fecha"),
  CNDV(VAR("d", ∃, "Día"),
    CNDV(VAR("m", ∃, "Mes"),
      CNDV(VAR("a", ∃, "Año"),
        CNDF(FORI(
          FORC(
            FORA(APAT(VAR("f"), EXZV(VAR("d")), "día")),
            FORC(
              FORA(APAT(VAR("f"), EXZV(VAR("m")), "mes")),
              FORA(APAT(VAR("f"), EXZV(VAR("a")), "año"))
```

```
            )
          ),
          FORD(
            FORC(
              FORD(
                FORA(ACV(=,(EXZV(VAR("m")),EXNV(4)))),
                FORD(
                  FORA(ACV(=,(EXZV(VAR("m")),EXNV(6)))),
                  FORD(
                    FORA(ACV(=,(EXZV(VAR("m")),EXNV(9)))),
                    FORA(ACV(=,(EXZV(VAR("m")),EXNV(11))))
                  )
                )
              ),
              FORA(ACV(<,(EXZV(VAR("d")),EXNV(31))))
            ),
            FORD(
              FORC(
                FORC(
                  FORA(ACV(=,(EXZV(VAR("m")),EXNV(2)))),
                  FORC(
                    FORA(ACV(=,(EXZO(%,(VAR("m"),EXNV(4))),EXNV(0)))),
                    FORD(
                      FORA(ACV(≠,(EXZO(%,(VAR("m"),EXZV(100))),
                                  EXZV(0)))),
                      FORA(ACV(=,(EXZO(%,(VAR("m"),EXZV(400))),EXZV(0))))
                    )
                  )
                ),
                FORA(ACV(<,(EXZV(VAR("d")),EXZV(30))))
              ),
              FORC(
                FORC(
                  FORA(ACV(=,(EXZV(VAR("m")),EXZV(2)))),
                  FORD(
                    FORA(ACV(≠,(EXZO(%,(VAR("m"),EXZV(4))),EXZV(0)))),
                    FORC(
                      FORA(ACV(=,(EXZO(%,(VAR("m"),EXZV(100))),
                                  EXZV(0)))),
                      FORA(ACV(≠,(EXZO(%,(VAR("m"),EXZV(400))),EXZV(0))))
                    )
                  )
                ),
                FORA(ACV(<,(EXNV(VAR("d")),EXNV(29))))
              )
            )
          )
        )
      )
    )
  )
)
```

Ésta es una condición compleja en la que aparecen:

- 4 condiciones con definición de variable (CNDV) y 1 condición de fórmula (CNDF).
- 1 fórmula de implicación (FORI) que contiene 9 fórmulas de conjunción (FORC), 7 fórmulas de disyunción (FORD) y 18 fórmulas de átomo (FORA).
- 3 átomos de pertenencia de par (instancia, valor) a atributo de concepto (APAT) y 15 átomos de comparación de valores (ACV).
- 31 expresiones de valor entero (EXZV) y 5 expresiones de operación binaria entera (EXZO).

Una vez formalizados todos los tipos de condiciones lógicas, es conveniente definir una función que permita conocer el **conjunto de variables de una condición lógica** (***VARS$_{CND}$***). Es una aplicación entre el conjunto de condiciones lógicas de cualquier tipo (*CND*) y las partes del conjunto de variables (*ΠI(VAR)*), de forma que, dada una condición lógica, devuelve el conjunto de variables de dicha condición.

$$VARS_{CND}: CND \rightarrow \Pi(VAR)$$

El cálculo de la función $VARS_{CND}$ depende del tipo de condición:

- Si se trata de una condición de variable (CNDV), entonces su conjunto de variables será la variable definida, más las variables de su condición asociada.

$$\forall CND_i \in CNDV \bullet VARS_{CND}(CND_i) = VAR_{CV}(CND_i) \cup VARS_{CND}(CND_{CV}(CND_i))$$

- Si se trata de una condición de conjunción (CNDC), entonces su conjunto de variables será la unión de los conjuntos de variables de las condiciones asociadas.

$$\forall CND_i \in CNDC \bullet sea\ (CND_j, CND_k) = CNDS_{CC}(CND_i) \bullet VARS_{CND}(CND_i) = VARS_{CND}(CND_j) \cup VARS_{CND}(CND_k)$$

- Si se trata de una condición de disyunción (CNDD), su conjunto de variables será la unión de los conjuntos de variables de las condiciones asignadas.

$$\forall CND_i \in CNDD \bullet sea\ (CND_j, CND_k) = CNDS_{CD}(CND_i) \bullet VARS_{CND}(CND_i) = VARS_{CND}(CND_j) \cup VARS_{CND}(CND_k)$$

- Si se trata de una condición de implicación (CNDI), su conjunto de variables será la unión de los conjuntos de variables de las condiciones asignadas.

$$\forall CND_i \in CNDI \bullet sea\ (CND_j, CND_k) = CNDS_{CI}(CND_i) \bullet VARS_{CND}(CND_i) = VARS_{CND}(CND_j) \cup VARS_{CND}(CND_k)$$

- Si se trata de una condición de negación (CNDN), su conjunto de variables será el conjunto de variables de la condición negada.

$$\forall CND_i \in CNDN \bullet VARS_{CND}(CND_i) = VARS_{CND}(CND_{CN}(CND_i))$$

- Si se trata de una condición de fórmula (CNDF), su conjunto de variables será el conjunto vacío.

$$\forall CND_i \in CNDF \bullet VARS_{CND}(CND_i) = \varnothing$$

4.2.3.1.2 Variables

VAR es el universo de **definiciones de variables** de condiciones lógicas. Cada variable se define mediante un nombre, un cuantificador, un conjunto del que toma valores y un tipo de variable:

$$VAR = \{VAR_i \mid VAR_i = (N_{VAR}(VAR_i), CUA_{VAR}(VAR_i), CJTO_{VAR}(VAR_i), TP_{VAR}(VAR_i))\ \}$$

- **N$_{VAR}$** (**Nombre** de variable) es una aplicación entre el conjunto de variables (*VAR*) y el conjunto de cadenas de caracteres (*CAD*), de forma que a cada variable se le asocia un nombre:

$$N_{VAR}: VAR \rightarrow CAD$$

  Dentro de una misma condición, el nombre de una variable debe ser único (dos variables diferentes deben tener nombres distintos).

$$\forall CND_i \in CND \bullet \forall VAR_j, VAR_k \in VARS_{CND}(CND_i) \bullet VAR_j \neq VAR_k \Rightarrow N_{VAR}(VAR_j)) \neq N_{VAR}(VAR_k))$$

- **CUA$_{VAR}$** (**Cuantificador** de variable) es una aplicación entre el conjunto de definiciones de variables (*VAR*) y la unión del conjunto de cuantificadores con el conjunto cuyo único elemento es el conjunto vacío ($CUA \cup \{\varnothing\}$), de forma que a cada variable se le puede asociar un cuantificador o ninguno:

  $$CUA_{VAR}: VAR \rightarrow CUA \cup \{\varnothing\}$$

  CUA es el conjunto formado por tres valores: $CUA = \{\forall, \exists, \exists_1\}$, donde "$\forall$" es el cuantificador universal ("para todo"), "$\exists$" es el cuantificador existencial ("existe") y "$\exists_1$" es el cuantificador de existencia única ("existe único").

  Dada esta definición podrán existir dos tipos de variables respecto a su cuantificador: *variables cuantificadas* (las que tienen asociado un cuantificador) y *variables libres* (las que no tienen cuantificador).

- **CJTO$_{VAR}$** (**Conjunto** de variable) es una aplicación entre el conjunto de definiciones de variables (*VAR*) y la unión entre el universo de los tipos (*T*), el universo de los conceptos (*C*) y el universo de las asociaciones (*AS*). A cada variable se le asignará un tipo, concepto o asociación que será el conjunto del que pueda tomar valores.

  $$CJTO_{VAR}: VAR \rightarrow T \cup C \cup AS$$

  Dada esta definición podrán existir tres tipos de variables respecto a su conjunto de valores: variables de tipo (cuyos valores serán elementos del tipo asignado), variables de concepto (cuyos valores podrán ser instancias del concepto asignado) y variables de asociación (cuyos valores serán elementos de la asociación correspondiente).

- **TP$_{VAR}$** (**Tipo** de variable) es una aplicación entre el conjunto de definiciones de variables (*VAR*) y el conjunto de tipos de variable (*TPV*) de forma que a cada variable se le asocia un tipo que dice si representa un valor simple o un conjunto de valores:

  $$TP_{VAR}: VAR \rightarrow TPV$$

  TPV es un conjunto formado por dos valores: *TPV = {"simple", "conjunto"}*.

  Si TP_{VAR} es "simple", entonces la variable tendrá como valores elementos de su conjunto. Si TP_{VAR} es "conjunto", entonces la variable tendrá como valores conjuntos de elementos de su conjunto de referencia.

  Dada esta definición podrán existir dos tipos de variables respecto cómo son sus valores: *variables simples* (las que representan valores) y *variables de conjunto* (las que representan variables de conjunto).

Equipo de fútbol

Algunos ejemplos de definición de variables:

- Una variable "p" que representa una persona cuantificada universalmente:

```
VAR("p", ∀, C"Persona")
```

- Una variable "país" que representa un valor de tipo "País" cuantificado existencialmente:

```
VAR("país", ∃, T"País")
```

- Una variable "e" que representa a un equipo sin cuantificar:

```
VAR("e",∅, C"Equipo")
```

Conviene indicar que si se define una variable simple de asociación, automáticamente se consideran definidas variables para cada uno de los roles de la asociación. Cada una de esas variables será de tipo, concepto o asociación según corresponda, tendrá el mismo cuantificador que la variable de la que se deriva y tendrá un nombre formado por el nombre de la variable, más el carácter punto '.' más el nombre del rol.

Equipo de fútbol

Si se define la siguiente variable que representa una tupla de la asociación "Pertenencia":

```
VAR("p",∅, AS"(Persona, Fecha).Pertenencia")
```

Entonces se consideran definidas automáticamente las siguientes variables:

```
VAR("p.persona",∅, C"Persona")
VAR("p.fecha",∅, C"Fecha")
VAR("p.equipo",∅, C"Equipo")
VAR("país", ∃, T"País")
```

4.2.3.1.3 Fórmulas

FORM es el **universo de las fórmulas lógicas** (expresiones lógicas sin definición de variables). Las fórmulas pueden ser de 5 tipos.

$$FORM = FORC \cup FORD \cup FORI \cup FORN \cup FORA$$

FORC es el conjunto de **fórmulas de conjunción**. Cada elemento de este conjunto representa la conjunción lógica (AND) de dos fórmulas de cualquier tipo. Cada fórmula de conjunción se define mediante las dos fórmulas cuya conjunción representa:

$$FORC = \{FORC_i \mid FORC_i \cong FORS_{FC}(FORC_i)\}$$

- **$FORS_{FC}$** (**Fórmulas** de la fórmula de conjunción). Es una aplicación entre el conjunto de las fórmulas de conjunción (*FORC*) y el producto cartesiano del conjunto de fórmulas (de cualquier tipo) consigo mismo ($FORM \times FORM$), de forma que a cada fórmula de conjunción se le asignan dos fórmulas.

$$FORS_{FC}: FORC \rightarrow FORM \times FORM$$

Equipo de fútbol

Ejemplo de fórmula de conjunción de dos fórmulas de tipo átomo: "Casillas" (instancia de Jugador) es portero y es jugador nacional (APC es un átomo que representa la pertenencia de una instancia a un concepto).

```
FORC(FORA(APC("Casillas","Portero")), FORA(APC("Casillas","Nacional")))
```

FORD es el conjunto de **fórmulas de disyunción**. Cada elemento de este conjunto representa la disyunción lógica (OR) de dos fórmulas de cualquier tipo. Cada fórmula de disyunción se define mediante las dos fórmulas cuya disyunción representa:

$$FORD = \{FORD_i \mid FORD_i \cong FORS_{FD}(FORD_i)\}$$

- **$FORS_{FD}$** (**Fórmulas** de la fórmula de disyunción). Es una aplicación entre el conjunto de las fórmulas de disyunción (*FORD*) y el producto cartesiano del conjunto de fórmulas (de cualquier tipo) consigo mismo (*FORM×FORM*), de forma que a cada fórmula de disyunción se le asignan dos fórmulas.

$$FORS_{FD}: FORD \rightarrow FORM \times FORM$$

Equipo de fútbol

Ejemplo de fórmula de disyunción de dos fórmulas de tipo átomo: una persona (variable "p" puede ser jugador o directivo).

```
FORD(FORA(APC(VAR("p"),"Jugador")), FORA(APC(VAR("p"),"Directivo")))
```

FORI es el conjunto de **fórmulas de implicación**. Cada elemento de este conjunto representa la implicación lógica (→) de dos fórmulas de cualquier tipo. Cada fórmula de implicación se define mediante las dos fórmulas cuya implicación representa:

$$FORI = \{FORI_i \mid FORI_i \cong FORS_{FI}(FORI_i)\}$$

- **$FORS_{FI}$** (**Fórmulas** de la fórmula de implicación). Es una aplicación entre el conjunto de las fórmulas de implicación (*FORI*) y el producto cartesiano del conjunto de fórmulas (de cualquier tipo) consigo mismo (*FORM×FORM*), de forma que a cada fórmula de implicación se le asignan dos fórmulas.

$$FORS_{FI}: FORI \rightarrow FORM \times FORM$$

Equipo de fútbol

Ejemplo de fórmula de implicación de dos fórmulas de tipo átomo: el que "Raúl" sea un jugador nacional implica que "España" es el valor de su atributo "país" (APAT es un átomo que representa la pertenencia de un par (instancia, valor) a un atributo de concepto)

```
FORI(FORA(APC("Raúl","Nacional")), FORA(APAT("Raúl", EXCV("España"),
"país")))
```

FORN es el conjunto de **fórmulas de negación**. Cada elemento de este conjunto representa la negación lógica (NOT) de una fórmula de cualquier tipo. Cada fórmula de negación se define mediante la fórmula cuya negación representa:

$$FORN = \{FORN_i \mid FORN_i \cong FORM_{FN}(FORN_i)\}$$

- **$FORM_{FN}$** (**Fórmula** de la fórmula de negación). Es una aplicación entre el conjunto de las fórmulas de negación (*FORN*) y el conjunto de fórmulas de cualquier tipo (*FORM*), de forma que a cada fórmula de negación se le asigna una fórmula.

$$FORM_{FN}: FORN \rightarrow FORM$$

Equipo de fútbol

Ejemplo de fórmula de negación de una fórmula de tipo átomo: "Rivaldo" no es jugador comunitario.

```
FORN(FORA(APC("Rivaldo","Comunitario")))
```

FORA es el conjunto de **fórmulas de átomo**. Cada elemento de este conjunto representa un átomo (condición elemental). Cada fórmula de átomo se define mediante su átomo:

$$FORA = \{FORA_i \mid FORA_i \cong ATOM_{FA}(FORA_i)\}$$

- **$ATOM_{FA}$** (**Átomo** de la fórmula de átomo). Es una aplicación entre el conjunto de las fórmulas de átomo (*FORA*) y el conjunto de átomos de cualquier tipo (*ATOM*), de forma que a cada fórmula de átomo se le asigna un átomo.

$$ATOM_{FA}: FORA \rightarrow ATOM$$

Equipo de fútbol

En los ejemplos anteriores han aparecido ejemplos de fórmula de tipo átomo:

- La instancia "Casillas" pertenece al concepto "Portero"

```
FORA(APC("Casillas","Portero"))
```

- La instancia "Casillas" pertenece al concepto "Nacional"

```
FORA(APC("Casillas","Nacional"))
```

- La variable "p" pertenece al concepto "Jugador"

```
FORA(APC(VAR("p"),"Jugador"))
```

- La variable "p" pertenece al concepto "Directivo"

```
FORA(APC(VAR("p"),"Directivo"))
```

- "Raúl" es un jugador nacional

```
FORA(APC("Raúl","Nacional"))
```

- El valor "España" pertenece al atributo "país" de la instancia "Raúl"

```
FORA(APAT("Raúl", EXCV("España"), "país"))
```

- Rivaldo es jugador comunitario

```
FORA(APC("Rivaldo","Comunitario"))
```

4.2.3.1.4 Átomos

ATOM es el **universo de los átomos**. Los átomos pueden ser de 8 tipos.

$$ATOM = APC \cup APAT \cup APAS \cup APATS \cup APT \cup ACV \cup ACIC \cup ACIAS$$

En estos átomos, los valores de tipo se representarán mediante expresiones que devuelven valores (en las que pueden aparecer variables de tipo), las instancias de concepto podrán representarse directamente o bien usando variables de concepto y, por último, las instancias de asociación podrán representarse directamente o mediante variables de asociación.

APC es el conjunto de **átomos de pertenencia de instancia a concepto**. Cada elemento APC_i de este conjunto representa la condición de pertenencia de una instancia de concepto a un concepto concreto. Cada APC_i se define mediante una instancia (que puede ser una variable) y un concepto:

$$APC = \{APC_i \mid APC_i \cong (I_{APC}(APC_i), C_{APC}(APC_i))\}$$

- **I_{APC}** (**Instancia** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de instancia a concepto (*APC*) y la unión entre el conjunto de instancias de conceptos y el conjunto de definiciones de variables ($I \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o bien una variable.

$$I_{APC}: APC \rightarrow I \cup VAR$$

Si la instancia de un átomo de pertenencia de instancia a concepto es una variable, deberá ser una variable simple de concepto (es decir, su conjunto deberá ser un concepto):

$$\forall APC_i \in APC \bullet I_{APC}(APC_i) \in VAR \Rightarrow CJTO_{VAR}(I_{APC}(APC_i)) \in C \wedge TP_{VAR}(I_{APC}(APC_i)) = \text{“simple”}$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

- **C_{APC}** (**Concepto** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de instancia a concepto (*APC*) y la unión del conjunto de conceptos con el conjunto de variables ($C \cup VAR$), de forma que a cada átomo se le asocia un concepto concreto o una variable.

$$C_{APC}: APC \rightarrow C \cup VAR$$

Si el concepto de un átomo de pertenencia de instancia a concepto es una variable, deberá ser una variable de conjunto de concepto (es decir, su conjunto deberá ser un concepto):

$$\forall APC_i \in APC \bullet C_{APC}(APC_i) \in VAR \Rightarrow CJTO_{VAR}(C_{APC}(APC_i)) \in C \wedge TP_{VAR}(C_{APC}(APC_i)) = \text{“conjunto”}$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

Equipo de fútbol

Ejemplos de átomos de pertenencia de instancia a concepto:

- La instancia “Figo” pertenece a “Jugador Campo”

```
APC("Figo","Jugador Campo")
```

- La variable “jug” (Un jugador) pertenece a “Nacional”

```
APC(VAR("jug"),"Nacional")
```

APAT es el conjunto de **átomos de pertenencia de par a atributo**. Cada elemento $APAT_i$ de este conjunto representa la condición de pertenencia de un par (instancia de concepto, valor) a un atributo concreto. Cada $APAT_i$ se define mediante una instancia de concepto (que puede ser una variable), un valor y un atributo:

$$APAT = \{APAT_i \mid APAT_i \cong (I_{APAT}(APAT_i), V_{APAT}(APAT_i), AT_{APAT}(APAT_i))\}$$

- **I_{APAT}** (**Instancia** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo (*APAT*) y la unión entre el conjunto de instancias de conceptos y el conjunto de definiciones de variables ($I \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o bien una variable.

$$I_{APAT}: APAT \rightarrow I \cup VAR$$

Si la instancia de un átomo de pertenencia de par a atributo es una variable, deberá ser una variable simple de concepto (es decir, su conjunto deberá ser un concepto):

$$\forall APAT_i \in APAT \bullet I_{APAT}(APAT_i) \in VAR \Rightarrow CJTO_{VAR}(I_{APAT}(APAT_i)) \in C \wedge TP_{VAR}(I_{APAT}(APAT_i)) = \text{“simple”}$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

- **V_{APAT}** (**Valor** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo (*APAT*) y el conjunto de expresiones que devuelven valores (*EXPV*), de forma que a cada átomo se le asocia una expresión de valor.

$$V_{APAT}: APAT \rightarrow EXPV$$

EXPV es el conjunto de **expresiones de valores**, que será descrito en el apartado siguiente. Estas expresiones incluyen valores concretos, variables de tipo y expresiones enteras, de cadenas o booleanas entre ellos.

- **AT_{APAT}** (**Atributo** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo (*APAT*) y el conjunto de atributos de concepto (*AT*), de forma que a cada átomo se le asocia un atributo concreto.

$$AT_{APAT}: APAT \rightarrow AT$$

Equipo de fútbol

Ejemplos de pertenencia de par a atributo de concepto:

- La valoración de "regate" para "Rivaldo" es 8 (EXRV representa una expresión real con un valor concreto o variable):

```
APAT("Rivaldo",EXRV(8),"regate")
```

- La posición del jugador representado por la variable "jug" es "Delantero" (EXCV representa una expresión de cadena con un valor concreto o variable):

```
APAT(VAR("jug"),EXCV("Delantero"),"posición")
```

- El nombre de "Figo" es el representado por la variable "nom":

```
APAT("Figo", EXCV(VAR"nom")),"nombre")
```

APAS es el conjunto de **átomos de pertenencia de tupla a asociación**. Cada elemento $APAS_i$ de este conjunto representa la condición de pertenencia de una tupla (instancia de asociación) a una asociación concreta. Cada $APAS_i$ se define mediante una tupla (que puede contener variables) y una asociación:

$$APAS = \{APAS_i \mid APAS_i \cong (IAS_{APAS}(APAS_i), AS_{APAS}(APAS_i))\}$$

- **IAS_{APAS}** (**Instancia de asociación** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de tupla a asociación (*APAS*) y la unión del conjunto de instancias de asociación con variables con el conjunto de variables ($IASV \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o una variable que representa una tupla.

$$IAS_{APAS}: APAS \rightarrow IASV \cup VAR$$

Si la instancia de un átomo de pertenencia de tupla a asociación es una variable, deberá ser una variable simple de asociación:

$$\forall APAS_i \in APAS \bullet IAS_{APAS}(APAS_i) \in VAR \Rightarrow CJTO_{VAR}(IAS_{APAS}(APAS_i)) \in AS \wedge TP_{VAR}(IAS_{APAS}(APAS_i)) = \text{“simple”}$$

IASV es el conjunto de **instancias de asociación con variables**. Representa listas en las que cada elemento puede ser una instancia de concepto, una variable de

concepto o bien otra instancia de asociación con variables. Este conjunto se define como la unión de todas las potencias (a partir de dos) del conjunto de instancias genéricas con variables (IGV):

$$IASV = \bigcup_{n=2}^{\infty} IGV^n$$

IGV es el conjunto de **instancias genéricas con variables**. Cada elemento de este conjunto es una instancia de concepto, una variable o bien otra tupla con variables (es decir, un elemento de IASV):

$$IGV = I \cup VAR \cup IASV$$

Si una instancia genérica es una variable, podrá ser una variable simple de concepto o de asociación:

$$\forall IGV_i \in APAT \bullet IGV_i \in VAR \Rightarrow CJTO_{VAR}(IGV_i)) \in C \cup AS \; TP_{VAR}(IGV_i) = \text{“simple”}$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

Equipo de fútbol

Ejemplos de listas de instancias genéricas con variables que representan tuplas de asociaciones:

- Tupla formada por las instancias de concepto “Loïc” (una persona) y “16.10.1969” (una fecha)

```
IASV(“Loïc”, “16.10.1969”)
```

- Tupla formada por dos instancias de concepto (“Rivaldo”, “1.8.1998”) y una variable que representa un club (club):

```
IASV(“Rivaldo”, “1.8.1998”, VAR(“club”))
```

- Tupla formada por una tupla (con tres elementos) y una instancia de concepto:

```
IASV(IASV(“Rivaldo”, “1.8.1998”, VAR(“club”)), “1.8.2002”)
```

- **AS_{APAS} (Asociación** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de tupla a asociación (*APAS*) y la unión el conjunto de asociaciones con el conjunto de variables ($AS \cup VAR$), de forma que a cada átomo se le asocia una asociación concreta o una variable que representa un subconjunto de una asociación.

$$AS_{APAS}: APAS \rightarrow AS \cup VAR$$

Si la asociación de un átomo de pertenencia de tupla a asociación es una variable, deberá ser una variable de conjunto de asociación:

$$\forall APAS_i \in APAS \bullet AS_{APAS}(APAS_i) \in VAR \Rightarrow CJTO_{VAR}(AS_{APAS}(APAS_i)) \in AS \wedge TP_{VAR}(AS_{APAS}(APAS_i)) = \text{“conjunto”}$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

Equipo de fútbol

Ejemplos de átomos de pertenencia de tupla a asociación:

- La tupla formada por “Loïc” y “16.10.1969” pertenece a la asociación “Nacimiento”

```
APAS(IASV("Loïc", "16.10.1969"), "Nacimiento")
```

- La tupla formada por "Rivaldo", "1.8.1998" y la variable "club" pertenece a la asociación "Fichaje":

```
APAS(IASV("Rivaldo", "1.8.1998", VAR("club")), "Fichaje")
```

- Una tupla pertenece a la asociación pertenencia en la que el origen es otra asociación:

```
APAS(IASV(IASV("Rivaldo", "1.8.1998", VAR("club")), "1.8.2002"), "Fin de pertenencia")
```

APATS es el conjunto de **átomos de pertenencia de par a atributo de asociación**. Cada elemento $APATS_i$ de este conjunto representa la condición de pertenencia de un par (instancia de asociación, valor) a un atributo de asociación concreto. Cada $APATS_i$ se define mediante una instancia de asociación (que puede contener variables), un valor y un atributo de asociación:

$$APATS = \{APATS_i \mid APATS_i \cong (IAS_{APATS}(APATS_i), V_{APATS}(APATS_i), ATS_{APATS}(APATS_i))\}$$

- **IAS_{APATS}** (**Instancia de asociación** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo de asociación (*APATS*) y la unión del conjunto de instancias de asociación con variables con el conjunto de variables ($IASV \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o una variable que representa una tupla.

$$IAS_{APATS}: APATS \rightarrow IASV \cup VAR$$

  Si la instancia de un átomo de pertenencia de par a atributo de asociación es una variable, deberá ser una variable simple de asociación:

$$\forall APATS_i \in APATS \bullet IAS_{APATS}(APATS_i) \in VAR \Rightarrow CJTO_{VAR}(IAS_{APATS}(APATS_i)) \in AS \wedge TP_{VAR}(IAS_{APATS}(APATS_i)) = \text{“simple”}$$

- **V_{APATS}** (**Valor** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo de asociación (*APATS*) y el conjunto de expresiones que devuelven valores (*EXPV*), de forma que a cada átomo se le asocia una expresión de valor (en la que pueden aparecer variables).

$$V_{APATS}: APATS \rightarrow EXPV$$

- **ATS_{APATS}** (**Atributo de asociación** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de par a atributo de asociación (*APATS*) y el conjunto de atributos de asociación (*ATS*), de forma que a cada átomo se le asocia un atributo de asociación concreto.

$$ATS_{APATS}: APATS \rightarrow ATS$$

Equipo de fútbol

Ejemplo de átomo de pertenencia de par (tupla, valor) a atributo de asociación: "Gaspart" (presidente del "Barcelona" desde el 20 de julio de 2001 no es propietario del club (EXBV representa una expresión booleana con un valor concreto o una variable).

```
APATS(IASV("Gaspart, "20.7.2001", "Barcelona"),EXBV(f), "propietario")
```

APT es el conjunto de **átomos de pertenencia de valor a tipo**. Cada elemento APT_i de este conjunto representa la condición de pertenencia de un valor a un tipo concreto.

Cada APT_i se define mediante una expresión de valor (que puede contener variables) y un tipo:

$$APT = \{APT_i \mid APT_i \cong (V_{APT}(APT_i), T_{APT}(APT_i))\}$$

- **V_{APT}** (**Valor** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de valor a tipo (*APT*) y el conjunto de expresiones que devuelven valores (*EXPV*), de forma que a cada átomo se le asocia una expresión de valor (que puede contener variables).

$$V_{APT}: APT \rightarrow EXPV$$

- **T_{APC}** (**Tipo** del átomo). Es una aplicación entre el conjunto de los átomos de pertenencia de valor a tipo (*APT*) y la unión entre el conjunto de tipos y el conjunto de variables ($T \cup VAR$), de forma que a cada átomo se le asocia un tipo concreto o una variable.

$$T_{APT}: APT \rightarrow T \cup VAR$$

  Si tipo de un átomo de pertenencia de valor a tipo de asociación es una variable, deberá ser una variable de conjunto de valores de un tipo:

$$\forall APT_i \in APT \bullet T_{APT}(APT_i) \in VAR \Rightarrow CJTO_{VAR}(T_{APT}(APT_i)) \in T \wedge TP_{VAR}(T_{APT}(APT_i)) = \text{“conjunto”}$$

Equipo de fútbol

Ejemplos de átomos de pertenencia de valor a tipo:

- El valor “Francia” pertenece al tipo “País comunitario”

```
APT(EXCV("Francia"),"País comunitario")
```

- La variable “m” pertenece al tipo “Mes”

```
APT(EXZV(VAR("m")), "Mes")
```

ACV es el conjunto de **átomos de comparación de valores**. Cada elemento ACV_i de este conjunto representa la comparación entre dos valores. Cada ACV_i se define mediante un operador de comparación y dos valores:

$$ACV = \{ACV_i \mid ACV_i \cong (CMP_{ACV}(ACV_i), VS_{ACV}(ACV_i))\}$$

- **CMP_{ACV}** (**Operador de comparación** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de valores (*ACV*) y el conjunto de operadores de comparación de valores (*CMPV*), de forma que a cada átomo se le asocia un operador concreto.

$$CMP_{ACV}: ACV \rightarrow CMPV$$

  CMPV es el conjunto de los **operadores de comparación de valores**: igual, distinto, menor, menor o igual, mayor, mayor o igual.

$$CMPV = \{=, \neq, <, \leq, >, \geq\}$$

- **VS_{ACV}** (**Valores** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de valores (*ACV*) y el producto cartesiano del conjunto de expresiones de valor consigo mismo ($EXPV \times EXPV$), de forma que a cada átomo se le asocian dos expresiones de valor (que pueden contener variables).

$$VS_{ACV}: ACV \rightarrow EXPV \times EXPV$$

Equipo de fútbol

Ejemplos de átomos de comparación de valores:

- La variable "p" (de tipo "Posición") es distinto de "Media punta"

```
ACV(≠, (EXCV(VAR("p")), EXCV("Media punta")))
```

- La variable "m" es menor que el valor 12

```
ACV(<, (EXZV(VAR("m")), EXZV(12)))
```

ACIC es el conjunto de **átomos de comparación de instancias de concepto**. Cada elemento $ACIC_i$ de este conjunto representa la comparación entre dos instancias de concepto. Cada $ACIC_i$ se define mediante un operador de comparación de instancias y dos instancias:

$$ACIC = \{ACIC_i \mid ACI_i \cong (CMP_{ACIC}(ACIC_i), IS_{ACIC}(ACIC_i))\}$$

- **CMP_{ACIC}** (**Operador de comparación** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de instancias (*ACIC*) y el conjunto de operadores de comparación de instancias (*CMPI*), de forma que a cada átomo se le asocia un operador concreto.

  $$CMP_{ACIC}: ACIC \rightarrow CMPI$$

  CMPI es el conjunto de los **operadores de comparación de instancias**: igual, distinto.

  $$CMPI = \{=, \neq\}$$

- **IS_{ACIC}** (**Instancias** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de instancias (*ACIC*) y el producto cartesiano de la unión de instancias de concepto y variables consigo mismo ($(I \cup VAR) \times (I \cup VAR)$), de forma que a cada átomo se le asocian dos elementos, cada uno de los cuales puede ser una instancia de concepto o una variable.

  $$IS_{ACIC}: ACIC \rightarrow (I \cup VAR) \times (I \cup VAR)$$

  Si alguna de las dos instancias de un átomo de comparación de instancias de concepto es una variable, deberá ser una variable simple de concepto:

  $$\forall ACIC_i \in ACIC \bullet (IS_{ACIC}(ACIC_i)[1] \in VAR \Rightarrow CJTO_{VAR}(IS_{ACIC}(ACIC_i)[1]) \in C \wedge TP_{VAR}(IS_{ACIC}(ACIC_i)[1]) = \text{"simple"}) \wedge (IS_{ACIC}(ACIC_i)[2] \in VAR \Rightarrow CJTO_{VAR}(IS_{ACIC}(ACIC_i)[2]) \in C \wedge TP_{VAR}(IS_{ACIC}(ACIC_i)[2]) = \text{"simple"})$$

Equipo de fútbol

Ejemplo de átomo de comparación de instancias:

- "Figo" (instancia de Jugador) es distinto que la variable "jug":

```
ACI(≠, ("Figo", VAR("jug")))
```

ACIAS es el conjunto de **átomos de comparación de instancias de asociación**. Cada elemento $ACIAS_i$ de este conjunto representa la comparación entre dos instancias de asociación. Cada $ACIAS_i$ se define mediante un operador de comparación de instancias y dos instancias:

$$ACIAS = \{ACIAS_i \mid ACIAS_i \cong (CMP_{ACIAS}(ACIAS_i), IS_{ACIAS}(ACIAS_i))\}$$

- **CMP_{ACIAS}** (**Operador de comparación** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de instancias (*ACIAS*) y el conjunto de operadores de comparación de instancias (*CMPI*), de forma que a cada átomo se le asocia un operador concreto.

$$CMP_{ACIAS}: ACIAS \rightarrow CMPI$$

- **IS_{ACIAS}** (**Instancias** del átomo). Es una aplicación entre el conjunto de los átomos de comparación de instancias (*ACIC*) y el producto cartesiano de la unión de instancias de asociación con variables y variables consigo mismo ($(IASV \cup VAR) \times (IASV \cup VAR)$), de forma que a cada átomo se le asocian dos elementos, cada uno de los cuales puede ser una instancia de asociación (con alguna variable) o una variable.

$$IS_{ACIAS}: ACIAS \rightarrow (IASV \cup VAR) \times (IASV \cup VAR)$$

  Si alguna de las dos instancias de un átomo de comparación de instancias de asociación es una variable, deberá ser una variable simple de asociación:

$$\forall ACIAS_i \in ACIAS \bullet (IS_{ACIAS}(ACIAS_i)[1] \in VAR \Rightarrow CJTO_{VAR}(IS_{ACIAS}(ACIAS_i)[1]) \in AS \wedge TP_{VAR}(IS_{ACIAS}(ACIAS_i)[1]) = \text{“simple”}) \wedge (IS_{ACIAS}(ACIAS_i)[2] \in VAR \Rightarrow CJTO_{VAR}(IS_{ACIAS}(ACIAS_i)[2]) \in AS \wedge TP_{VAR}(IS_{ACIAS}(ACIAS_i)[2]) = \text{“simple”})$$

4.2.3.1.5 Expresiones

EXPV es el **conjunto de las expresiones que devuelven valores**. Estas expresiones pueden ser de cuatro tipos.

$$EXPV = EXPZ \cup EXPR \cup EXPC \cup EXPB$$

EXPZ es el **conjunto de las expresiones que devuelven valores enteros**. Estas expresiones pueden ser de tres tipos.

$$EXPZ = EXZO \cup EXZU \cup EXZV$$

EXZO es el conjunto de las **expresiones que representan operaciones enteras binarias**. Cada elemento de este conjunto es una operación entera entre dos expresiones enteras. Una expresión de operación entera binaria se representa mediante un operador entero y las dos expresiones con las que opera:

$$EXZO = \{EXZO_i \mid EXZO_i \cong (OP_{EXZO}(EXZO_i), EXS_{EXZO}(EXZO_i))\}$$

Los elementos que definen una expresión de operación entera binaria son:

- **OP_{EXZO}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación entera binaria (*EXZO*) y el conjunto de operadores enteros binarios (*OPZB*), de forma que a cada expresión se le asocia un operador concreto.

$$OP_{EXZO}: EXZO \rightarrow OPZB$$

  OPZB es el **conjunto de los operadores enteros binarios**: suma, resta, multiplicación, división entera, módulo (resto de división) y potencia.

$$OPZB = \{+, -, *, /, \%, \wedge\}$$

- **EXS_{EXZO}** (**Expresiones** de la operación). Es una aplicación entre el conjunto de las expresiones de operación entera binaria (*EXZO*) y el producto cartesiano del

conjunto de expresiones enteras consigo mismo (*EXPZ× EXPZ*), de forma que a cada expresión de operación entera binaria se le asocian dos expresiones enteras de cualquier tipo.

$$EXS_{EXZO}: EXZO \rightarrow EXPZ \times EXPZ$$

EXZU es el conjunto de las **expresiones que representan operaciones enteras unarias**. Cada elemento de este conjunto es una operación entera sobre una expresión. Una expresión de operación entera unaria se representa mediante un operador entero y la expresión con la que opera:

$$EXZU = \{EXZU_i \mid EXZU_i \cong (OP_{EXZU}(EXZU_i), EX_{EXZU}(EXZU_i))\}$$

Los elementos que definen una expresión de operación entera unaria son:

- **OP_{EXZU}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación entera unaria (*EXZU*) y el conjunto de operadores enteros unarios (*OPZU*), de forma que a cada expresión se le asocia un operador concreto.

  $$OP_{EXZU}: EXZU \rightarrow OPZU$$

  OPZU es el conjunto de los **operadores enteros unarios**: cambio de signo y factorial.

  $$OPZU = \{ -, ! \}$$

- **EX_{EXZU}** (**Expresión** de la operación). Es una aplicación entre el conjunto de las expresiones de operación entera unaria (*EXZU*) y el conjunto de expresiones enteras (*EXPZ*), de forma que a cada expresión de operación entera unaria se le asigna una expresión entera de cualquier tipo.

  $$EX_{EXZU}: EXZU \rightarrow EXPZ$$

EXZV es el conjunto de las **expresiones que representan valores enteros concretos**. Cada elemento de este conjunto es un valor o una variable. Una expresión de valor se representa mediante su valor o variable:

$$EXZV = \{EXZV_i \mid EXZV_i \cong (V_{EXZV}(EXZV_i))\}$$

Una expresión entera de valor se define mediante un elemento:

- **V_{EXZV}** (**Valor** de la expresión). Es una aplicación entre el conjunto de las expresiones de valor entero (*EXZV*) y la unión del conjunto de números con el conjunto de variables (*NUM∪VAR*), de forma que a cada expresión se le asocia un valor entero o una variable.

  $$V_{EXZV}: EXZV \rightarrow NUM \cup VAR$$

  Si la expresión de valor entera está asociada a una variable, esta variable deberá ser una variable de tipo y su tipo deberá ser Z o bien entre sus antecesores se encuentra Z:

$$\forall EXZV_i \in EXZV\ VAR_i = V_{EXZV}(EXZV_i) \in VAR \Rightarrow CJTO_{VAR}(VAR_i) \in T \wedge (CJTO_{VAR}(VAR_i)=Z \vee Z \in ANT_T(CJTO_{VAR}(VAR_i)))$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo al que pertenece la expresión.

EXPR es el **conjunto de las expresiones que devuelven valores reales**. Estas expresiones pueden ser de tres tipos.

$$EXPR = EXRO \cup EXRU \cup EXRV$$

EXRO es el conjunto de las **expresiones que representan operaciones reales binarias**. Cada elemento de este conjunto es una operación real entre dos expresiones reales. Una expresión de operación real binaria se representa mediante un operador real y las dos expresiones con las que opera:

$$EXRO = \{EXRO_i \mid EXRO_i \cong (OP_{EXRO}(EXRO_i), EXS_{EXRO}(EXRO_i))\}$$

Los elementos que definen una expresión de operación real binaria son:

- **OP_{EXRO}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación real binaria (*EXRO*) y el conjunto de operadores reales binarios (*OPRB*), de forma que a cada expresión se le asocia un operador concreto.

  $$OP_{EXRO}: EXRO \rightarrow OPRB$$

  OPRB es el **conjunto de los operadores reales binarios**: suma, resta, multiplicación, división real, división entera, módulo (resto de división entera) y potencia.

  $$OPRB = \{+, -, *, /, div, \%, \hat{}\}$$

- **EXS_{EXRO}** (**Expresiones** de la operación). Es una aplicación entre el conjunto de las expresiones de operación real binaria (*EXRO*) y el producto cartesiano del conjunto de expresiones reales consigo mismo ($EXPR \times EXPR$), de forma que a cada expresión de operación real binaria se le asocian dos expresiones reales de cualquier tipo.

  $$EXS_{EXRO}: EXRO \rightarrow EXPR \times EXPR$$

EXRU es el conjunto de las **expresiones que representan operaciones reales unarias**. Cada elemento de este conjunto es una operación real sobre una expresión. Una expresión de operación real unaria se representa mediante un operador real y la expresión con la que opera:

$$EXRU = \{EXRU_i \mid EXRU_i \cong (OP_{EXRU}(EXRU_i), EX_{EXRU}(EXRU_i))\}$$

Los elementos que definen una expresión de operación real unaria son:

- **OP_{EXRU}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación real unaria (*EXRU*) y el conjunto de operadores reales unarios (*OPRU*), de forma que a cada expresión se le asocia un operador concreto.

  $$OP_{EXRU}: EXRU \rightarrow OPRU$$

  OPRU es el conjunto de los **operadores reales unarios**: cambio de signo.

  $$OPRU = \{ - \}$$

- **EX_{EXRU}** (**Expresión** de la operación). Es una aplicación entre el conjunto de las expresiones de operación real unaria (*EXRU*) y el conjunto de expresiones reales (*EXPR*), de forma que a cada expresión de operación real unaria se le asigna una expresión real de cualquier tipo.

$$EX_{EXRU}: EXRU \rightarrow EXPR$$

EXRV es el conjunto de las **expresiones que representan valores reales concretos**. Cada elemento de este conjunto es un valor o una variable. Una expresión de valor se representa mediante su valor o variable:

$$EXRV = \{EXRV_i \mid EXRV_i \cong (V_{EXRV}(EXRV_i))\}$$

Una expresión real de valor se define mediante un elemento:

- **V_{EXRV}** (**Valor** de la expresión). Es una aplicación entre el conjunto de las expresiones de valor real (*EXRV*) y la unión del conjunto de números con el conjunto de variables (*NUM*$\cup$*VAR*), de forma que a cada expresión se le asocia un valor real o una variable.

$$V_{EXRV}: EXRV \rightarrow NUM \cup VAR$$

  Si la expresión de valor real está asociada a una variable, esta variable deberá ser una variable de tipo y su tipo deberá ser R o bien entre sus antecesores se encuentra R:

$$\forall EXRV_i \in EXRV\ VAR_i = V_{EXRV}(EXRV_i) \in VAR \Rightarrow CJTO_{VAR}(VAR_i) \in T \wedge (CJTO_{VAR}(VAR_i)=R \vee R \in ANT_T(CJTO_{VAR}(VAR_i)))$$

  Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo al que pertenece la expresión.

EXPC es el **conjunto de las expresiones que devuelven cadenas**. Estas expresiones pueden ser de dos tipos:

$$EXPC = EXCO \cup EXCV$$

EXCO es el conjunto de las **expresiones que representan operaciones binarias de cadena**. Cada elemento de este conjunto es una operación entre dos expresiones de cadena. Una expresión de operación binaria de cadena se representa mediante un operador de cadena y las dos expresiones con las que opera:

$$EXCO = \{EXCO_i \mid EXCO_i \cong (OP_{EXCO}(EXCO_i), EXS_{EXCO}(EXCO_i))\}$$

- **OP_{EXCO}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación binaria de cadena (*EXCO*) y el conjunto de operadores binarios de cadena (*OPCB*), de forma que a cada expresión se le asocia un operador concreto.

$$OP_{EXCO}: EXCO \rightarrow OPCB$$

  OPCB es el **conjunto de los operadores binarios de cadena**: concatenación.

$$OPCB = \{+\}$$

- **EXS_{EXCO}** (**Expresiones** de la operación). Es una aplicación entre el conjunto de las expresiones de operación binaria de cadena (*EXCO*) y el producto cartesiano del conjunto de expresiones de cadena consigo mismo ($EXPC \times EXPC$), de forma que a cada expresión de operación binaria de cadena se le asocian dos expresiones de cadena de cualquier tipo.

$$EXS_{EXCO}: EXCO \rightarrow EXPC \times EXPC$$

EXCV es el conjunto de las **expresiones que representan valores de cadena concretos**. Cada elemento de este conjunto es un valor o una variable. Una expresión de valor se representa mediante su valor o variable:

$$EXCV = \{EXCV_i \mid EXCV_i \cong (V_{EXCV}(EXCV_i))\}$$

- **V_{EXCV}** (**Valor** de la expresión). Es una aplicación entre el conjunto de las expresiones de valor de cadena (*EXCV*) y la unión del conjunto de cadenas con el conjunto de variables ($CAD \cup VAR$), de forma que a cada expresión se le asocia una cadena o una variable.

$$V_{EXCV}: EXCV \rightarrow CAD \cup VAR$$

  Si la expresión de valor de cadena está asociada a una variable, esta variable deberá ser una variable de tipo y su tipo deberá ser CAD o bien entre sus antecesores se encuentra CAD:

$$\forall EXCV_i \in EXCV\ VAR_i = V_{EXCV}(EXCV_i) \in VAR \Rightarrow CJTO_{VAR}(VAR_i) \in T \wedge (CJTO_{VAR}(VAR_i) = CAD \vee CAD \in ANT_T(CJTO_{VAR}(VAR_i)))$$

  Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo al que pertenece la expresión.

EXPB es el **conjunto de las expresiones que devuelven valores booleanos**. Estas expresiones pueden ser de tres tipos:

$$EXPB = EXBO \cup EXBU \cup EXBV$$

EXBO es el conjunto de las **expresiones que representan operaciones booleanas binarias**. Cada elemento de este conjunto es una operación booleana entre dos expresiones booleana. Una expresión booleana de operación binaria se representa mediante un operador booleano y las dos expresiones con las que opera:

$$EXBO = \{EXBO_i \mid EXBO_i \cong (OP_{EXBO}(EXBO_i), EXS_{EXBO}(EXBO_i))\}$$

Los elementos que definen una expresión de operación booleana binaria son:

- **OP_{EXBO}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación booleana binaria (*EXBO*) y el conjunto de operadores booleanos binarios (*OPBB*), de forma que a cada expresión se le asocia un operador concreto.

$$OP_{EXBO}: EXBO \rightarrow OPBB$$

  OPNB es el **conjunto de los operadores booleanos binarios**: and, or, xor, implica ($A \rightarrow B \equiv \neg A \vee B$).

$$OPBB = \{\wedge, \vee, \oplus, \rightarrow\}$$

- **EXS_{EXBO}** (**Expresiones** de la operación). Es una aplicación entre el conjunto de las expresiones de operación booleana binaria (*EXBO*) y el producto cartesiano del conjunto de expresiones booleanas consigo mismo ($EXPB \times EXPB$), de forma que a cada expresión de operación booleana binaria se le asocian dos expresiones booleanas de cualquier tipo.

$$EXS_{EXBO}: EXBO \rightarrow EXPB \times EXPB$$

EXBU es el conjunto de las **expresiones que representan operaciones booleanas unarias**. Cada elemento de este conjunto es una operación booleana sobre una expresión. Una expresión de operación booleana unaria se representa mediante un operador booleano unario y la expresión con la que opera:

$$EXBU = \{EXBU_i \mid EXBU_i \cong (OP_{EXBU}(EXBU_i), EX_{EXBU}(EXBU_i))\}$$

Los elementos que definen una expresión de operación numérica unaria son:

- **OP_{EXBU}** (**Operador** de la expresión). Es una aplicación entre el conjunto de las expresiones de operación booleana unaria (*EXBU*) y el conjunto de operadores booleanos unarios (*OPBU*), de forma que a cada expresión se le asocia un operador concreto.

$$OP_{EXBU}: EXBU \rightarrow OPBU$$

  OPBU es el conjunto de los **operadores booleanos unarios**: not.

$$OPNU = \{ \neg \}$$

- **EX_{EXBU}** (**Expresión** de la operación). Es una aplicación entre el conjunto de las expresiones de operación booleana unaria (*EXBU*) y el conjunto de expresiones booleanas (*EXPB*), de forma que a cada expresión de operación numérica booleana se le asigna una expresión booleana de cualquier tipo.

$$EX_{EXBU}: EXBU \rightarrow EXPB$$

EXBV es el conjunto de las **expresiones que representan valores booleanos concretos**. Cada elemento de este conjunto es un valor o una variable. Una expresión de valor se representa mediante su valor o variable:

$$EXBV = \{EXBV_i \mid EXBV_i \cong (V_{EXBV}(EXBV_i))\}$$

Una expresión booleana de valor se define mediante un elemento:

- **V_{EXBV}** (**Valor** de la expresión). Es una aplicación entre el conjunto de las expresiones de valor booleano (*EXBV*) y la unión del conjunto de booleanos con el conjunto de variables ($B \cup VAR$), de forma que a cada expresión se le asocia un valor booleano o una variable.

$$V_{EXBV}: EXBV \rightarrow B \cup VAR$$

  Si la expresión de valor booleana está asociada a una variable, esta variable deberá ser una variable de tipo y su tipo deberá ser B:

$$\forall EXBV_i \in EXBV\ VAR_i = V_{EXBV}(EXBV_i) \in VAR \Rightarrow CJTO_{VAR}(VAR_i) \in T \wedge CJTO_{VAR}(VAR_i) = B$$

Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo al que pertenece la expresión.

Equipo de fútbol

Ejemplos expresiones de distintos tipos:

- Valor entero 10

```
EXZV(10)
```

- Valor cadena "Loïc Antonio Martínez Normand"

```
EXCV("Loïc Antonio Martínez Normand")
```

- Valor de cadena representado por la variable "país" (sólo se indica el nombre de la variable para simplificar la expresión).

```
EXCV(VAR("país"))
```

- Módulo (resto) de la variable "año" y 4

```
EXZO(%,(EXZV(VAR("año")),EXZV(4)))
```

4.2.3.2 Formalización de condiciones lógicas binarias

Las condiciones lógicas binarias son condiciones lógicas que se evalúan sobre dos estados. La principal diferencia radica en que las condiciones binarias contienen átomos que pueden hacer referencia a uno u otro estado.

4.2.3.2.1 Condiciones y Fórmulas

CNB es el **universo de las condiciones lógicas binarias** (o condiciones lógicas de doble estado). Estas condiciones lógicas pueden ser de 6 tipos.

$$CNB = CNBV \cup CNBC \cup CNBD \cup CNBI \cup CNBN \cup CNBF$$

Los elementos de CNB (condiciones binarias) van a seguir exactamente el mismo esquema que las condiciones de un estado (CND). La única diferencia es que las condiciones lógicas binarias contienen fórmulas lógicas de doble estado (binarias).

CNBV es el conjunto de **condiciones lógicas binarias con variable**.

$$CNBV = \{CNBV_i \mid CNBV_i \cong (VAR_{CBV}(CNBV_i), CNB_{CBV}(CNBV_i))\}$$

$$VAR_{CBV}: CNBV \rightarrow VAR$$

$$\forall CNB_i \in CNB \bullet \forall VAR_j, VAR_k \in VARS_{CNB}(CNB_i) \bullet VAR_j \neq VAR_k \Rightarrow N_{VAR}(VAR_j)) \neq N_{VAR}(VAR_k))$$

$$CNB_{CBV}: CNBV \rightarrow CNB$$

CNBC es el conjunto de **condiciones lógicas binarias de conjunción**.

$$CNBC = \{CNBC_i \mid CNBC_i \cong CNBS_{CBC}(CNBC_i)\}$$

$$CNBS_{CBC}: CNBC \rightarrow CNB \times CNB$$

CNBD es el conjunto de **condiciones lógicas binarias de disyunción**.

$$CNBD = \{CNBD_i \mid CNBD_i \cong CNBS_{CBD}(CNBD_i)\}$$

$$CNBS_{CBD}: CNBD \rightarrow CNB \times CNB$$

CNBI es el conjunto de **condiciones lógicas binarias de implicación**.

$$CNBI = \{CNBI_i \mid CNBI_i \cong CNBS_{CBI}(CNBI_i)\}$$

$$CNBS_{CI}: CNBI \rightarrow CNB \times CNB$$

CNBN es el conjunto de **condiciones lógicas binarias de negación**.

$$CNBN = \{CNBN_i \mid CNBN_i \cong CNB_{CBN}(CNBN_i)\}$$

$$CNB_{CN}: CNBN \rightarrow CNB$$

CNBF es el conjunto de **condiciones lógicas binarias de fórmula.**

$$CNBF = \{CNBF_i \mid CNBF_i \cong FORB_{CBF}(CNBF_i)\}$$

$$FORB_{CBF}: CNBF \rightarrow FORB$$

Equipo de fútbol

Seguidamente se muestra de manera formal la condición binaria de incremento de la fuerza de un jugador en 0.5 unidades. Esta condición tiene una variable libre que representa a un jugador cualquiera.

Esta condición se puede representar en lenguaje natural de la siguiente forma: "sea "jug" un Jugador, si existe una valoración "f1" tal que sea la fuerza de ese jugador en el estado 1 (inicial) y sea menor o igual que 9.5, entonces existe una valoración "f2" tal que "f2" es igual a "f1" más 0.5 y "f1" no es la fuerza del jugador en el estado 2 y "f2" sí es la fuerza del jugador en el estado 2 (final)".

```
CNBV(VAR("jug", ∅, "Jugador"),
  CNBV(VAR("f1", ∃, "Valoración"),
    CNBI(
      CNBF(FOBC(
        FOBA(BPAT(VAR("jug"), EXZV(VAR("f1")), "fuerza", 1)),
        FOBA(ACV(≤, EXZV(VAR("f1")), EXZV(9.5)))
      )),
      CNBV(VAR("f2", ∃, "Valoración"),
        CNBF(FOBC(
          FOBA(ACV(=, EXZV(VAR("f2")),
                      EXZO(+, EXZV(VAR("f1")), EXZV(0.5)))),
          FOBC(
            FOBN(FOBA(BPAT(VAR("jug"), EXZV(VAR("f1")), "fuerza", 2))),
            FOBA(BPAT(VAR("jug"), EXZV(VAR("f2")), "fuerza", 2))
          )
        ))
      )
    )
  )
)
```

Ésta es una condición en la que aparecen:

- 3 condiciones binarias con definición de variable (CNBV), una condición binaria de implicación (CNBI) y 2 condiciones binarias de fórmula (CNBF).
- 3 fórmulas binarias de conjunción (FOBC), una fórmula binaria de negación (FOBN) y 5 fórmulas binarias de átomos (FOBA).
- 3 átomos binarios de pertenencia de par (instancia, valor) a atributo de concepto en uno de los estados (BPAT) y 2 átomos de comparación de valores (ACV).
- 8 expresiones de valor real (EXZV) y una expresión de operación binaria real (EXZO).

Una vez definidas formalmente las condiciones lógicas binarias, es conveniente definir una función que permita conocer el **conjunto de variables de una condición lógica binaria** (***$VARS_{CNB}$***). Es una aplicación entre el conjunto de condiciones lógicas binarias de cualquier tipo (*CNB*) y las partes del conjunto de variables (*P(VAR)*), de forma que, dada una condición lógica, devuelve el conjunto de variables de dicha condición.

$$VARS_{CNB}: CNB \rightarrow P(VAR)$$

El cálculo de la función $VARS_{CNB}$ depende del tipo de condición:

- Si se trata de una condición binaria de variable (CNBV), entonces su conjunto de variables será la variable definida, más las variables de su condición asociada.

$$\forall CNB_i \in CNBV\ VARS_{CNB}(CNB_i)=VAR_{CBV}(CNB_i)\cup VARS_{CNB}(CNB_{CBV}(CNB_i))$$

- Si se trata de una condición binaria de conjunción (CNBC), entonces su conjunto de variables será la unión de los conjuntos de variables de las condiciones asociadas.

$$\forall CNB_i \in CNBC \text{ sea } (CNB_j, CNB_k)=CNBS_{CBC}(CNB_i) \text{ entonces } VARS_{CNB}(CNB_i)= VARS_{CNB}(CNB_j)\cup VARS_{CNB}(CNB_k)$$

- Si se trata de una condición binaria de disyunción (CNBD), su conjunto de variables será la unión de los conjuntos de variables de las condiciones asignadas.

$$\forall CNB_i \in CNBD \text{ sea } (CNB_j, CNB_k)=CNBS_{CBD}(CNB_i) \text{ entonces } VARS_{CNB}(CNB_i)= VARS_{CNB}(CNB_j)\cup VARS_{CNB}(CNB_k)$$

- Si se trata de una condición binaria de implicación (CNBI), su conjunto de variables será la unión de los conjuntos de variables de las condiciones asignadas.

$$\forall CNB_i \in CNBI \text{ sea } (CNB_j, CNB_k)=CNBS_{CBI}(CNB_i) \text{ entonces } VARS_{CNB}(CNB_i)= VARS_{CNB}(CNB_j)\cup VARS_{CNB}(CNB_k)$$

- Si se trata de una condición binaria de negación (CNBN), su conjunto de variables será el conjunto de variables de la condición negada.

$$\forall CNB_i \in CNBN\ VARS_{CNB}(CNB_i)= VARS_{CNB}(CNB_{CBN}(CNB_i))$$

- Si se trata de una condición binaria de fórmula (CNBF), su conjunto de variables será el conjunto vacío.

$$\forall CNB_i \in CNBF\ VARS_{CNB}(CNB_i)=\varnothing$$

FORB es el **universo de las fórmulas lógicas binarias** (expresiones lógicas binarias sin definición de variables). Las fórmulas pueden ser de 5 tipos.

$$FORB = FOBC \cup FOBD \cup FOBI \cup FOBN \cup FOBA$$

FOBC es el conjunto de **fórmulas binarias de conjunción**.

$$FOBC = \{FOBC_i \mid FOBC_i \cong FOBS_{FBC}(FOBC_i)\}$$

$$FOBS_{FBC}: FOBC \rightarrow FORB \times FORB$$

FOBD es el conjunto de **fórmulas binarias de disyunción**.

$$FOBD = \{FOBD_i \mid FOBD_i \cong FOBS_{FBD}(FOBD_i)\}$$

$$FOBS_{FD}: FOBD \rightarrow FORB \times FORB$$

FOBI es el conjunto de **fórmulas binarias de implicación**.

$$FOBI = \{FOBI_i \mid FOBI_i \cong FOBS_{FBI}(FOBI_i)\}$$

$$FOBS_{FBI}: FOBI \rightarrow FORB \times FORB$$

FOBN es el conjunto de **fórmulas binarias de negación**.

$$FOBN = \{FOBN_i \mid FOBN_i \cong FORB_{FBN}(FOBN_i)\}$$

$$FORB_{FBN}: FOBN \rightarrow FORB$$

FOBA es el conjunto de **fórmulas binarias de átomo**.

$$FOBA = \{FOBA_i \mid FOBA_i \cong ATOB_{FBA}(FOBA_i)\}$$

$$ATOB_{FBA}: FOBA \rightarrow ATOB$$

Equipo de fútbol

Seguidamente se muestran las fórmulas binarias aparecidas en la condición de incremento de la fuerza de un jugador en 0.5 unidades:

- Fórmula binaria de átomo que indica que el valor representado por la variable "f1" es la fuerza del jugador representado por la variable "jug" en el estado 1 (inicial).

```
FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 1))
```

- Fórmula binaria de átomo que representa que el valor de la variable "f1" es menor o igual que 9.5.

```
FOBA(ACV(≤, EXRV(VAR("f1")), EXRV(9.5))
```

- Fórmula binaria de conjunción de las dos anteriores ("f1" es la fuerza de "jug" y "f1" es menor o igual que 9.5).

```
FOBC(FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 1)), FOBA(ACV(≤,
EXRV(VAR("f1")), EXRV(9.5)))
```

- Fórmula de átomo que representa que el valor de "f2" es igual al valor de "f1" más 0.5.

```
FOBA(ACV(=, EXRV(VAR("f2")), EXRO(+, EXRV(VAR("f1")), EXRV(0.5))))
```

- Fórmula binaria de átomo que indica que el valor representado por la variable "f1" es la fuerza del jugador representado por la variable "jug" en el estado 2 (final).

```
FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 2))
```

- Fórmula binaria de negación que niega la fórmula anterior ("f1" no es la fuerza de "jug" en el estado 2).

```
FOBN(FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 2)))
```

- Fórmula binaria de átomo que indica que el valor representado por la variable "f2" es la fuerza del jugador representado por la variable "jug" en el estado 2 (final).

```
FOBA(BPAT(VAR("jug"), EXRV(VAR("f2")), "fuerza", 2))
```

- Fórmula binaria de conjunción entre las dos fórmulas anteriores ("f1" no es la fuerza de "jug" en el estado 2 y "f2" es la fuerza de "jug" en el estado 2).

```
FOBC(FOBN(FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 2))),
FOBA(BPAT(VAR("jug"), EXRV(VAR("f2")), "fuerza", 2)))
```

- Fórmula binaria de conjunción entre la fórmula anterior y la fórmula de comparación entre los valores de "f1" y "f2".

```
FOBC(FOBA(ACV(=, EXRV(VAR("f2")), EXRO(+, EXRV(VAR("f1")), EXRV(0.5)))),
FOBC(FOBN(FOBA(BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 2))),
FOBA(BPAT(VAR("jug"), EXRV(VAR("f2")), "fuerza", 2))))
```

4.2.3.2.2 Átomos

ATOB es el **universo de los átomos binarios**. Los átomos pueden ser de 8 tipos:.

$$ATOB = BPC \cup BPAT \cup BPAS \cup BPATS \cup APT \cup ACV \cup ACIC \cup ACIAS$$

Los átomos de pertenencia de valor a tipo (APT), de comparación de valores (ACV), de comparación de instancias de concepto (ACIC) y de comparación de instancias de asociación (ACIAS) son los mismos que los descritos para las condiciones de un estado, dado que su evaluación no requiere hacer referencia explícita a uno de los dos instantes de tiempo de una condición binaria.

BPC es el conjunto de **átomos binarios de pertenencia de instancia a concepto**. Cada elemento BPC_i de este conjunto representa la condición de pertenencia de una instancia de concepto a un concepto concreto en uno de los dos instantes de tiempo a los que puede hacer referencia la condición binaria la que pertenece el átomo. Cada BPC_i se define mediante una instancia de concepto (que puede ser una variable), un concepto y una indicación del instante de tiempo:

$$BPC = \{BPC_i \mid BPC_i \cong (I_{BPC}(BPC_i), C_{BPC}(BPC_i), TI_{BPC}(BPC_i))\}$$

Los elementos que definen un átomo binario de pertenencia de instancia a concepto son:

- **I_{BPC}** (**Instancia** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de instancia a concepto (*BPC*) y la unión entre el conjunto de instancias de conceptos y el conjunto de definiciones de variables ($I \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o bien una variable.

  $$I_{BPC}: BPC \rightarrow I \cup VAR$$

  Si la instancia de un átomo binario de pertenencia de instancia a concepto es una variable, deberá ser una variable de concepto (es decir, su conjunto deberá ser un concepto):

  $$\forall BPC_i \in BPC\ I_{BPC}(BPC_i) \in VAR \Rightarrow CJTO_{VAR}(I_{BPC}(BPC_i)) \in C$$

  Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

- **C_{BPC}** (**Concepto** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de instancia a concepto (*BPC*) y la unión de conceptos y variables ($C \cup VAR$), de forma que a cada átomo se le asocia un concepto concreto o una variable.

  $$C_{BPC}: BPC \rightarrow C \cup VAR$$

- **TI_{BPC}** (**Estado** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de instancia a concepto (*BPC*) y el conjunto de representaciones de instantes de tiempo (*TI*).

$$TI_{BPC}: BPC \rightarrow TI$$

TI es el conjunto de representaciones de instantes de tiempo de condición binaria. Es un conjunto formado por los valores 1 (instante inicial) y 2 (instante final):

$$TI = \{1, 2\}$$

El valor de TI_{BPC} indicará si la pertenencia de la instancia al concepto debe cumplirse en el estado inicial de la condición binaria (valor 1) o en el estado final (valor 2).

BPAT es el conjunto de **átomos binarios de pertenencia de par a atributo de concepto**. Cada elemento $BPAT_i$ de este conjunto representa la condición de pertenencia de un par (instancia de concepto, valor) a un atributo de concepto concreto en uno de los estados de la condición binaria. Cada $BPAT_i$ se define mediante una instancia de concepto (que puede ser una variable), un valor, un atributo y una representación de estado:

$$BPAT = \{BPAT_i \mid BPAT_i \cong (I_{BPAT}(BPAT_i), V_{BPAT}(BPAT_i), AT_{BPAT}(BPAT_i), TI_{BPAT}(BPAT_i))\}$$

Los elementos que definen un átomo binario de pertenencia de par a atributo de concepto son los siguientes:

- **I_{BPAT}** (**Instancia** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo (*BPAT*) y la unión entre el conjunto de instancias de conceptos y el conjunto de definiciones de variables ($I \cup VAR$), de forma que a cada átomo se le asocia una instancia concreta o bien una variable.

  $$I_{BPAT}: BPAT \rightarrow I \cup VAR$$

  Si la instancia de un átomo de pertenencia de par a atributo es una variable, deberá ser una variable de concepto (es decir, su conjunto deberá ser un concepto):

  $$\forall BPAT_i \in BPAT\ I_{BPAT}(BPAT_i) \in VAR \Rightarrow CJTO_{VAR}(I_{BPAT}(BPAT_i)) \in C$$

  Además esa variable deberá pertenecer a las variables de la cadena de condiciones lógicas dentro de la cual se encuentra la fórmula a la que pertenece el átomo.

- **V_{BPAT}** (**Valor** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo (*BPAT*) y el conjunto de expresiones que devuelven valores (*EXPV*), de forma que a cada átomo se le asocia una expresión de valor.

  $$V_{BPAT}: BPAT \rightarrow EXPV$$

  EXPV es el mismo conjunto de **expresiones de valores** que el definido para el caso de condiciones de un único estado. Los valores tienen existencia propia independientemente del estado y, por lo tanto, las expresiones de valor no dependen de ningún estado.

- **AT_{BPAT}** (**Atributo** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo (*BPAT*) y el conjunto de atributos de concepto (*AT*), de forma que a cada átomo se le asocia un atributo concreto.

  $$AT_{BPAT}: BPAT \rightarrow AT$$

- **TI_{BPAT}** (**Estado** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo (*BPAT*) y el conjunto de representaciones de instantes de tiempo (*TI*).

$$TI_{BPAT}: BPAT \rightarrow TI$$

BPAS es el conjunto de **átomos binarios de pertenencia de tupla a asociación**. Cada elemento $BPAS_i$ de este conjunto representa la condición de pertenencia de una tupla (instancia de asociación) a una asociación concreta en un estado. Cada $BPAS_i$ se define mediante una tupla (que puede contener variables), una asociación y un estado:

$$BPAS = \{BPAS_i \mid BPAS_i \cong (IAS_{BPAS}(BPAS_i), AS_{BPAS}(BPAS_i), TI_{BPAS}(BPAS_i))\}$$

Los elementos que definen un átomo binario de pertenencia de tupla a asociación son:

- **IAS_{BPAS}** (**Instancia de asociación** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de tupla a asociación (*BPAS*) y el conjunto de instancias de asociación con variables (*IASV*$\cup$*VAR*), de forma que a cada átomo se le asocia una instancia concreta.

$$IAS_{BPAS}: BPAS \rightarrow IASV \cup \mathrm{VAR}$$

- **AS_{BPAS}** (**Asociación** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de tupla a asociación (*BPAS*) y el conjunto de asociaciones (*AS*), de forma que a cada átomo se le asocia una asociación concreta.

$$AS_{BPAS}: BPAS \rightarrow AS$$

- **TI_{BPAS}** (**Estado** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de tupla a asociación (*BPAS*) y el conjunto de representaciones de estado (*TI*), de forma que a cada átomo se le asocia una representación de estado.

$$TI_{BPAS}: BPAS \rightarrow TI$$

BPATS es el conjunto de **átomos binarios de pertenencia de par a atributo de asociación**. Cada elemento $BPATS_i$ de este conjunto representa la condición de pertenencia de un par (instancia de asociación, valor) a un atributo de asociación concreto en un estado. Cada $BPATS_i$ se define mediante una instancia de asociación (que puede contener variables), un valor, un atributo de asociación y un estado:

$$BPATS = \{BPATS_i \mid BPATS_i \cong (IAS_{BPATS}(BPATS_i), V_{BPATS}(BPATS_i), ATS_{BPATS}(BPATS_i), TI_{BPATS}(BPATS_i))\}$$

Los elementos que definen un átomo binario de pertenencia de par a atributo de asociación son:

- **IAS_{BPATS}** (**Instancia de asociación** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo de asociación (*BPATS*) y el conjunto de instancias de asociación con variables (*IASV*$\cup$*VAR*), de forma que a cada átomo se le asocia una instancia de asociación concreta.

$$IAS_{BPATS}: BPATS \rightarrow IASV \cup \mathrm{VAR}$$

- **V_{BPATS}** (**Valor** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo de asociación (*BPATS*) y el conjunto de

expresiones que devuelven valores (*EXPV*), de forma que a cada átomo se le asocia una expresión de valor.

$$V_{BPATS}: BPATS \rightarrow EXPV$$

- **ATS_{BPATS}** (**Atributo de asociación** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo de asociación (*BPATS*) y el conjunto de atributos de asociación (*ATS*), de forma que a cada átomo se le asocia un atributo concreto.

$$ATS_{BPATS}: BPATS \rightarrow ATS$$

- **TI_{BPATS}** (**Estado** del átomo). Es una aplicación entre el conjunto de los átomos binarios de pertenencia de par a atributo de asociación (*BPATS*) y el conjunto de representaciones de estado (*TI*), de forma que a cada átomo se le asocia una representación de estado.

$$TI_{BPATS}: BPATS \rightarrow TI$$

Equipo de fútbol

Seguidamente se muestran los átomos binarios aparecidos en la condición de incremento de fuerza de un jugador en 0.5.

- El valor representado por la variable "f1" es la fuerza del jugador representado por la variable "jug" en el estado 1.

```
BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 1)
```

- El valor representado por la variable "f1" es la fuerza del jugador representado por la variable "jug" en el estado 2.

```
BPAT(VAR("jug"), EXRV(VAR("f1")), "fuerza", 2)
```

- El valor representado por la variable "f2" es la fuerza del jugador representado por la variable "jug" en el estado 2.

```
BPAT(VAR("jug"), EXRV(VAR("f2")), "fuerza", 2)
```

- El valor representado por la variable "f2" es igual al valor representado por la variable "f1" más 0.5

```
ACV(=, EXRV(VAR("f2")), EXRO(+, EXRV(VAR("f1")), EXRV(0.5)))
```

4.2.3.3 Evaluación de condiciones lógicas unarias

En este apartado se va a definir la función de evaluación de condiciones lógicas. Esta función indicará si una condición lógica es cierta, falsa o desconocida en un estado concreto de un modelo estructural.

Se van a definir funciones de evaluación para todos los elementos de las expresiones lógicas: condiciones lógicas, fórmulas, átomos. También se definirá la evaluación de expresiones numéricas, de cadena y booleanas.

Los detalles de la evaluación de condiciones lógicas unarias (y binarias) se encuentran el en Anexo B.

La **función de evaluación de condiciones** (**$EVAL_{CND}$**) es una aplicación entre el producto cartesiano del conjunto de modelos estructurales, con el conjunto de condiciones lógicas, con el conjunto de enteros (instantes de tiempo) y con las partes del conjunto de asignaciones de variables ($EST \times CND \times N \times \Pi(ASV)$) y la unión entre el conjunto de valores booleanos y el conjunto cuyo único elemento es el conjunto vacío

($B \cup \{\varnothing\}$), de forma que, dada una condición lógica, dado un estado y dado un conjunto de asignaciones de variables, la función indica si la condición se evalúa a cierto (v), falso (f) o desconocido ($\varnothing$).

$$EVAL_{CND}: EST \times CND \times ST \times \Pi(ASV) \rightarrow B \cup \{\varnothing\}$$

ASV es el **conjunto de asignaciones de variables**. Cada elemento de este conjunto representa la asignación de un valor concreto a una variable. Para definir una asignación de variable se necesitan dos parámetros: la variable y su valor.

$$ASV = \{ ASV_i \mid ASV_i = (VAR_{ASV}(ASV_i), VAL_{ASV}(ASV_i)) \}$$

Los elementos de una asignación de variable son dos:

- **VAR_{ASV}** (**Variable** de la asignación). Es una aplicación entre el conjunto de las asignaciones de variable (ASV) y el conjunto de variables (VAR), de forma que a cada asignación se le asocia una variable concreta.

  $$VAR_{ASV}: ASV \rightarrow VAR$$

- **VAL_{ASV}** (**Valor** de la asignación). Es una aplicación entre el conjunto de las asignaciones de variable (ASV) y la unión entre el conjunto el conjunto de valores , el conjunto de instancias de concepto y el conjunto de instancias de asociación ($I \cup V$).

  $$VAL_{ASV}: ASV \rightarrow V \cup I \cup IAS$$

  Para cada asignación de variable se debe cumplir que, si su variable es de concepto, entonces su valor deberá ser una instancia perteneciente al concepto de la variable. Si, por otro lado, la variable de la asignación es una variable de tipo, entonces su valor deberá pertenecer al tipo de la variable:

  $$\forall ASV_i \in ASV \; CJTO_{VAR}(VAR_{ASV}(ASV_i)) \in C \Rightarrow VAL_{ASV}(ASV_i) \in I \wedge VAL_{ASV}(ASV_i) \in CJTO_{VAR}(VAR_{ASV}(ASV_i))$$

  $$\forall ASV_i \in ASV \; CJTO_{VAR}(VAR_{ASV}(ASV_i)) \in T \Rightarrow VAL_{ASV}(ASV_i) \in V \wedge VAL_{ASV}(ASV_i) \in CJTO_{VAR}(VAR_{ASV}(ASV_i))$$

  $$\forall ASV_i \in ASV \; CJTO_{VAR}(VAR_{ASV}(ASV_i)) \in AS \Rightarrow VAL_{ASV}(ASV_i) \in IAS \wedge VAL_{ASV}(ASV_i) \in CJTO_{VAR}(VAR_{ASV}(ASV_i))$$

Una evaluación de una condición será válida si y sólo si todas sus asignaciones de variables se corresponden con variables de la condición:

$$\forall EST_i \in EST \;\; \forall CND_i \in CND \;\; \forall t_j \in N \;\; \forall ASS_k \in \Pi(ASV) \; EVAL_{CND}(EST_i, CND_i, t_j, ASS_k)) \text{ es válida} \Leftrightarrow \forall ASV_l \in ASS_k \; VAR_{ASV}(ASV_l) \in VARS_{CND}(CND_i)$$

De forma similar se define la evaluación de fórmulas, átomos y expresiones. Los detalles pueden encontrarse en el Anexo B.

Debe señalarse que pueden existir condiciones lógicas que, aun siendo sintácticamente correctas, no puedan evaluarse, debido a sus peculiaridades. Por ejemplo, no es posible saber en un tiempo finito si se cumple una condición con la variable siguiente:

```
VAR("p", ∀, T"Real")
```

Esta variable está cuantificada universalmente en todos los números reales, que como es bien sabido no es un conjunto numerable y, por tanto, no pueden recorrerse todos sus elementos. Este tipo de problemas también se da en las condiciones binarias.

4.2.3.4 Evaluación de condiciones lógicas binarias

La **función de evaluación de condiciones binarias** (**$EVAL_{CNB}$**) es una aplicación entre el producto cartesiano del conjunto de modelos estructurales, con el conjunto de condiciones lógicas binarias con dos conjuntos de naturales y con las partes del conjunto de asignaciones de variables ($EST \times CNB \times N \times N \times \Pi(ASV)$) y la unión entre el conjunto de valores booleanos y el conjunto cuyo único elemento es el conjunto vacío ($B \cup \{\varnothing\}$), de forma que, dada una condición lógica, dados dos estados y dado un conjunto de asignaciones de variables, la función indica si la condición se evalúa a cierto (v), falso (f) o desconocido ($\varnothing$) para esos dos estados.

$$EVAL_{CNB}: EST \times CNB \times N \times N \times \Pi(ASV) \rightarrow B \cup \{\varnothing\}$$

ASV es el **conjunto de asignaciones de variables**, tal y como se definió al describir la evaluación de condiciones lógicas sobre un estado. Cada elemento de este conjunto representa la asignación de un valor concreto a una variable.

Una evaluación de una condición será válida si y sólo si todas sus asignaciones de variables se corresponden con variables de la condición:

$$\forall EST_i \in EST \ \forall CNB_i \in CNB \ \forall t_j \in N \ \forall t_k \in N \ \forall ASS_l \in \Pi(ASV) \ EVAL_{CNB}(EST_i, CNB_i, t_j, t_k, ASS_l)) \text{ es válida} \Leftrightarrow \forall ASV_x \in ASS_k \ VAR_{ASV}(ASV_x) \in VARS_{CND}(CNB_i)$$

Los detalles de la evaluación de condiciones lógicas binarias se encuentran en el Anexo B.

4.2.3.5 Formalización de Invariantes de Concepto

En este apartado se van a describir las características propias de aquellas condiciones lógicas que son invariantes de concepto. Se comenzará por recordar el término **invariante de concepto**.

INV_C (**Invariante** de concepto) es la condición que deben cumplir todas las instancias del concepto. Es una aplicación inyectiva entre el conjunto de los conceptos (C) y el universo de las condiciones lógicas (CND), de forma que a cada concepto se le asocia su invariante:

$$INV_C: C \rightarrow CND$$

Toda condición que sea invariante de concepto tendrá una única **variable libre,** con el mismo nombre que el concepto que representará a cualquier instancia de ese concepto. Además, cualquier otra variable de la condición deberá estar cuantificada.

$$\forall C_i \in C, \text{ sea } CND_j = INV_C(C_i) \text{ entonces } \exists VAR_k \in VARS_{CND}(CND_j) \text{ tal que } N_{VAR}(VAR_k) = N_C(C_i) \wedge CJTO_{VAR}(VAR_k) = C_i \wedge CUA_{VAR}(VAR_k) = \varnothing \wedge (\forall VAR_l \in VARS_{CND}(CND_j) \ VAR_l \neq VAR_k \Rightarrow CUA_{VAR}(VAR_l) \neq \varnothing)$$

Dados un concepto C_i y una instancia I_j, entonces la evaluación de la invariante del concepto C_i, teniendo como única asignación el de la instancia I_j como valor de la

variable que representa al concepto, será cierta si y sólo si la instancia pertenece al concepto. Expresado formalmente:

$$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall C_i \in C_{EST}(EST_i) \bullet \forall I_j \in I \bullet, EVAL_{CND}(EST_i, INV_C(C_i), t_i, \{(N_C(C_i), I_j)\}) = v \Leftrightarrow I_j \in ST_C(C_i, t_i)$$

Equipo de fútbol

Seguidamente se va a poner la definición completa del concepto "Fecha", con la invariante que dice cuándo es válida una fecha. En esa invariante se puede apreciar cómo la primera variable es libre y tiene como nombre el nombre del concepto.

```
C("Fecha",
  CND(
    sea ?"Fecha" de C"Fecha"
     existe ?"d" de T"Día"
      existe ?"m" de T"Mes"
       existe ?"a" de T"Año"
        si
          (?"Fecha", ?"d") pertenece a AT"Fecha"."día" and
          (?"Fecha", ?"m") pertenece a AT"Fecha"."mes" and
          (?"Fecha", ?"a") pertenece a AT"Fecha"."año"
        entonces
          (?"m"=4 or ?"m"=6 or ?"m"=9 or ?"m"=11) and (?"d"<31) or
          (?"m"=2 and ?"a"%4=0 and (?"a"%100!=0 or ?"a"%400=0) and
          (?"d"<30) or
          (?"m"=2 and (?"a"%4!=0 or (?"a"%100=0 and ?"a"%400!=0)) and
          (?"d"<30)
        fin si
       fin condición
      fin condición
     fin condición
    fin condición
  )
)
```

4.2.3.6 Formalización de Invariantes de Asociación

En este apartado se van a describir las características propias de aquellas condiciones lógicas que son invariantes de asociación. Se comenzará por recordar el término **invariante de asociación**.

INV_{AS} (**Invariante** de asociación) es la condición que deben cumplir todas las instancias de la asociación. Es una aplicación inyectiva entre el conjunto de los conceptos (*C*) y el universo de las condiciones lógicas (*CND*), de forma que a cada asociación se le asocia su invariante:

$$INV_{AS}: AS \rightarrow CND$$

Toda condición que sea invariante de concepto tendrá una única **variable libre,** con el mismo nombre que la asociación que representará a cualquier tupla de esa asociación. Además, cualquier otra variable de la condición deberá estar cuantificada.

$$\forall AS_i \in AS, \text{ sea } CND_j = INV_{AS}(AS_i) \text{ entonces } \exists VAR_k \in VARS_{CND}(CND_j) \text{ tal que } N_{VAR}(VAR_k) = N_{AS}(AS_i) \wedge CJTO_{VAR}(VAR_k) = AS_i \wedge CUA_{VAR}(VAR_k) = \varnothing \wedge (\forall VAR_l \in VARS_{CND}(CND_j)\ VAR_l \neq VAR_k \Rightarrow CUA_{VAR}(VAR_l) \neq \varnothing)$$

Dados una asociación AS_i y una instancia de asociación IAS_j, entonces la evaluación de la invariante de la asociación AS_i, teniendo como única asignación la de la instancia IAS_j como valor de la variable que representa a la asociación, será cierta si y sólo si la

instancia pertenece estado de la asociación en ese instante de tiempo. Expresado formalmente:

$$\forall t_i \in N \bullet \forall EST_i \in EST \bullet \forall AS_i \in AS_{EST}(EST_i) \bullet \forall IAS_j \in IAS \bullet, EVAL_{CND}(EST_i, INV_{AS}(AS_i), t_i, \{(N_{AS}(AS_i), IAS_j)\}) = v \Leftrightarrow IAS_j \in ST_{AS}(AS_i, t_i)$$

4.3 LENGUAJE DE REPRESENTACIÓN

En este apartado se va a definir un lenguaje para representar todos los elementos del modelo conceptual. Se va a seguir el mismo orden de presentación que en los apartados anteriores.

Respecto a las definiciones realizadas hasta ahora, el lenguaje de representación aporta un nuevo elemento de modelado: las descripciones, que pueden asociarse a gran parte de los elementos del modelo. Una descripción es un párrafo en lenguaje natural que aporta información sobre el elemento correspondiente.

A lo largo de la exposición del lenguaje las "palabras reservadas" aparecerán en negrita, para facilitar su identificación. En el uso habitual del lenguaje se utilizarán mayúsculas para identificarlas.

4.3.1 Lenguaje para el Modelo Estructural

4.3.1.1 Valores y Tipos

Representación de **valores**:

1. Enteros y reales: se representan de la forma habitual → 0, 1.3, -4, ...
2. Cadenas: se representan como texto encerrado entre comillas dobles → "Juan", "Figo", "Delantero", ...
3. Booleanos: se representan con una palabra completa → verdad, falso

Representación de los **tipos básicos**:

- R → **Real**
- Z → **Entero**
- CAD → **Cadena**
- B → **Booleano**

Para representar de un **tipo derivado** se utilizará la siguiente notación:

Tipo <nombre> (<tipo base>)
[**descripción:**
  <descripción>]
definición:
  <definición>
Fin tipo

Significado de los campos:

- <nombre>: nombre del tipo derivado.
- <tipo base>: nombre del tipo del que deriva.

- <descripción>: descripción del tipo representada como una cadena de caracteres encerrada entre comillas dobles. Esta descripción es opcional.
- <definición>: define cómo es el tipo derivado respecto a su tipo base. Su representación dependerá de la forma en la que se deriva el tipo:
  - Derivación de enteros o reales mediante intervalos. Se representará cada intervalo usando la notación matemática habitual (teniendo en cuenta que el valor ∞ se representa como infinito). Si hay más de un intervalo se pondrán separados por el signo '+', como forma de representar la unión de intervalos:

    <definición> → <intervalo>[+ <intervalo>] ... [+ <intervalo>]
  - Derivación de cadena mediante lista de valores. Se representan todos los valores, encerrados entre llaves:

    <definición> → {<valor1>, <valor2>, ..., <valorn>}
  - Derivación de tipo mediante restricción calculada. Se representará el tipo de operación (diferencia o intersección) y el segundo tipo usado en el cálculo:

    <definición> → [**diferencia** | **intersección**] <tipo derivado>

Equipo de fútbol

Tipos derivados para el ejemplo del equipo de fútbol:

```
Tipo País (Cadena)
descripción:
  "Nombres de todos los países"
definición:
  {"España", "Portugal", "Francia", ..., "Argentina", "Brasil", ...,
   "Japón", "Corea", ...}
Fin tipo

Tipo País comunitario (País)
descripción:
  "Nombres de los países de la Unión Europea"
definición:
  {"Alemania", "Austria", "Bélgica", "Dinamarca", "España",
   "Finlandia", "Francia", "Gran Bretaña", "Grecia", "Irlanda",
   "Italia", "Luxemburgo", "Países Bajos", "Portugal", "Suecia"}
Fin tipo

Tipo País nacional (País comunitario)
descripción:
  "Nombre de países de jugadores  considerados nacionales"
definición:
  {"España"}
Fin tipo

Tipo País extranjero (País)
descripción:
  "Nombres de países extranjeros no comunitarios"
definición:
   restando País comunitario
Fin tipo

Tipo Posición (Cadena)
descripción:
  "Posición natural que puede ocupar un jugador en un equipo de fútbol"
definición:
  {"Portero", "Lateral izquierdo", "Lateral derecho",
   "Defensa central", "Líbero", "Medio centro", "Interior izquierdo",
   "Interior derecho", "Extremo izquierdo", "Extremo derecho",
   "Media punta", "Delantero"}
```

```
Fin tipo

Tipo Posición portero (Posición)
descripción:
  "Posición que ocupa un portero"
definición:
  {"Portero"}
Fin tipo

Tipo Posición campo (Posición)
descripción:
  "Posición que ocupa un jugador de campo"
definición:
  restando Posición portero
Fin tipo

Tipo Estado (Cadena)
descripción:
  "Estado físico de un jugador"
definición:
  {"bien", "lesionado"}
Fin tipo

Tipo Valoración (Real)
descripción:
  "Valoración numérica entre 0 y 10 de una cualidad de un jugador
  (fuerza, velocidad, etc.)"
definición:
  [0, 10]
Fin tipo

Tipo Dorsal (Entero)
descripción:
  "Número de dorsal asignado a un jugador"
definición:
  [1, 25]
Fin tipo

Tipo Categoría (Cadena)
descripción:
  "Categoría en la que juega un equipo de fútbol"
definición:
  {"Primera", "Segunda", "Segunda B", "Tercera", "Regional"}
Fin tipo

Tipo Carácter (Cadena)
descripción:
  "Carácter de una táctica: ofensivo, defensivo o neutro"
definición:
  {"ofensivo", "defensivo", "neutro"}
Fin tipo

Tipo Día (Entero)
descripción:
  "Número de día dentro del mes, entre 1 y 31"
definición:
  [1, 31]
Fin tipo

Tipo Mes (Entero)
descripción:
  "Número de mes dentro del año, entre 1 y 12"
definición:
  [1, 12]
Fin tipo

Tipo Año (Entero)
descripción:
  "Número que representa un año. Debe ser mayor que 1800"
definición:
  [1800, infinito)
Fin tipo
```

Además de la representación descrita anteriormente, también se admite una representación condensada, que podrá utilizarse, por ejemplo, cuando simplemente se desee hacer una enumeración de los tipos de un modelo:

Tipo <nombre> (<tipo base>)

4.3.1.2 Instancias y Conceptos

Las **instancias de concepto** se representarán indicando su nombre y el concepto (o conceptos) del que deriva como sigue:

Instancia <nombre> (<conceptos>) "["<ti>, <tf>"]"
Fin instancia

Significado de los campos:

- <nombre>: nombre asignado a la instancia.
- <conceptos>: lista de los conceptos a los que pertenece la instancia.
- <ti>: instante inicial (creación) de la instancia. Es un número natural.
- <tf>: instante final (destrucción) de la instancia. Si no está definido (es decir, si la instancia no ha sido destruida) se representará como infinito.

A esta representación se añadirán elementos para indicar los valores de atributo asignados a la instancia concreta:

Ejemplo de **instancia** del concepto "Persona":

```
Instancia "Juan" (Persona) [0, infinito]
Fin instancia
```

En cuanto a los **conceptos**, su representación básica (nombre, descripción e invariante) será la siguiente:

Concepto <nombre>
[**descripción:**
<descripción>]
[**invariante**:
<invariante>]
Fin concepto

Significado de los campos:

- <nombre>: nombre del concepto
- <descripción>: descripción asignada al concepto. Puede no aparecer.
- <invariante>: invariante definida para el concepto. La invariante se representa usando el lenguaje de representación de condiciones lógicas (apartado 4.3.2). Puede no aparecer.

Más adelante se definirán elementos para representar información sobre atributos y clasificaciones de concepto.

Equipo de fútbol

Seguidamente se recogen los conceptos principales de este ejemplo:

```
Concepto Persona
descripción:
```

```
    "Una persona cualquiera"
Fin Concepto

Concepto Jugador
descripción:
  "Persona que juega en un equipo"
Fin Concepto

Concepto Equipo
descripción:
  "Un equipo de fúbol"
Fin Concepto

Concepto Alineación
descripción:
  "Una posible alineación de jugadores de un equipo para un partido"
Fin Concepto

Concepto Táctica
descripción:
  "Una táctica (por ejemplo 4-4-2) asignada a una o más alineaciones de
   equipos"
Fin Concepto

Concepto Puesto
descripción:
  "Definición de un puesto que puede ocupar un jugador en una
   alineación"
Fin Concepto

Concepto Fecha
descripción:
  "Una fecha definida mediante número de día, número de mes y año"
invariante:
  sea ?"Fecha" de C"Fecha" existe ?"d" de T"Día" existe ?"m" de T"Mes"
  existe ?"a" de T"Año"
   si
    (?"Fecha", ?"d") pertenece a AT"Fecha"."día" and
    (?"Fecha", ?"m") pertenece a AT"Fecha"."mes" and
    (?"Fecha", ?"a") pertenece a AT"Fecha"."año"
   entonces
    (?"m"=4 or ?"m"=6 or ?"m"=9 or ?"m"=11) and (?"d"<31) or
    (?"m"=2 and ?"a"%4=0 and (?"a"%100!=0 or ?"a"%400=0) and
    (?"d"<30) or
    (?"m"=2 and (?"a"%4!=0 or (?"a"%100=0 and ?"a"%400!=0)) and
    (?"d"<30)
   fin si
  fin condición
 fin condición fin condición fin condición
Fin Concepto
```

4.3.1.3 Atributos de concepto

Los atributos de concepto se representarán dentro del concepto al que pertenecen, formando parte de un bloque encabezado por la palabra reservada "atributos:".

Cada **atributo** tendrá la siguiente representación:

<nombre>:<tipo> [(<grado>:<límite>)] [= <valor omisión>] [<descripción>]

Significado de los campos:

- <nombre>: nombre del atributo.
- <tipos>: nombre del tipo asignado al atributo.
- <grado>: se representa como 0 para "opcional" y 1 para "obligatorio".
- <límite>: se representa como 1 para "único" y 'N' para "múltiple".

- <valor omisión>: valor por omisión asignado al atributo en ese concepto. Puede no aparecer.
- <descripción>: explicación del significado del atributo. Puede no aparecer.

Si el grado es obligatorio y el límite único (que es el caso más habitual), entonces no se representarán. En otras palabras, la combinación (1:1) no se representa.

Equipo de fútbol

Seguidamente se muestran los atributos de cada concepto:

```
Concepto Persona
descripción:
  "Una persona cualquiera"
atributos:
  nombre : Cadena "Nombre completo de la persona"
  país : País (1:N) "País(es) de la persona"
Fin Concepto

Concepto Jugador
descripción:
  "Persona que juega en un equipo"
atributos:
  posición : Posición "Posición natural del jugador en el campo"
  dorsal : Dorsal "Número de dorsal elegido por el jugador"
  fuerza : Valoración "Valoración de la fuerza del jugador"
  velocidad : Valoración "Valoración de la velocidad del jugador"
  resistencia : Valoración
     "Valoración de la resistencia física del jugador"
  estado : Estado = "bien" "Estado físico (lesionado o no)"
Fin Concepto

Concepto Equipo
descripción:
  "Un equipo de fúbol"
atributos:
  nombre : Cadena "Nombre completo del equipo"
  categoría : Categoría "Categoría en la que compite el equipo"
Fin Concepto

Concepto Alineación
descripción:
  "Una posible alineación de jugadores de un equipo para un partido"
atributos:
  nombre : Cadena "Nombre de la alineación"
Fin Concepto

Concepto Táctica
descripción:
  "Una táctica (por ejemplo 4-4-2) asignada a una o más alineaciones de
   equipos"
atributos:
  nombre : Cadena "Nombre de la táctica"
  carácter : Carácter = "neutro"
     "Carácter ofensivo, defensivo o neutro de la táctica"
Fin Concepto

Concepto Puesto
descripción:
  "Definición de un puesto que puede ocupar un jugador en una
   alineación"
atributos:
  titular : Booleano "Indica si el puesto es de titular o no"
  posición : Posición
     "Posición de jugador más adecuada para este puesto"
  afines : Posición (0:N) "Posiciones afines para este puesto"
Fin Concepto

Concepto Fecha
```

```
descripción:
  "Una fecha definida mediante número de día, número de mes y año"
atributos:
  día : Día "Número de día dentro del mes"
  mes : Mes "Número de mes dentro del año"
  año : Año "Número de año"
invariante:
  sea ?"Fecha" de C"Fecha" existe ?"d" de T"Día" existe ?"m" de T"Mes"
  existe ?"a" de T"Año"
   si
    (?"Fecha", ?"d") pertenece a AT"Fecha"."día" and
    (?"Fecha", ?"m") pertenece a AT"Fecha"."mes" and
    (?"Fecha", ?"a") pertenece a AT"Fecha"."año"
   entonces
    (?"m"=4 or ?"m"=6 or ?"m"=9 or ?"m"=11) and (?"d"<31) or
    (?"m"=2 and ?"a"%4=0 and (?"a"%100!=0 or ?"a"%400=0) and
    (?"d"<30) or
    (?"m"=2 and (?"a"%4!=0 or (?"a"%100=0 and ?"a"%400!=0)) and
    (?"d"<30)
   fin si
  fin condición
 fin condición fin condición fin condición
Fin Concepto
```

En cuanto a las **instancias de atributo**, se representarán dentro de la instancia de concepto correspondiente. Para ello se representarán dentro de un bloque "atributos:" de la siguiente forma:

<nombre> = <valor> "["<ti>, <tf>"]"

Significado de los campos:

- <nombre>: nombre del atributo cuya instancia se está definiendo.
- <valor>: valor asignado para esa instancia.
- <ti>: instante de creación del valor.
- <tf>: instante de destrucción del valor.

Si la instancia de concepto tiene más de un valor asignado, se representarán todos los valores (y sus correspondientes tiempos de existencia) encerrados entre llaves.

Si el atributo no tiene valor asignado para la instancia de concepto y hay un valor por omisión, se representará este valor entre paréntesis.

Si, por último, no hay valor asignado y no se ha definido valor por omisión, entonces no se pondrá nada después del signo igual.

Equipo de fútbol

Seguidamente se muestran algunos ejemplos de instancias de concepto con sus valores de atributo:

```
Instancia Loïc (Persona)
atributos:
  nombre = "Loïc Antonio Martínez Normand"
  país = {España, Francia}
Fin instancia

Instancia Casillas (Jugador) [0, infinito]
atributos:
  posición = "Portero" [1, infinito]
  dorsal = 1 [1, infinito]
  fuerza = 7.0 [1, infinito]
  velocidad = 8.0 [1, infinito]
```

```
    resistencia = 7.0 [1, infinito]
    estado = ("bien")
Fin instancia
```

4.3.1.4 Asociaciones y Tuplas

La parte básica de las **asociaciones** se representará como sigue:

Asociación <nombre>
[**descripción**:
  <descripción>]
origen:
  <roles>
destino:
  <rol>
tipo:
  <funcionalidad> (<completitud>)
[**invariante**:
  <invariante>]
Fin asociación

Significado de los campos:

- <nombre>: nombre de la asociación
- <descripción>: explicación del significado de la asociación. Puede no aparecer.
- <roles>: lista de los roles que forman el origen de la asociación.
- <rol>: rol que define el destino de la asociación.
- <funcionalidad>: valor de la funcionalidad de la asociación. Será: "función", "función inyectiva", "función sobreyectiva", "función biyectiva" o "no función".
- <completitud>: indica la completitud de la asociación: "completa" o "parcial".
- <invariante>: invariante de la asociación. Puede no aparecer.

Más adelante se definirá la representación de atributos y clasificaciones de asociación.

Cada uno de los **roles** se definirá en una línea de la forma siguiente:

<nombre>:<participante> (<grado>:<límite>)

Significado de estos campos:

- <nombre>: nombre del rol.
- <participante>: identificador del participante en ese rol. Si el participante es un tipo o un concepto se pondrá su nombre. Si el participante es una asociación, deberá aparecer el identificador completo de asociación (formado por los nombres de los participantes del origen de esa asociación seguido del nombre de la asociación).
- <grado>: grado del rol. Se representa igual que el grado de atributos.
- <límite>: límite del rol. Se representa igual que el límite de atributos.

En los roles de una asociación siempre deberán aparecer tanto el grado como el límite.

Equipo de fútbol

Seguidamente se recogen las asociaciones existentes:

```
Asociación Nacimiento
descripción:
  "Asociación entre una persona y su fecha de nacimiento"
origen:
  persona : Persona (1:1)
destino:
  fecha : Fecha (0:N)
tipo:
  función (completa)
Fin asociación

Asociación Pertenencia
descripción:
  "Asociación entre una persona, una fecha y el equipo al que pertenece
   esa persona a partir de esa fecha. La fecha podrá participar
   cualquier número de veces en la asociación. La persona podrá
   participar cualquier número de veces, siempre que sea en fechas
   distintas (esto último se debería definir en la invariante de la
   asociación). Un equipo deberá aparecer al menos una vez en la
   asociación (sino no es equipo)."
origen:
  persona : Persona (0:N)
  fecha comienzo : Fecha (0:N)
destino:
  equipo : Equipo (1:N)
tipo:
  función sobreyectiva (parcial)
Fin asociación

Asociación Fin de Pertenencia
descripción:
  "Asociación entre una pertenencia de una persona a un equipo y la
   fecha en la que termina esa pertenencia"
origen:
  pertenencia : (Persona).Pertenencia (0:1)
destino:
  fecha fin: Fecha (0:N)
tipo:
  función (parcial)
Fin asociación

Asociación Convocatoria
descripción:
  "Asociación entre un equipo, una fecha y su alineación para ese día"
origen:
  equipo : Equipo (1:N)
  fecha : Fecha (0:N)
destino:
  alineación : Alineación (1:1)
tipo:
  función biyectiva (parcial)
Fin asociación

Asociación Táctica prevista
descripción:
  "Relaciona una alineación con su táctica"
origen:
  alineación : Alineación (1:1)
destino:
  táctica elegida : Táctica (0:N)
tipo:
  función (completa)
Fin asociación

Asociación Puestos
descripción:
  "Puestos definidos por una táctica"
origen:
  táctica : Táctica (1:N)
destino:
```

```
    puesto elegido : Puesto (1:N)
  tipo:
    no función (completa)
  invariante:
    // Para cada táctica habrá 11 puestos de titulares (con 1 portero) y
    // 5 suplentes (con 1 portero)
    ...
  Fin asociación

  Asociación Asignación
  descripción:
    "Asignación de un jugador a un puesto dentro de una alineación de un
     equipo"
  origen:
    alineación : Alineación (1:N)
    puesto asignado : Puesto (0:N)
  destino:
    jugador convocado : Jugador (0:N)
  tipo:
    función (parcial)
  invariante:
    // El jugador deberá pertenecer al equipo asociado a la alineación y
    // un jugador sólo podrá aparecer una vez en cada alineación y un
    // puesto sólo podrá aparecer una vez en cada alineación y cada
    // puesto deberá pertenecer a la táctica prevista en la alineación y
    // no puede haber más de 3 extranjeros titulares
    ...
  Fin asociación

  Asociación Bisiesto
  descripción:
    "Asociación entre un año y un booleano que indica si el año es
     bisiesto"
  origen:
    año: Año (1:1)
  destino:
    bisiesto: Booleano (1:N)
  tipo:
    función sobreyectiva (completa)
  invariante:
    // El año es bisiesto si es múltiplo de 4 pero no múltiplo de 100,
    // excepto los múltiplos de 400 que siempre son bisiestos
  Fin asociación
```

Las **instancias de asociación** (tuplas) se representarán indicando la asociación (o asociaciones) a la que pertenecen, más la asignación de participantes a cada uno de sus roles (más adelante se añadirá la representación de valores de atributo):

Tupla (<asociaciones>) "["<ti>, <tf>"]"
participantes:
  <participantes>
Fin tupla

Significado de los campos:

- <asociaciones>: identificador de la asociación (o asociaciones) a la que pertenece la tupla.
- <ti>: instante de creación de la tupla.
- <tf>: instante de destrucción de la tupla.
- <participantes>: cada uno de los participantes de la tupla. Para cada participante se representará el nombre del rol y la identificación de la instancia del participante

(valor, nombre de instancia de concepto o tupla expresada de forma abreviada, con los identificadores de sus participantes entre paréntesis):

<rol> = <instancia participante>

Ejemplos de **tuplas**:

```
Tupla (Persona).Nacimiento [10, infinito]
participantes:
   persona = Loïc
   fecha = 16.10.1969
Fin instancia

Tupla (Persona, Fecha).Pertenencia [32, infinito]
participantes:
   persona = Rivaldo
   fecha comienzo = 1.8.1998
   equipo = Barcelona
Fin tupla


Tupla (Persona, Fecha).Pertenencia [33, 55]
participantes:
   persona = Figo
   fecha comienzo = 1.8.2000
   equipo = Real Madrid
Fin tupla

Tupla ((Persona).Pertenencia).Fin de Pertenencia [65, infinito]
participantes:
   pertenencia = (Rivaldo, 1.8.1998, Barcelona)
   fecha fin = 1.8.2000
Fin tupla

Tupla (Equipo, Fecha).Convocatoria [32, infinito]
participantes:
   equipo = Real Madrid
   fecha = 7.9.2002
   alineación = RM Habitual
Fin tupla

Tupla (Alineación).Táctica prevista [21, infinito]
participantes:
   alineación = RM Habitual
   táctica elegida = 4-4-2
Fin tupla

Tupla (Táctica).Puestos [61, infinito]
participantes:
   táctica = 4-4-2
   puesto elegido = Portero titular
Fin tupla

Tupla (Alineación, Puesto).Asignación [75, infinito]
participantes:
   alineación = RM Habitual
   puesto asignado = Portero titular
   jugador convocado = Casillas
Fin tupla
```

4.3.1.5 Atributos de Asociación

Los atributos de asociación se representarán dentro de la asociación a la que pertenecen, formando parte de un bloque encabezado por la palabra reservada “atributos:”.

Cada **atributo de asociación** tendrá la siguiente representación:

<nombre>:<tipo> [(<grado>:<límite>)] [= <valor omisión>] [<descripción>]

Significado de los campos:

- <nombre>: nombre del atributo.
- <tipos>: nombre del tipo asignado al atributo.
- <grado>: se representa como 0 para "opcional" y 1 para "obligatorio".
- <límite>: se representa como 1 para "único" y 'N' para "múltiple".
- <valor omisión>: valor por omisión asignado al atributo en ese concepto. Puede no aparecer.
- <descripción>: explicación del significado del atributo. Puede no aparecer.

Si el grado es obligatorio y el límite único (que es el caso más habitual), entonces no se representarán. En otras palabras, la combinación (1:1) no se representa.

Equipo de fútbol

Seguidamente se recoge la asociación "Pertenencia" que tiene un atributo:

```
Asociación Pertenencia
descripción:
  "Asociación entre una persona, una fecha y el equipo al que pertenece
   esa persona a partir de esa fecha. La fecha podrá participar
   cualquier número de veces en la asociación. La persona podrá
   participar cualquier número de veces, siempre que sea en fechas
   distintas (esto último se debería definir en la invariante de la
   asociación). Un equipo deberá aparecer al menos una vez en la
   asociación (sino no es equipo)."
origen:
  persona : Persona (0:N)
  fecha comienzo : Fecha (0:N)
destino:
  equipo : Equipo (1:N)
atributos:
  es socio : Booleano = falso
     "Indica si la persona es socio del equipo"
tipo:
  función sobreyectiva (parcial)
Fin asociación
```

En cuanto a las **instancias de atributo de asociación**, se representarán dentro de la tupla correspondiente. Para ello se representarán dentro de un bloque "atributos:" de la siguiente forma:

<nombre> = <valor> "["<ti>, <tf>"]"

Significado de los campos:

- <nombre>: nombre del atributo cuya instancia se está definiendo.
- <valor>: valor asignado para esa instancia.
- <ti>: instante de creación del valor.
- <tf>: instante de destrucción del valor.

Si la tupla tiene más de un valor asignado, se representarán todos los valores (y sus correspondientes tiempos de existencia) encerrados entre llaves.

Si el atributo no tiene valor asignado para la tupla y hay un valor por omisión, se representará este valor entre paréntesis.

Si, por último, no hay valor asignado y no se ha definido valor por omisión, entonces no se pondrá nada después del signo igual.

Equipo de fútbol

Seguidamente se muestran algunos ejemplos de instancias de asociación con sus valores de atributo:

```
Tupla (Persona, Fecha).Pertenencia [33, infinito]
participantes:
   persona = Figo
   fecha comienzo = 1.8.2000
   equipo = Real Madrid
atributos:
   es socio = falso [33, infinito]
Fin tupla

Tupla (Persona, Fecha).Pertenencia [26, infinito]
participantes:
   persona = Raúl
   fecha comienzo = 1.8.1995
   equipo = Real Madrid
atributos:
   es socio = cierto [27, infinito]
Fin tupla
```

4.3.1.6 Clasificaciones de Concepto

Las clasificaciones de concepto se van a representar en el lenguaje mediante un elemento nuevo (la clasificación), pero también se reflejará información sobre las clasificaciones en los conceptos correspondientes.

En primer lugar se verá cómo se representa una **clasificación**:

Clasificación de concepto <superconcepto> [(<nombre>)]
[**descripción:**
<descripción>]
subconceptos:
<subconceptos>
propiedades:
(<disjunta>, <completa>)
Fin clasificación

Seguidamente se describe cada uno de los campos:

- <superconcepto>: nombre del superconcepto de la clasificación
- <nombre>: nombre de la clasificación entre paréntesis (si está definido)
- <descripción>: descripción de la clasificación. Puede no aparecer.
- <subconceptos>: definición de los subconceptos. La forma de representar los subconceptos se verá más abajo.
- <disjunta>: tendrá el valor disjunta si la clasificación es disjunta o no disjunta si no lo es.
- <completa>: tendrá el valor completa si la clasificación es completa o incompleta si no lo es.

Queda pendiente definir cómo se representa cada uno de los **subconceptos**:

subconcepto: <nombre subconcepto>
[especializaciones:

<especializaciones>]

Es decir, para cada subconcepto se representará el nombre del concepto y una lista de sus especializaciones (si tiene alguna). En cuanto a las **especializaciones**, éstas pueden ser de seis tipos:

- Grado de asociación: se definirá mediante el identificador de la asociación (formado uniendo el nombre de los orígenes más el nombre de la asociación) más el nombre del rol cuyo grado se vuelve más específico:

  grado de asociación <asociación> <rol>

- *Límite de asociación:* se definirá mediante el identificador de la asociación más el nombre del rol cuyo límite se vuelve más específico:

  límite de asociación <asociación> <rol>

- *grado de atributo:* se definirá mediante el nombre del atributo:

  grado de atributo <atributo>

- *límite de atributo:* se definirá mediante el nombre del atributo:

  límite de atributo <atributo>

- *tipo de atributo:* se definirá mediante el nombre del atributo y el nuevo tipo:

  tipo de atributo <atributo> : <tipo>

- *valor por omisión de atributo:* se definirá mediante el nombre del atributo y el nuevo valor por omisión:

  valor por omisión <atributo> = <valor>

Equipo de fútbol

Seguidamente se recogen ejemplos de clasificaciones:

```
Clasificación de concepto Persona (por ocupación)
descripción:
  "Clasificación de personas en función de su ocupación en un equipo"
subconceptos:
  subconcepto Jugador
  subconcepto Directivo
  subconcepto Técnico
propiedades:
  (no disjunta,incompleta)
Fin clasificación

Clasificación de concepto Jugador (por país)
descripción:
  "Clasificación de jugadores por país de origen"
subconceptos:
  subconcepto Comunitario
  especializaciones:
    tipo de atributo país : País comunitario
  subconcepto Nacional
  especializaciones:
    tipo de atributo país : País nacional
  subconcepto Extranjero
  especializaciones:
    tipo de atributo país : País extranjero
propiedades:
  (disjunta,completa)
Fin clasificación
```

```
Clasificación de concepto Jugador (por posición)
descripción:
  "Clasificación de jugador en función de su posición en el campo"
subconceptos:
  subconcepto Portero
  especializaciones:
    tipo de atributo posición : Posición portero
  subconcepto Jugador campo
  especializaciones:
    tipo de atributo posición : Posición campo
propiedades:
  (disjunta,completa)
Fin clasificación

Clasificación de concepto Puesto (por titularidad)
descripción:
  "Tipos de puestos: titular y suplente"
subconceptos:
  subconcepto Titular
  especializaciones:
    valor por omisión titular = verdad
  subconcepto Suplente
  especializaciones:
    valor por omisión titular = falso
propiedades:
  (disjunta,completa)
Fin clasificación
```

Una vez vista la representación de clasificaciones, queda por describir cómo quedan reflejadas **estas clasificaciones en la definición de un concepto**.

Se va a añadir una sección más a la representación de un concepto, "padres", que aparecerá entre "descripción" y "atributos". Esta sección sólo aparecerá si el concepto es subconcepto en alguna clasificación.

En esta sección se mostrarán cada uno de los padres del concepto actual indicándose el nombre de la clasificación (si existe):

<nombre superconcepto> [(<nombre clasificación>)]

Conviene indicar que la sección "atributos" únicamente contendrá los atributos definidos en ese concepto, es decir, no se reflejarán los atributos heredados.

Equipo de fútbol

Seguidamente se muestran aquellos conceptos implicados en clasificaciones (algunos todavía no han sido definidos):

```
Concepto Jugador
descripción:
  "Persona que juega en un equipo"
padres:
  Persona (por ocupación)
atributos:
  posición : Posición "Posición natural del jugador en el campo"
  dorsal : Dorsal "Número de dorsal elegido por el jugador"
  fuerza : Valoración "Valoración de la fuerza del jugador"
  velocidad : Valoración "Valoración de la velocidad del jugador"
  resistencia : Valoración "Valoración de resistencia del jugador"
  estado : Estado = "bien" "Estado físico (lesionado o no)"
Fin Concepto

Concepto Directivo
descripción:
  "Una persona que desempeña labores de directivo en un equipo de
   fútbol"
```

```
padres:
  Persona (por ocupación)
Fin Concepto

Concepto Técnico
descripción:
  "Una persona que ejerce labores de técnico en un equipo"
padres:
  Persona (por ocupación)
Fin Concepto

Concepto Nacional
descripción:
  "Jugador nacido en España"
padres:
  Jugador (por país)
Fin Concepto

Concepto Comunitario
descripción:
  "Un jugador nacido en país comunitario"
padres:
  Jugador (por país)
Fin Concepto

Concepto Extranjero
descripción:
  "Jugador nacido en país no comunitario"
padres:
  Jugador (por país)
Fin Concepto

Concepto Portero
descripción:
  "Un jugador que juega en la posición de portero"
padres:
  Jugador (por posición)
atributos:
  reflejos : Valoración "Valoración de los reflejos del portero"
  pie : Valoración "Valoración del juego con el pie"
  alto : Valoración "Valoración de las paradas por alto"
  bajo : Valoración "Valoración de las paradas por bajo"
  salidas : Valoración "Valoración de la calidad de las salidas"
Fin Concepto

Concepto Jugador campo
descripción:
  "Jugador que no juega de portero"
padres:
  Jugador (por posición)
atributos:
  robos : Valoración "Valoración de la capacidad para robar balones"
  centros : Valoración
     "Valoración de la capacidad para centrar el balón"
  regate : Valoración "Valoración de la capacidad para regatear"
  remate pie : Valoración
     "Valoración de la capacidad para rematar con el pie"
  remate cabeza : Valoración
     "Valoración de la capacidad para rematar con la cabeza"
Fin Concepto

Concepto Titular
descripción:
  "Puesto de titular"
padres:
  Puesto (por titularidad)
Fin Concepto

Concepto Suplente
descripción:
  "Puesto de suplente"
```

```
padres:
  Puesto (por titularidad)
Fin Concepto
```

Por último, en las instancias de concepto se tendrán que representar los valores de todos los atributos, tanto directos como heredados.

4.3.1.7 Clasificaciones de Asociación

Las clasificaciones de asociación se van a representar en el lenguaje mediante un elemento nuevo (la clasificación de asociación), pero también se reflejará información sobre las clasificaciones en las asociaciones correspondientes

En primer lugar se verá cómo se representa una **clasificación de asociación**:

Clasificación de asociación <superasociación> [(<nombre>)]
[**descripción:**
<descripción>]
subasociaciones:
<subasociaciones>
propiedades:
(<disjunta>, <completa>)
Fin clasificación

Seguidamente se describe cada uno de los campos:

- <superasociación>: identificador de la superasociación de la clasificación (formado por los identificadotes de los participantes del origen más el nombre de la asociación)
- <nombre>: nombre de la clasificación entre paréntesis (si está definido)
- <descripción>: descripción de la clasificación. Puede no aparecer.
- <subasociaciones>: definición de las subasociaciones (ver más abajo).
- <disjunta>: tendrá el valor *disjunta* si la clasificación es disjunta o *no disjunta* si no lo es.
- <completa>: tendrá el valor *completa* si la clasificación es completa o *incompleta* si no lo es.

Queda pendiente definir cómo se representa cada uno de las **subasociaciones**:

subasociación: <identificador subasociación>
[especializaciones:
<especializaciones>]

Es decir, para cada subasociación se representará el identificador de la asociación y una lista de sus especializaciones (si tiene alguna). En cuanto a las especializaciones, éstas se representan igual que en el caso de clasificaciones de concepto:

- *grado de asociación*:

grado de asociación <asociación> <rol>

- *límite de asociación*:

límite de asociación <asociación> <rol>

- *grado de atributo*:

grado de atributo <atributo>

- *límite de atributo*:

límite de atributo <atributo>

- *tipo de atributo*:

tipo de atributo <atributo> : <tipo>

- *valor por omisión de atributo*:

valor por omisión <atributo> = <valor>

Equipo de fútbol

Seguidamente se recogen ejemplos de clasificaciones de asociación:

```
Clasificación de asociación (Persona, Fecha).Pertenencia (por papel)
descripción:
  "Clasificación de la pertenencia de una persona a un equipo en una
   fecha en función del papel que desempeña: jugador, entrenador,
   presidente"
subasociaciones:
  subasociación (Jugador, Fecha).Fichaje
  subasociación (Técnico, Fecha).Entrenador
  subasociación (Directivo, Fecha).Presidente
propiedades:
  (no disjunta,incompleta)
Fin clasificación

Clasificación de asociación (Táctica).Puestos (por titularidad)
descripción:
  "Clasificación de la asociación entre una táctica y sus puestos, en
   función de si son titulares o suplentes".
subasociaciones:
  subasociación (Táctica).Titulares
  subasociación (Táctica).Suplentes
propiedades:
  (disjunta,completa)
Fin clasificación
```

Una vez vista la representación de clasificaciones, queda por describir cómo quedan reflejadas estas **clasificaciones en la definición de una asociación**.

Se va a añadir una sección más a la representación de una asociación, "padres", que aparecerá entre "descripción" y "origen". Esta sección sólo aparecerá si la asociación es subasociación en alguna clasificación.

En esta sección se listarán cada uno de los padres de la asociación actual indicándose el nombre de la clasificación (si existe):

<identificador superasociación> [(<nombre clasificación>)]

Conviene indicar que la sección "atributos" únicamente contendrá los atributos definidos en esa asociación, es decir, no se reflejarán los atributos heredados.

Equipo de fútbol

Seguidamente se muestran aquellas asociaciones implicadas en clasificaciones (que todavía no han sido definidas):

```
Asociación Fichaje
descripción:
```

```
    "Asociación entre un jugador, una fecha y el equipo en el que juega
     ese jugador a partir de esa fecha. La fecha podrá participar
     cualquier número de veces en la asociación. El jugador deberá
     participar al menos una vez y podrá participar cualquier número de
     veces, siempre que sea en fechas distintas (esto último se debería
     definir en la invariante de la asociación). Un equipo deberá
     aparecer al menos una vez en la asociación (sino no es equipo)."
padres:
   (Persona,Fecha).Pertenencia (por papel)
origen:
   jugador : Jugador (1:N)
   fecha comienzo : Fecha (0:N)
destino:
   equipo : Equipo (1:N)
atributos:
   ficha : Entero "Importe de la ficha del jugador"
   rescisión: Entero "Importe de la cláusula de rescisión"
tipo:
   función (parcial)
Fin asociación

Asociación Entrenador
descripción:
   "Asociación entre un técnico, una fecha y el equipo en el que ese
    técnico es entrenador a partir de esa fecha. La fecha podrá
    participar cualquier número de veces en la asociación. El técnico
    podrá participar cualquier número de veces, siempre que sea en
    fechas distintas (esto último se debería definir en la invariante de
    la asociación). Un equipo deberá aparecer al menos una vez en la
    asociación (sino no es equipo)."
padres:
   (Persona,Fecha).Pertenencia (por papel)
origen:
   entrenador : Técnico (0:N)
   fecha comienzo : Fecha (0:N)
destino:
   equipo : Equipo (1:N)
atributos:
   sueldo: Entero "Sueldo anual del entrenador"
   objetivo: Cadena "Descripción del objetivo del entrenador"
tipo:
   función sobreyectiva (parcial)
Fin asociación

Asociación Presidente
descripción:
   "Asociación entre un directivo, una fecha y el equipo en el que ese
    directivo es presidente a partir de esa fecha. La fecha podrá
    participar cualquier número de veces en la asociación. El directivo
    podrá participar cualquier número de veces, siempre que sea en
    fechas distintas (esto último se debería definir en la invariante de
    la asociación). Un equipo deberá aparecer al menos una vez en la
    asociación (sino no es equipo)."
padres:
   (Persona,Fecha).Pertenencia (por papel)
origen:
   presidente : Directivo (0:N)
   fecha comienzo : Fecha (0:N)
destino:
   equipo : Equipo (1:N)
atributos:
   propietario: Booleano "Indica si el presidente es, además, dueño"
tipo:
   función sobreyectiva (parcial)
Fin asociación

Asociación Titulares
descripción:
   "Puestos de titulares definidos por una táctica"
```

```
padres:
  (Táctica).Puestos (por titularidad)
origen:
  táctica : Táctica (1:N)
destino:
  puesto elegido : Titular (1:N)
tipo:
  no función (completa)
invariante:
  // Para cada táctica habrá 11 puestos de titulares (con 1 portero)
  ...
Fin asociación

Asociación Suplentes
descripción:
  "Puestos de suplentes definidos por una táctica"
padres:
  (Táctica).Puestos (por titularidad)
origen:
  táctica : Táctica (1:N)
destino:
  puesto elegido : Suplente (1:N)
invariante:
  // Para cada táctica habrá 5 puestos de suplentes (con 1 portero)
  ...
tipo:
  no función (completa)
Fin asociación
```

Por último, en las instancias de asociación se tendrán que representar los valores de todos los atributos, tanto directos como heredados.

4.3.2 Lenguaje para el Modelo de Comportamiento

4.3.2.1 Lenguaje para Tareas

La representación de una **tarea** es la siguiente:

Tarea <nombre>
[**descripción:**
  <descripción>]
métodos:
  <métodos>
entrada:
  <entrada>
precondición:
  <precondición>
[**salida**:
  <salida>]
postcondición:
  <postcondición>
Fin tarea

Significado de los campos:

- <nombre>: nombre de la tarea.
- <descripción>: descripción de la tarea.
- <métodos>: lista de los métodos que resuelven esta tarea.

- <entrada>: lista de las variables que forman parte de la entrada de la tarea. Las variables estarán expresadas en el lenguaje de declaración de variables que se describe como parte del lenguaje de las condiciones.
- <precondición>: condición lógica unaria que representa la precondición de la tarea.
- <salida>: lista de las variables que forman parte de la salida de la tarea. Puede no aparecer.
- <postcondición>: condición lógica binaria que representa la postcondición de la tarea.

Equipo de fútbol

Mejorar la fuerza de un jugador en un incremento determinado, proporcionando el jugador y el incremento de fuerza deseado.

```
Tarea Mejorar fuerza jugador
  descripción:
    "Mejorar la fuerza de un jugador en un incremento determinado"
  métodos:
    Partiendo de táctica
  entrada:
    sea jug de C"Jugador"
    sea "inc" de T"Real"
  precondición:
    existe f1 de T"Valoración"
      (jug, f1) pertenece a AT"Jugador"."fuerza" and
      f1 < 10 and inc > 0 and f1 + inc <= 10
    fin condición
  salida:
  postcondición:
    existe f1 de T"Valoración"
      (jug, f1) pertenece a AT"Jugador"."fuerza" en [1] and
      not (jug, f1) pertenece a AT"Jugador"."fuerza" en [2] and
      (jug, f1 + inc) pertenece a AT"Jugador"."fuerza" en [2]
    fin condición
Fin Tarea
```

Crear un jugador de campo desde cero, proporcionando todos sus datos (nombre completo, país de origen, fecha de nacimiento, posición, dorsal, fuerza, velocidad, resistencia, nombre del equipo al que pertenece, fecha de comienzo del fichaje, importe de la ficha, cláusula de rescisión, valoración de robos, centros, regate, remate con el pie y remate con la cabeza).

```
Tarea Crear jugador campo
  descripción:
    "Crea un nuevo jugador de campo a partir de todos sus datos"
  métodos:
    ...
  entrada:
    sea nombre de T"Cadena" sea país de T"País" sea dn de T"Día"
    sea mn de T"Mes" sea an de T"Año" sea pos de T"Posición"
    sea dorsal de T"Dorsal" sea fuerza de T"Valoración"
    sea vel de T"Valoración" sea res de T"Valoración"
    sea nequipo de T"CAD" sea df de T"Día" sea mf de T"Mes"
    sea af de T"Año" sea ficha de T"NUM" sea rescisión de T"Entero"
    sea robos de Valoración sea centros de T"Valoración"
    sea regate de T"Valoración" sea rpie de T"Valoración"
    sea rcabeza de T"Valoración"
  precondición:
    existe equipo de C"Equipo"
      pos pertenece a T"Posición campo" and
      (equipo, nequipo) pertenece a AT"Equipo"."nombre" and
      para todo jug de C"Jugador"
       not (jug, nombre) pertenece a AT"Persona"."nombre" and
```

```
       si
        existe f de C"Fecha"
         (jug,f,equipo) pertenece a AS("Jugador","Fecha")."Fichaje"
        fin condición
       entonces
        not (jug, dorsal) pertenece a AT"Jugador"."dorsal" and
       fin si
      fin condición
    fin condición
  salida:
    sea jug de C"Jugador campo" sea fn de C"Fecha" sea ff de C"Fecha"
  postcondición:
    existe equipo de C"Equipo"
     (equipo, nequipo) pertenece a AT"Equipo"."nombre" en [2] and
     (jug,nombre) pertenece a AT"Persona"."nombre" en [2] and
     (jug,país) pertenece a AT"Persona"."país" en [2] and
     si
      país pertenece a T"País nacional"
     entonces
      jug pertenece a C"Nacional" en [2]
     fin si and
     si
      país pertenece a T"País comunitario"
     entonces
      jug pertenece a C"Comunitario" en [2]
     fin si and
     si
      país pertenece a T"País extranjero"
     entonces
      jug pertenece a C"Extranjero" en [2]
     fin si and
     (fn,dn) pertenece a AT"Fecha"."día" en [2] and
     (fn,mn) pertenece a AT"Fecha"."mes" en [2] and
     (fn,an) pertenece a AT"Fecha"."año" en [2] and
     (jug,fn) pertenece a AS("Persona")."Nacimiento" en [2] and
     (jug,pos) pertenece a AT"Jugador"."posición" en [2] and
     (jug,dorsal) pertenece a AT"Jugador"."dorsal" en [2] and
     (jug,fuerza) pertenece a AT"Jugador"."fuerza" en [2] and
     (jug,vel) pertenece a AT"Jugador"."velocidad" en [2] and
     (jug,res) pertenece a AT"Jugador"."resistencia" en [2] and
     (ff,df) pertenece a AT"Fecha"."día" en [2] and
     (ff,mf) pertenece a AT"Fecha"."mes" en [2] and
     (ff,af) pertenece a AT"Fecha"."año" en [2] and
     (jug,ff,equipo) pertenece a AS("Jugador,"Fecha")."Fichaje" en [2] and
     ((jug,ff,equipo),ficha) pertenece a
       ATS("Jugador, "Fecha")."Fichaje"."ficha" en [2] and
     ((jug,ff,equipo),rescisión) pertenece a
       ATS("Jugador, "Fecha")."Fichaje"."rescisión" en [2] and
     (jug,robos) pertenece a AT"Jugador campo"."robos" en [2] and
     (jug,centros) pertenece a AT"Jugador campo"."centros" en [2] and
     (jug,regate) pertenece a AT"Jugador campo"."regate" en [2] and
     (jug,rpie) pertenece a AT"Jugador campo"."remate pie" en [2] and
     (jug,rcabeza) pertenece a AT"Jugador campo"."remate cabeza" en [2]
    fin condición
Fin Tarea
```

Crear la alineación de un equipo para un partido concreto.

```
Tarea Crear alineación de equipo

  descripción:
    "Crear la alineación de un equipo para un partido concreto"
  métodos:
    ...
  entrada:
    sea eq de C"Equipo" sea da de T"Día" sea ma de T"Mes"
    sea aa de T"Año"
```

```
  precondición:
    para todo al de C"Alineación"
     para todo f de C"Fecha"
      si
       (eq,f,al) pertenece a AS("Equipo","Fecha")."Convocatoria"
      entonces
       not (f,da) pertenece a AT"Fecha"."día" or
       not (f,ma) pertenece a AT"Fecha"."mes" or
       not (f,aa) pertenece a AT"Fecha"."año"
      fin si
     fin condición
    fin condición
  salida:
    sea aln de C"Alineación" sea t de C"Táctica" sea fa de C"Fecha"
  postcondición:
    (fa,da) pertenece a AT"Fecha"."día" en [2] and
    (fa,ma) pertenece a AT"Fecha"."mes" en [2] and
    (fa,aa) pertenece a AT"Fecha"."año" en [2] and
    (eq,fa,aln) pertenece a AS("Equipo","Fecha")."Convocatoria" en[2] and
    (aln,t) pertenece a AS("Alineación")."Táctica prevista" en [2] and
    para todo puesto de C"Puesto"
     si
      (t,puesto) pertenece a AS("Táctica")."Puestos" en [2]
     entonces
      existe jug de C"Jugador"
       (aln, puesto, jug) pertenece a
          AS("Alineación","Puesto")."Asignación" en [2] and
       existe pos de T"Jugador"
        (jug,pos) pertenece a AT"Jugador"."posición" en [2] and
        ((puesto,pos") pertenece a AT"Puesto"."posición" en [2] or
         (puesto",pos") pertenece a AT"Puesto"."afines") en [2]
       fin condición
      fin condición
     fin si
    fin condición
Fin Tarea
```

4.3.2.2 Lenguaje para Métodos de Tarea

La representación de un **método** de tarea es la siguiente:

Método <nombre>
[**descripción:**
<descripción>]
tarea:
<tarea>
[**subtareas:**
<subtareas>]
control:
<control>
Fin método

Significado de los campos:

- <nombre>: nombre del método.
- <descripción>: descripción del método.
- <tarea>: nombre de la tarea principal del método.
- <subtareas>: lista de los nombres de las subtareas del método. Puede no aparecer.
- <control>: especificación de control del método, siguiendo el lenguaje descrito en el apartado siguiente.

Equipo de fútbol

Se muestra a continuación un **método para resolver la tarea "Crear alineación equipo"**. El método consiste en elegir en primer lugar una táctica y después asignar los mejores jugadores posibles a los puestos de esa táctica.

```
Método Partiendo de táctica
  descripción:
    "Crear una alineación eligiendo primero una táctica y luego
     asignando jugadores"
  tarea:
    Crear alineación equipo
  subtareas:
    Elegir táctica
    Asignar jugadores
  control:
    resolver "Elegir táctica" con (eq = eq) resultado (t=t)
    resolver "Asignar jugadores" con (eq=eq, t=t, da=da, ma=ma, aa=aa)
      resultado (aln=aln, fa=fa)
    fin control
Fin Método
```

4.3.2.3 Lenguaje para control de métodos

Una secuencia de control se representará directamente escribiendo cada uno de sus nodos de control. Seguidamente se va a definir el lenguaje que permite representar cada uno de esos nodos:

- **Nodo fin** (NF):

  fin control

- **Nodo de declaración de variable** (NV): se declarará la variable usando el lenguaje descrito en las condiciones lógicas.

  <variable>

- **Nodo de resolución de tarea** (NT):

  resolver <tarea> **con** <entrada> **resultado** <salida>

- **Nodo de ejecución de operador** (NO). Se usarán los nombres de operador de la tabla 4.1. El resultado puede no aparecer

  operador <operador> **con** <entrada> [**resultado** <salida>]

- **Nodo de condición simple** (NCS).

  si
  <condición>
  entonces
  <control cierto>
  fin si

- **Nodo de condición doble** (NCD).

  si
  <condición>
  entonces
  <control cierto>
  si no
  <control falso>
  fin si

- **Nodo de condición múltiple** (NCM).

  seleccionar
  <expresión>
  caso <valor 1>
  <control valor 1>
  caso <valor 2>
  <control valor 2>
  ...
  caso <valor n>
  <control valor n>
  en otro caso
  <control falso>
  fin seleccionar

- **Nodo de bucle mientras** (NBM).

  mientras
  <condición>
  hacer
  <control bucle>
  fin mientras

- **Nodo de bucle repetir** (NBR).

  repetir
  <control bucle>
  hasta
  <condición>
  fin repetir

- **Nodo de bucle para** (NBF).

  para <variable> **desde** <inferior> **hasta** <superior> **por** <inc>
  <control bucle>
  fin para

- **Nodo de bucle para todo** (NBFA).

  para todo <variable>
  <control bucle>
  fin para todo

- **Nodo de caminos paralelos** (NP).

  paralelo
  camino
  <bloque>
  camino
  <bloque>
  ...
  hasta <parada>

- **Nodo de espera a evento** (NEV).

  cuando
  <condición>
  hacer

<control evento>
fin evento

- **Nodo de control heurístico** (NH).

heurística
<tarea>
<expresión>
caso <valor 1>
<control valor 1>
caso <valor 2>
<control valor 2>
…
caso <valor n>
<control valor n>
en otro caso
<control falso>
fin heurística

4.3.3 Lenguaje para las Condiciones Lógicas

Dado que las condiciones lógicas se van a utilizar tanto en las invariantes como en el modelo de comportamiento, es conveniente definir un lenguaje que permita representarlas de forma más cómoda que siguiendo la definición formal. En este apartado se va a definir la gramática de dicho lenguaje, expresada en formato EBNF.

En dicha gramática se van a utilizar mayúsculas para representar elementos no terminales y comillas dobles para representar elementos terminales.

CND es una **condición lógica**, que puede ser:

- Una variable (cuantificada o no) seguida de una condición lógica.
- La conjunción de dos condiciones lógicas.
- La disyunción de dos condiciones lógicas.
- La implicación de dos condiciones lógicas.
- La negación de una condición lógica.
- Una condición lógica entre paréntesis.
- Una fórmula.

CND ← *VAR CND "fin condición" | CND "and" CND | CND "or" CND | "si" CND "entonces" CND "fin si" | "not" CND | "("CND")" | FORM*

VAR es una representación de **variable**, que puede estar cuantificada o no. Una variable se representa mediante un cuantificador, el identificador de la variable y el conjunto al que pertenece (concepto o tipo).

VAR ← *CUANT IDVAR TPV "de" CJTO*

CUANT representa un **cuantificador** de variable que puede ser "para todo" (∀) "existe" (∃), "existe uno" ($\exists_1$) o bien nada (para el caso de variables no cuantificadas). Para representar "nada" se utiliza "sea"

CUANT ← *"para todo" | "existe" | "existe uno" | "sea"*

IDVAR es un **identificador de variable** que se forma uniendo "?" a una cadena

IDVAR ← *"?"cad*

cad (símbolo terminal) representa una **cadena de caracteres** que estará encerrada entre comillas si tiene más de una palabra.

Ejemplos de identificadores de variables:

```
?persona, ?nombre
```

TPV representa un tipo de variable: "simple" o "conjunto". Si es simple no se pone nada.

TPV ← | *"conjunto"*

CJTO representa el **conjunto** en el que toma valores una variable. Ese conjunto puede ser un tipo, un concepto o bien una asociación.

CJTO ← *TIPO* | *CONC* | *ASOC*

CONC representa a un **identificador de concepto**: "C" más una cadena que representa el nombre de un concepto o bien una variable:

CONC ← *"C"cad* | *IDVAR*

Equipo de fútbol

Ejemplos de identificadores de conceptos:

```
C"Persona", C"Jugador", C"Fecha", C"Táctica", C"Alineación", ...
```

TIPO representa un **tipo**: "T" más una cadena representando el nombre del tipo o bien una variable.

TIPO ← *"T"cad* | *IDVAR*

Equipo de fútbol

Ejemplos de identificadores de tipos:

```
T"Día", T"Mes", T"País", T"País comunitario", T"Posición", ...
```

ASOC representa a una **asociación**: "AS" más el identificador de asociación, que se obtiene como una lista genérica de identificadores (los orígenes) más el nombre de la asociación (una cadena), o bien una variable. La lista de identificadores genéricos (LIDG) se construye de forma similar a la lista de instancias genéricas, teniendo en cuenta que cada elemento de la lista es una cadena (identificador de concepto) o bien otra lista de identificadores genéricos seguidos de una cadena (identificador de asociación).

ASOC ← *"AS"LIDG"."cad* | *IDVAR*

LIDG ← *"("* *LLIDG* *")"*

LLIDG ← *cad* | *cad* *","* *LLIDG* | *LIDG"."cad* | *LIDG"."cad* *","* *LLIDG*

Equipo de fútbol

Ejemplos de identificadores de asociaciones:

```
AS("Persona")."Nacimiento"
AS("Persona", "Fecha")."Pertenencia"
AS(("Persona", "Fecha")."Pertenencia")."Fin de pertenencia"
```

Equipo de fútbol

Algunos ejemplos de variables:

- Una variable "p" que representa una persona cuantificada universalmente:

```
para todo ?"p" de C"Persona"
```

- Una variable "país" que representa un valor de tipo "País" cuantificado existencialmente:

```
existe ?"país" de T"País"
```

- Una variable "e" que representa a un equipo sin cuantificar:

```
sea ?"e" de C"Equipo"
```

Una vez vista la definición de variables se procede a describir la definición de fórmulas. **FORM** es una **fórmula lógica**, que ya no tiene representación de variables y que puede ser:

- La conjunción de dos fórmulas
- La disyunción de dos fórmulas
- La implicación de dos fórmulas
- La negación de una fórmula
- Una fórmula entre paréntesis
- Un átomo

FORM ← FORM "and" FORM | FORM "or" FORM | "si" FORM "entonces" FORM "fin si" | "not" FORM | "("FORM")" | ATOM

ATOM es un **átomo** que representa una pregunta elemental realizada sobre las instancias de un modelo estructural:

- Una instancia de concepto pertenece a un concepto
- Un par (instancia, valor) pertenece a un atributo de concepto
- Una lista de instancias genéricas pertenece a una asociación
- Un par (lista de instancias genéricas, valor) pertenece a un atributo de asociación
- Un valor pertenece a un tipo
- Comparación entre valores del mismo tipo: igual, distinto, mayor, mayor o igual, menor, menor o igual.
- Comparación entre instancias de conceptos: igual, distinto
- Comparación entre instancias de asociación: igual, distinto

ATOM ← INST "pertenece a" CONC | "("INST "," EVAL ")" "pertenece a" ATRC | LING "pertenece a" ASOC | "("LING "," EVAL ")" "pertenece a" ATRA | EVAL "pertenece a" TIPO | EVAL CMP EVAL | INST CMPI INST | LING CMPI LING

INST representa el **identificador de una instancia de concepto**: "I" más una cadena que representa el nombre de instancia o bien el nombre de una variable que representa a instancia de concepto:

INST ← "I"cad | IDVAR

Equipo de Fútbol

Ejemplos de identificadores de instancias de concepto:

I"Loïc", I"Raúl", I"Figo", I"Rivaldo", I"1.8.2000", I"4-4-2", ...

EVAL representa a una **expresión que devuelve valores**. Puede ser: una expresión de valores numéricos, una expresión de valores de tipo cadena de caracteres, una expresión booleana, o bien el nombre de una variable que represente valores, o bien.

EVAL ← *EZ* | *ER* | *ECAD* | *EBOOL*

EZ representa una **expresión entera**. Puede ser: una operación binaria entre dos expresiones enteras, una operación unaria sobre una expresión entera, un valor entero o un identificador de variable

EZ ← *EZ OPZ EZ* | *OPZU EZ* | *n* | *IDVAR*

OPZ representa un **operador entero binario** (con dos operandos), que puede ser: suma, resta, multiplicación, división entera, módulo, potencia:

OPZ ← *"+"* | *"-"* | *"*"* | *"/"* | *"%"* | *"^"*

OPZU representa un **operador entero unario** (con un operando), que puede ser: cambio de signo o factorial:

OPZU ← *"-"* | *"!"*

n (símbolo terminal) representa un **número entero** cualquiera, positivo o negativo.

ER representa una **expresión real**. Puede ser: una operación binaria entre dos expresiones reales, una operación unaria sobre una expresión real, un valor real o un identificador de variable

ER ← *ER OPR ER* | *OPRU ER* | *r* | *IDVAR*

OPR representa un **operador real binario** (con dos operandos), que puede ser: suma, resta, multiplicación, división real, división entera, módulo, potencia:

OPR ← *"+"* | *"-"* | *"*"* | *"/"* | *"div"* | *"%"* | *"^"*

OPRU representa un **operador real unario** (con un operando), que puede ser: cambio de signo:

OPZU ← *"-"*

r (símbolo terminal) representa un **número real** cualquiera, positivo o negativo, con o sin decimales.

ECAD representa una **expresión de cadenas**. Puede ser: una operación binaria entre dos expresiones de cadenas, una cadena de caracteres o un identificador de variable

ECAD ← *ECAD OPC ECAD* | *cad* | *IDVAR*

OPC representa un **operador binario de cadenas** (con dos operandos), que puede ser: concatenar:

OPC ← *"+"*

EBOOL representa una **expresión booleana**. Puede ser: una operación binaria entre dos expresiones booleanas, una operación unaria sobre una expresión booleana, un valor booleano o un identificador de variable.

EBOOL ← *EBOOL OPB EBOOL* | *OPBU EBOOL* | *BOOL* | *IDVAR*

OPB representa un **operador booleano binario** (con dos operandos), que puede ser: and, or, xor o implica:

OPN ← *"and"* | *"or"* | *"xor"* | *"implica"*

OPBU representa un **operador booleano unario** (con un operando), que puede ser: not:

OPBU ← *"not"*

BOOL representa un **valor booleano**: "v" (verdadero) o "f" (falso):

BOOL ← *"v"* | *"f"*

Equipo de fútbol

Ejemplos de valores sencillos:

```
10, "Loïc Antonio Martínez Normand", ...
```

Ejemplos de expresiones de valores:

```
?"año" % 4
```

ATRC representa un **atributo de concepto**: "AT" más un identificador de atributo formado por dos cadenas (<nombre de concepto>.<atributo>):

ATRC ← *"AT"cad"."cad*

Equipo de fútbol

Ejemplos de identificadores de atributos:

```
AT"Persona"."nombre"
AT"Fecha"."año"
AT"Jugador"."posición"
AT"Jugador Campo"."remate cabeza"
```

LING representa una **lista de instancias genéricas** entre paréntesis que sirve para representar una tupla de una asociación cualquiera con al menos dos elementos, o bien un identificador de variable. Cada elemento de la lista será una instancia de concepto (INST) o bien otra lista de instancias genéricas (LING).

LING ← *"("LLING")"* | *IDVAR*

LLING ← *INST* | *INST "," LLING* | *LING* | *LING "," LLING*

Equipo de fútbol

Ejemplos de listas de instancias genéricas que representan tuplas de asociaciones:

```
(I"Loïc", I"16.10.1969")
(I"Rivaldo", I"1.8.1998", ?"club")
((I"Rivaldo", I"1.8.1998", ?"club"), I"1.8.2002")
```

ATRA representa un **atributo de asociación**: "ATS" más el identificador de la asociación entre paréntesis más punto más el nombre del atributo de asociación.

ATRA ← *"ATS" "(" LIDG "." cad ")" "." cad*

Equipo de fútbol

Ejemplos de identificadores de atributos de asociaciones:

```
ATS(("Persona", "Fecha")."Pertenencia")."es socio"
ATS(("Jugador", "Fecha")."Fichaje")."ficha"
ATS(("Directivo", "Fecha")."Presidente")."propietario"
```

CMP es un operador de **comparación de valores**: =, ≠, <, ≤, >, ≥

CMP ← *"="* | *"!="* | *"<"* | *"<="* | *">"* | *">="*

CMPI es un operador de **comparación de instancias**: =, ≠

CMPI ← *"="* | *"!="*

Equipo de fútbol

En primer lugar se muestran algunos **ejemplos de átomos**:

1. Instancia pertenece a concepto

```
I"Figo" pertenece a C"Jugador Campo"
?"jug" pertenece a C"Nacional"
```

2. Par (instancia, valor) pertenece a atributo

```
(I"Rivaldo", 8) pertenece a AT"Jugador campo"."regate"
(I"Raúl", "Delantero") pertenece a AT"Jugador"."posición"
(I"Figo", "Luis Figo") pertenece a AT"Persona"."nombre"
```

3. Tupla pertenece a asociación

```
(I"Loïc", I"16.10.1969") pertenece a AS("Persona")."Nacimiento"
(I"Rivaldo", I"1.8.1998", ?"club") pertenece a AS("Jugador",
"Fecha")."Fichaje"
((I"Rivaldo", I"1.8.1998", ?"club"), I"1.8.2002") pertenece a
AS(("Persona", "Fecha")."Pertenencia")."Fin de pertenencia"
```

4. Par (tupla, valor) pertenece a atributo de asociación

```
((I"Gaspart, I"20.7.2001", I"Barcelona"),f) pertenece a ATS(("Directivo",
"Fecha")."Presidente")."propietario"
```

5. Valor pertenece a tipo

```
"Francia" pertenece a T"País comunitario"
?"m" pertenece a T"Mes"
```

6. Comparación entre valores

```
?"p" ≠ "Media punta"
?"m" < 12
```

7. Comparación entre instancias

```
I"Figo" ≠ I"Raúl"
```

Seguidamente se muestran algunos **ejemplos de fórmulas sencillas** con o sin variables:

- "Casillas" es un portero y es un jugador nacional:

```
"Casillas" pertenece a C"Portero" and "Casillas" pertenece a C"Nacional"
```

- Una persona puede ser jugador o directivo (se usa la variable "p" que representa a cualquier persona):

```
?"p" pertenece a C"Jugador" or ?"p" pertenece a C"Directivo"
```

- Raúl es un jugador nacional y, por lo tanto, su país es "España"

```
si I"Raúl" pertenece a C"Nacional" entonces (I"Raúl", "España") pertenece
a AT"Persona"."país" fin si
```

- Rivaldo no es jugador comunitario

```
not I"Rivaldo" pertenece a C"Comunitario"
```

Por último se verá un **ejemplo de condiciones lógicas completas**, que incluye variables:

- **Para toda fecha se define una restricción** (en su invariante) sobre los valores de los atributos "día" (número natural entre 1 y 31), "mes" (número natural entre 1 y 12) y "año" (número natural) para considerar la fecha válida. La fecha será válida si el día es menor que 31 cuando el mes es abril (4), junio (6), septiembre (9) o noviembre (11); si el día es menor que 30 cuando el mes es febrero (2) y el año es bisiesto (múltiplo de cuatro y no múltiplo de 100 y sí múltiplo de 400); si el día es menor que 29 cuando el mes es febrero y el año no es bisiesto; y, por último, si estamos en cualquier otro mes la fecha siempre es válida.

```
para todo ?"f" de C"Fecha"
 existe ?"d" de T"Día"
  existe ?"m" de T"Mes"
   existe ?"a" de T"Año"
    si
     (?"f", ?"d") pertenece a AT"Fecha"."día" and
     (?"f", ?"m") pertenece a AT"Fecha"."mes" and
     (?"f", ?"a") pertenece a AT"Fecha"."año"
    entonces
     (?"m"=4 or ?"m"=6 or ?"m"=9 or ?"m"=11) and (?"d"<31) or
     (?"m"=2 and ?"a"%4=0 and (?"a"%100!=0 or ?"a"%400=0) and
     (?"d"<30) or
     (?"m"=2 and (?"a"%4!=0 or (?"a"%100=0 and ?"a"%400!=0)) and
     (?"d"<29)
    fin si
   fin condición
  fin condición
 fin condición
fin condición
```

Puede observarse cómo la formulación de esta condición lógica mediante el lenguaje definido permite expresarla de una forma más sencilla y clara para el lector.

Respecto a las condiciones lógicas binarias (en dos estados de tiempo), la diferencia fundamental estriba en los átomos binarios (BPC, BPAT, BPAS, BPATS), que admiten como parámetro la indicación del instante de tiempo (inicial o final) en el que se deben evaluar. En el lenguaje esto se representará añadiendo, al final del átomo, el siguiente texto:

- "En [1]" cuando el átomo se refiere al instante inicial de la condición.
- "En [2]" cuando el átomo se refiere al instante final de la condición.

4.4 NOTACIÓN GRÁFICA

La notación gráfica propuesta está basada en el Lenguaje Unificado de Modelado – UML [Booch et al., 1999], dado que es la notación gráfica de modelado más utilizada y referenciada en la actualidad. De UML se van a utilizar dos tipos de diagramas:

- *Diagramas de clases*, para representar tanto el modelo estructural como las relaciones tarea – método. La Figura 4.34 recoge los elementos fundamentales de la notación de diagramas de clases de UML.

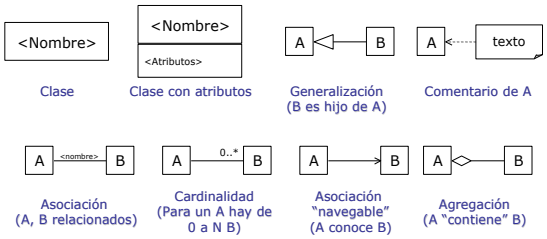


Figura 4.34 Notación básica de diagramas de clases de UML

- *Diagramas de actividad*, para representar el control de los modelos. La Figura 4.35 recoge los elementos fundamentales de la notación de diagramas de actividad de UML.

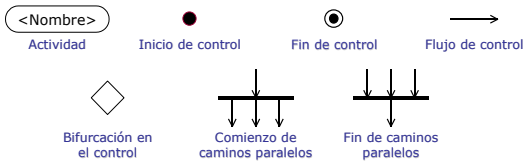


Figura 4.35 Notación básica de diagramas de actividad de UML

Para definir la notación gráfica a partir de UML se va a utilizar uno de sus mecanismos de extensión denominado estereotipos. Un estereotipo define un "versión especializada" de un elemento de un tipo (clase, actividad, etc.). Se representan como un texto encerrado entre << y >>.

Antes de proceder a detallar la notación gráfica de cada uno de los elementos de la técnica propuesta el autor desea poner de manifiesto que uno de los objetivos perseguidos con los diagramas que se van a especificar es que una herramienta pueda generarlos de forma automática a partir de una representación interna del modelo.

4.4.1 Notación gráfica del Modelo Estructural

4.4.1.1 Notación gráfica de Tipos

Los tipos se van a representar como clases de UML con el estereotipo <<tipo>> (Figura 4.36):

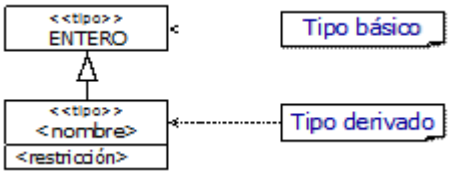


Figura 4.36 Representación gráfica de tipos

- En el caso de los tipos básicos se representará únicamente su nombre en mayúsculas (Entero, Real, Cadena, Booleano).
- En el caso de los tipos derivados se representará, además de su nombre, las restricciones, que se mostrarán en la sección de atributos del símbolo de clase, usando la notación descrita en el lenguaje de representación de tipos.
- La relación entre los tipos derivados y sus tipos base se representará mediante relaciones de generalización.

Como ejemplo de aplicación de esta notación gráfica, la Figura 4.37 recoge el diagrama de representación de todos los tipos del ejemplo de equipos de fútbol.

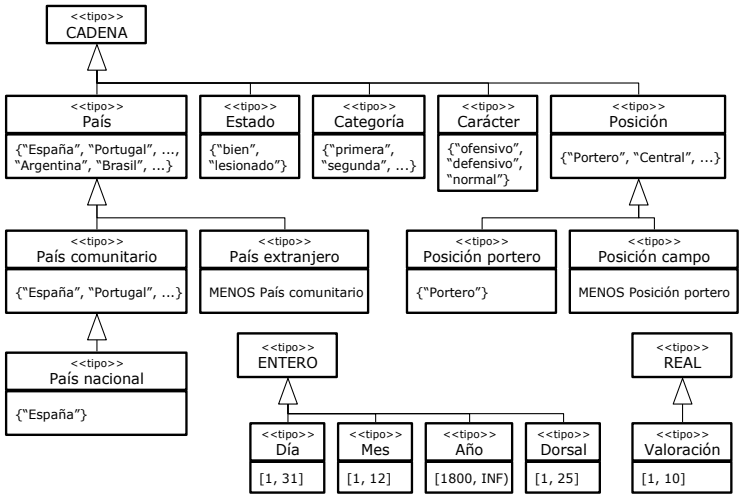


Figura 4.37 Diagrama de tipos del ejemplo de fútbol

4.4.1.2 Notación gráfica de Conceptos, Atributos y Clasificaciones

Los conceptos se van a representar como clases de UML con el estereotipo <<concepto>> (Figura 4.38):

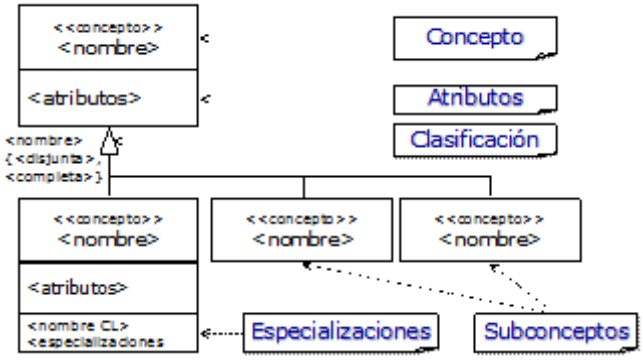


Figura 4.38 Notación gráfica de conceptos, atributos, clasificaciones

- Se representará como nombre de la clase el nombre del concepto
- Los atributos del concepto se representarán en la sección de atributos, usando la notación descrita en el apartado dedicado al lenguaje de representación de atributos.
- Las invariantes de concepto no se representan en los diagramas por razones de claridad.
- Las clasificaciones se representarán como relaciones de generalización entre el concepto y sus subconceptos.
- El nombre de la clasificación y sus características (completa, disjunta) se representarán como texto colocado cerca del superconcepto.
- En los subconceptos, debajo de los atributos, habrá una sección nueva para cada clasificación a la que pertenezca, en la que se mostrarán sus especializaciones.

Como ejemplo de aplicación de esta notación gráfica, la Figura 4.39 recoge un diagrama con la jerarquía de personas del ejemplo de equipos de fútbol.

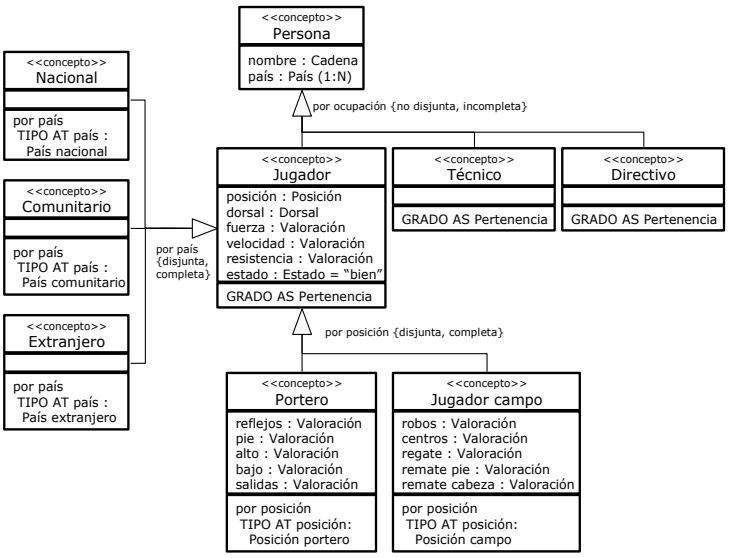


Figura 4.39 Diagrama con la jerarquía de personas del ejemplo de fútbol

4.4.1.3 Notación gráfica de Asociaciones, Atributos y Clasificaciones

Las asociaciones se van a representar como clases de UML con el estereotipo <<asociación>> (Figura 4.40):

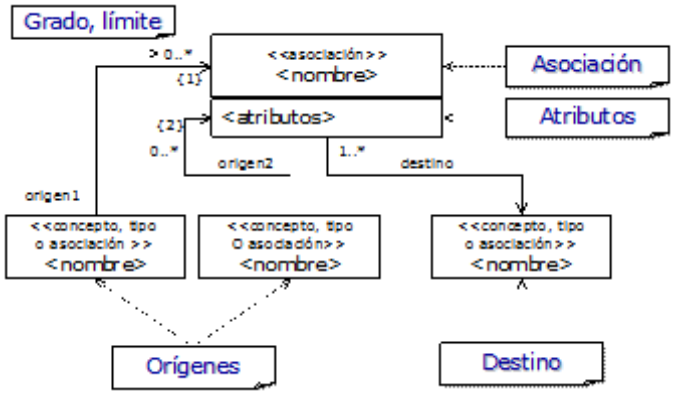


Figura 4.40 Notación gráfica de asociaciones, atributos, clasificaciones

- Se representará como nombre de la clase el nombre de la asociación.
- Los atributos de la asociación se representarán en la sección de atributos, usando la notación descrita en el apartado dedicado al lenguaje de representación de atributos.
- Las invariantes de asociación no se representan en los diagramas por razones de claridad.
- La relación entre la asociación y sus orígenes se representa como una línea con navegación desde el participante hacia la asociación, con nombre el correspondiente al rol y reflejando el límite y grado del rol como cardinalidad en el extremo correspondiente a la asociación. Para identificar el orden dentro de los orígenes, se pondrá un número entre llaves cerca del extremo de la línea más cercano a la asociación.

- La relación entre la asociación y su destino se representa como una línea con navegación desde la asociación hacia el participante, con nombre el correspondiente al rol y reflejando el límite y grado del rol como cardinalidad en el extremo correspondiente a la asociación.
- Las clasificaciones de asociación se representan como relaciones de generalización entre la superasociación y sus subasociaciones. El nombre de la clasificación y sus características (completa, disjunta) se representarán como texto colocado cerca del de la asociación.
- En las subasociaciones, debajo de los atributos, habrá una sección nueva para cada clasificación a la que pertenezca, en la que se mostrarán sus especializaciones.

Como ejemplo de la aplicación de esta notación gráfica, la Figura 4.41 muestra las asociaciones pertenencia y fin de pertenencia, así como la clasificación de pertenencia en fichaje, entrenador y presidente. En este diagrama se han ocultado los detalles relacionados con los conceptos involucrados en las asociaciones (Fecha, Persona, Equipo, Jugador, Técnico, Directivo). Tampoco se muestran los participantes de las subasociaciones, con el fin de no complicar en exceso el diagrama.

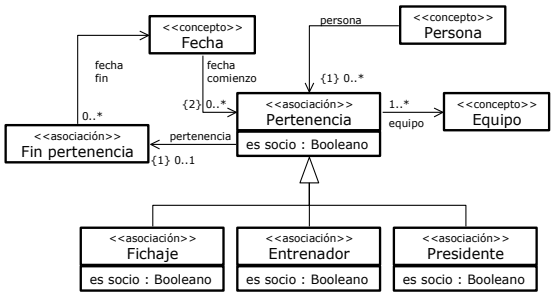


Figura 4.41 Representación gráfica de asociaciones en el ejemplo de fútbol

4.4.2 Notación gráfica del Modelo de Comportamiento

4.4.2.1 Notación gráfica de Tareas y Métodos

Las tareas se van a representar como clases con el estereotipo <<tarea>> y los métodos usarán el estereotipo <<método>> (Figura 4.42):

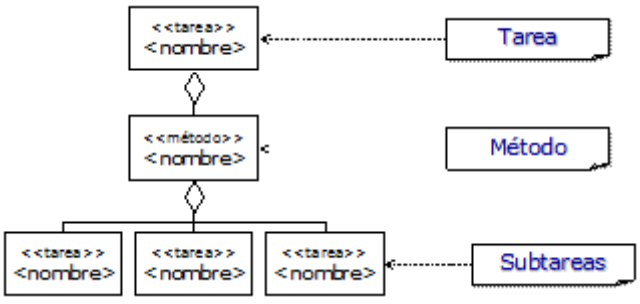


Figura 4.42 Notación gráfica de tareas y métodos

- La relación entre una tarea y el método o métodos que permiten resolverla se representa como una relación de agregación sin cardinalidades.

- La relación entre un método y sus subtareas también se representa como una agregación sin cardinalidades.

A modo de ejemplo, la Figura 4.43 muestra la resolución de la tarea "Crear alineación de equipo" mediante el método "Partiendo de táctica".

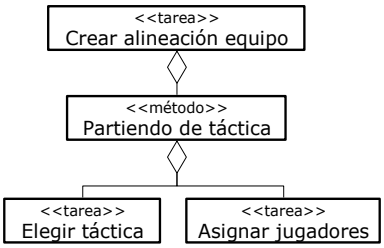


Figura 4.43 Un diagrama de métodos y tareas en el ejemplo de fútbol

4.4.2.2 Notación gráfica del Control de Métodos

Para el control de los métodos se van a utilizar los diagramas de actividad, con algunas extensiones que permitan representar los distintos nodos de control existentes en el método propuesto.

La Figura 4.44 recoge las ampliaciones realizadas para representar nodos de control:

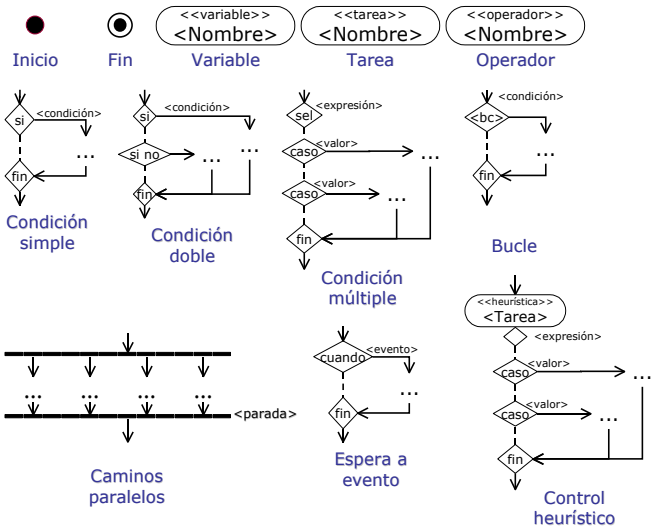


Figura 4.44 Notación para especificar control de métodos

- Un nodo de final de control se representa usando el símbolo habitual en UML.
- Un nodo de variable se representa como una actividad con estereotipo <<variable>>.
- Un nodo de resolución de tarea se representa como una actividad con estereotipo <<tarea>>.
- Un nodo de ejecución de operador se representa como una actividad con estereotipo <<operador>>.
- Los nodos de condiciones y bucles se representan con varios elementos de bifurcación etiquetados y unidos por líneas discontinuas. Las condiciones y expresiones se representan como comentarios.

- Un nodo de caminos paralelos se representa usando el símbolo de sincronización de UML y añadiendo un comentario para identificar el tipo de parada.
- Un nodo de eventos se representa con una bifurcación con una única salida y la etiqueta evento. El evento se representa como comentario.
- Un nodo de control heurístico se representa con una combinaciones de actividades y bifurcaciones etiquetadas convenientemente.

Como ejemplo, la Figura 4.45 recoge el control del método "Partiendo de táctica":

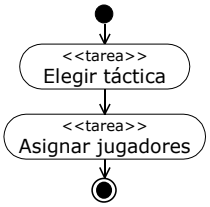


Figura 4.45 Ejemplo de diagrama de control de método

4.5 INTRODUCCIÓN AL PROCESO DE ANÁLISIS

Como ya se mencionó en el capítulo de hipótesis de trabajo, la definición de un proceso de análisis no es uno de los objetivos de esta tesis, sino que queda abierto como línea de trabajo futuro.

Sin embargo sí se pueden dar algunas indicaciones de las características que debería tener un proceso de análisis que utilizara el método propuesto:

- El proceso debe ser **iterativo e incremental**, de forma que los modelos se vayan creando poco a poco en varias iteraciones.
- El proceso debe ser **oportunista**, de forma que en cada momento se trabaje sobre la parte del modelo más adecuada, bien debido a preferencias del cliente, bien por el resultado de la verificación de las restricciones de los elementos del modelo o incluso por el resultado de aplicar un modelo de calidad. En este sentido ha comenzado a trabajarse en la adaptación del modelo de calidad propuesto en [Fuertes, 2003].
- El **grado de detalle será relativo** a las necesidades de cada proyecto. Así, podrán existir casos en los que no sea necesario definir invariantes de concepto, precondiciones de tareas, etc.
- El proceso de análisis deberá permitir generar un modelo de análisis con la suficiente cantidad de información como para poder **decidir cuáles son las mejores técnicas de diseño** para continuar con el desarrollo.

5 RESULTADOS

Una vez ha sido expuesta la técnica de análisis del problema SETCM, ha llegado el momento de mostrar los resultados obtenidos. Este capítulo está dividido en tres partes. En primer lugar se describirá la experimentación realizada con el método propuesto. En segundo lugar se realizará una breve descripción del desarrollo que se he realizado de una herramienta que dé soporte al método presentado en esta Tesis. Por último, se enumerarán las publicaciones derivadas del presente trabajo.

5.1 EXPERIMENTACIÓN

Este apartado de experimentación comenzará con el diseño experimental, en el que se presentarán los objetivos de la experimentación, así como los casos de estudio que han sido considerados para cubrir estos objetivos. Posteriormente se mostrarán los resultados de la aplicación de SETCM a los casos de estudio.

5.1.1 Diseño experimental

La experimentación debe ser diseñada partiendo de los cinco criterios que se establecieron para la comparación de técnicas de análisis del problema y que sirvieron para identificar los objetivos de la presente Tesis:

1. *Base formal*: el método debe tener una base formal completa, de forma que todos los elementos de modelado utilizados tengan una definición precisa.
2. *Independencia de paradigmas de diseño*, para que el uso del método de análisis no imponga la utilización de técnicas de diseño antes de que se comprenda realmente el problema.
3. *Adecuación a tipos de problemas*: el método debe permitir analizar de forma adecuada problemas de cualquier tipo (algorítmicos, de soporte a la decisión o basados en conocimientos).
4. *Notación gráfica*: el método debe ofrecer una notación gráfica que complemente la especificación detallada textual.
5. *Notación textual*: el método debe ofrecer como notación textual un lenguaje formal sin uso de símbolos matemáticos.

Obviamente, el primero de los criterios, que no es otro que la base formal del método, conlleva la prueba en su propia definición, por lo que no requiere de experimentación. Por lo tanto, tal y como recoge la Figura 5.1, el diseño experimental tratará de cubrir los cuatro criterios restantes: la independencia de diseño, la adecuación a cualquier tipo de problemas y la bondad de las representaciones gráfica y textual.

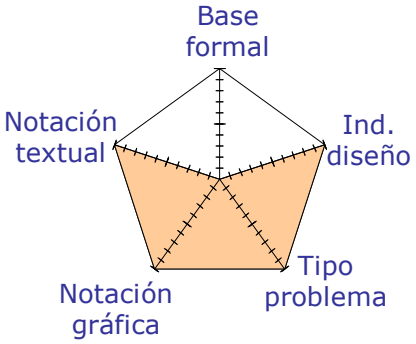


Figura 5.1 Criterios evaluados durante la experimentación

Partiendo de estos criterios, los **objetivos de la experimentación** son los siguientes:

1. *Evaluar la adecuación de SETCM a problemas de distinto tamaño*. Se abordará el análisis tanto de problemas pequeños, realizados en poco tiempo por equipos de una o dos personas, como de problemas grandes, con tiempos de desarrollo de varios años y equipos de varias personas.
2. *Evaluar la adecuación de SETCM en el desarrollo de sistemas con distintos paradigmas de diseño e incluso con paradigmas mixtos*. Se abordarán proyectos con diseños de naturaleza distinta, así como sistemas cuyo diseño requiera la combinación de distintos paradigmas.
3. *Evaluar la adecuación de SETCM para su uso por analistas con distinto nivel de experiencia*. Habrá proyectos en los que los analistas sean noveles (alumnos de la Facultad de Informática), de nivel medio (recién licenciados y estudiantes de doctorado) y expertos (profesores de la Facultad).
4. *Evaluar la curva de aprendizaje de SETCM*, tanto en usuarios noveles como expertos, con el fin de averiguar si es fácil de aprender su uso, independientemente de si se conoce o no alguna otra técnica de análisis.
5. *Evaluar la capacidad de ampliación de modelos generados con SETCM*. Se trata de comprobar si el uso de SETCM permite abordar proyectos realizados en varias fases, en los que el análisis de una fase consista en ampliar el análisis de la fase anterior.
6. *Evaluar la capacidad de integración de SETCM dentro de los procedimientos de una organización*. Se tratará de averiguar si es posible que una organización adopte SETCM como la técnica de análisis preferida para el desarrollo de software. Este objetivo es, en realidad, consecuencia de todos los anteriores.

Cada uno de los objetivos de la experimentación cubre en mayor o menor medida los cuatro criterios que se desea evaluar de forma empírica, tal y como se recoge en la Figura 5.2. Puede observarse cómo hay objetivos centrados en los criterios esenciales (independencia de diseño y adecuación a tipos de problemas) y otros más centrados en los criterios accidentales o pragmáticos (notación gráfica y textual). Es interesante destacar que todos los objetivos de experimentación permiten evaluar las notaciones gráficas y textual, en mayor o menor medida.

Con el fin de cubrir estos objetivos de experimentación y teniendo en cuenta el marcado carácter pragmático de la propuesta realizada en esta Tesis, ha sido necesario tener especial cuidado en la selección de los casos de estudio. En este sentido se podrían considerar dos enfoques diferentes:

1. *Experimentación en anchura*: los casos de estudio cubren la fase de análisis de varios proyectos de tamaño reducido.
2. *Experimentación en profundidad*: los casos de estudio cubren la evolución en el tiempo de un proyecto relativamente complejo.

En el caso de SETCM ha sido necesario utilizar ambos enfoques, lo que ha supuesto un esfuerzo considerable de validación que ha sido distribuido en la participación en varios proyectos durante un amplio espacio de tiempo, durante el cual la técnica de análisis ha

ido evolucionando hasta llegar a la forma que tiene en la presente Tesis. Por otra parte conviene destacar que se ha tratado de demostrar la viabilidad práctica del método eligiendo como casos de estudio proyectos reales, desarrollados para entidades públicas y privadas.

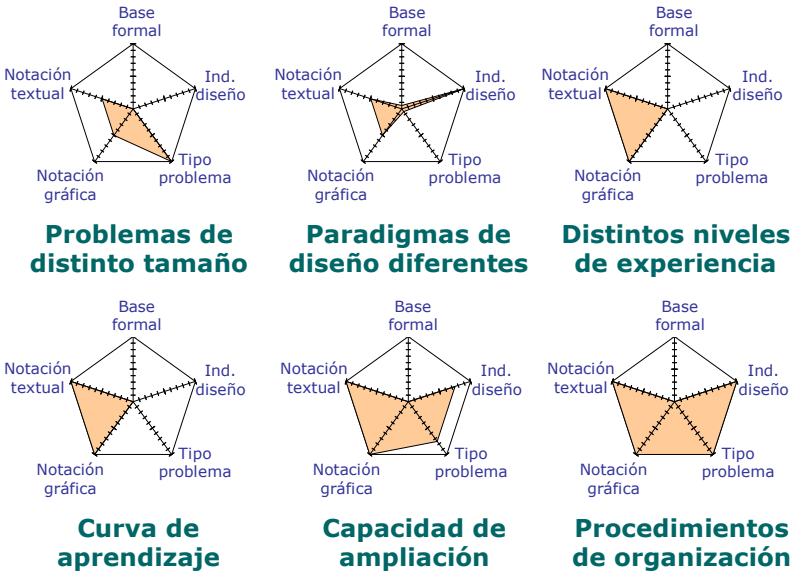


Figura 5.2 Criterios cubiertos por cada objetivo de la experimentación

En total se han considerado 13 casos de estudio, correspondientes todos ellos a proyectos que han concluido en productos transferidos, que se enumeran a continuación en orden cronológico, salvo en el caso de continuaciones de proyectos, que aparecerán inmediatamente después del proyecto original.

Para cada caso de estudio se realizará una breve descripción del mismo, así como una relación de los objetivos cubiertos total o parcialmente.

- **Caso 1: proyecto WINDI**, acrónimo de "Windows para Invidentes". Este proyecto terminó en 1997 y se realizó en colaboración con la ONCE (Organización Nacional de Ciegos Españoles). El objetivo principal del proyecto era la adaptación del entorno gráfico de MS Windows 3.1 para que pudiera ser utilizado por personas ciegas [Alonso, 1996] [Berengeno, 1997] [del Barrio, 1996] [Fuertes, 1999] [Rodríguez, 1996].

  Dentro de este proyecto se presenta como caso de estudio la realización de un modelo conceptual de todos los elementos de Windows [CETTICO, 1995a] y de la interacción entre el usuario y el entorno [CETTICO, 1995b]. Este proyecto supone la primera aplicación de las ideas que han culminado en esta propuesta de Tesis.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema grande.
  - *Criterio 2 – Paradigmas de diseño*: sistema estructurado dirigido por eventos.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles y medios.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles y medios.

- **Caso 2: proyecto WINLEE.** Este proyecto fue realizado para la ONCE y su desarrollo concluyó en 1997. El proyecto consistió en la realización de un lector de documentos para personas ciegas, que utiliza un escáner y un motor OCR (del inglés *Optical Character Recognition*, es decir, reconocedor óptico de caracteres) para permitir a las personas ciegas consultar documentos en papel [López, 2000] [Martínez, 2000]. El nombre del proyecto proviene del hecho de que se trate de una aplicación Windows con un comportamiento similar a un programa en MS-DOS llamado LEE [ONCE, 1996].

  El análisis del proyecto fue realizado utilizando una versión inicial de los modelos presentados en esta Tesis. Su implementación fue realizada mediante programación visual en un lenguaje de programación orientado a objetos, utilizando componentes ya existentes para el OCR y el uso del escáner. La interfaz de usuario fue diseñada para poder ser utilizada fácilmente por personas ciegas sin experiencia en el entorno Windows.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema pequeño.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos, dirigido por eventos, con interfaz adaptada e integración de componentes.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.
- **Caso 3: proyecto MEHIDA-PC.** El objetivo de este proyecto fue pasar a entorno Windows el sistema inteligente de enseñanza MEHIDA [Alonso et al., 1995], realizado en ordenadores Macintosh para que los niños sordos aprendieran a comunicarse usando distintas formas de comunicación: dactilológico, lengua de signos, lengua escrita, lectura labial, etc. Este proyecto, que concluyó en 1997, fue realizado en CETTICO (Centro de Transferencia Tecnológica en Informática y Comunicaciones), consistiendo en el desarrollo de la plataforma software necesaria para permitir ejecutar las actividades de MEHIDA en un PC con sistema operativo Windows [González, 1996] [Frutos et al., 1998].

  Este caso de estudio presenta el análisis de la funcionalidad requerida en el motor de comunicación multimedia que sustituyó al sistema original, que fue desarrollado usando la herramienta de autor Director de la empresa Macromedia. El sistema final fue implementado en el paradigma orientado a objetos y dirigido por eventos, con una arquitectura basada en el desarrollo de componentes reutilizables.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema grande y complejo.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos, con arquitectura basada en componentes reutilizables y con interfaz multimedia.
  - *Criterio 3 – Nivel de experiencia*: analistas expertos y medios.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en expertos y medios.

- **Caso 4: proyecto TUTOR.** Este caso de estudio supone una continuación del anterior. Es un proyecto desarrollado para el IMSERSO (Instituto Nacional de Migraciones y Asuntos Sociales), órgano perteneciente al Ministerio de Sanidad y Consumo. El proyecto concluyó en 1999 y consistió en la realización de un sistema completo de tutoría inteligente para niños con problemas auditivos, utilizando la plataforma desarrollada como resultado del caso 3 [de la Flor, 2000].

  Durante el proyecto fue necesario ampliar el número de componentes, para lo cual se procedió al análisis de la funcionalidad de cada uno de ellos, tomando como punto de partida el análisis realizado durante el caso anterior. Aquí se usó una versión más actualizada del método propuesto en esta Tesis.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema grande y complejo.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos, con arquitectura basada en componentes reutilizables y con interfaz multimedia.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles y medios.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles y medios.
  - *Criterio 5 – Capacidad de ampliación*: se ampliaron los modelos del caso 3.

- **Caso 5: sistema de música para invidentes**, proyecto Ruiseñor. Este proyecto fue desarrollado para la ONCE y concluyó en 1998. Consistió en el desarrollo de un prototipo de sistema de composición musical con interfaz de usuario adaptada para su uso por personas ciegas, con el fin de evaluar la viabilidad de un sistema completo de estas características [Gil, 1998] [Plaza, 1998].

  El análisis de la funcionalidad del sistema se realizó utilizando la misma versión del método propuesto que fue usada en el caso 2. El prototipo final fue realizado en el paradigma orientado a objetos y dirigido por eventos, en un entorno de desarrollo no visual.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema media.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos, dirigido por eventos, con interfaz adaptada y entorno de desarrollo no visual.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.

- **Caso 6: proyecto DELE**, acrónimo de Diccionario Enciclopédico Larousse Electrónico. Este proyecto fue realizado para la editorial Larousse España (actualmente perteneciente a Spes Editorial), concluyendo su desarrollo en el año 1998. Su objetivo era la realización de una versión electrónica del diccionario "El Pequeño Larousse Ilustrado" de 1997 [Larousse, 1997], que permitiera a los usuarios consultar de forma cómoda y rápida todo su contenido [Rodríguez, 2002] [Sánchez, 2001]

El sistema resultante tiene dos grandes componentes. En primer lugar hay un componente encargado de la generación y mantenimiento del contenido del diccionario, tomando como punto de partida el formato de imprenta de la edición en papel. El segundo componente es la aplicación de consulta que permite acceder a esos contenidos. El análisis de este segundo componente fue realizado usando la misma versión de SETCM que en los casos 2 y 5.

Este caso contribuye a la evaluación de los siguientes objetivos:

- *Criterio 1 – Tipo de problema*: problema medio.
- *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos con interfaz visual y entorno de desarrollo visual.
- *Criterio 3 – Nivel de experiencia*: analistas noveles.
- *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.

- **Caso 7: juego educativo para niños ciegos – el ahorcado**. Este proyecto fue desarrollado para la ONCE y consistió en un juego educativo adaptado para niños ciegos pero que también fuera atractivo para niños videntes. De esta forma se pretendía aumentar la integración de los niños ciegos, ya que podrían jugar al mismo juego con niños videntes. El desarrollo de este sistema concluyó en 1999 [Ortega, 1999].

  El análisis completo del sistema desarrollado fue realizado siguiendo una versión de SETCM similar a la utilizada en los casos 2, 5 y 6. El sistema final tiene un diseño orientado a objetos, con interfaz gráfica y adaptada para personas ciegas, y fue implementado en un entorno de programación visual.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema pequeño.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos con interfaz visual y adaptada usando un entorno visual.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.

- **Caso 8: navegador de Internet para personas ciegas**. Este proyecto fue desarrollado para la ONCE y consistió en el desarrollo de un prototipo de navegador Web adaptado para personas ciegas, con el fin de estudiar la viabilidad de un sistema completo. El desarrollo del prototipo concluyó en 1999 [de la Cruz, 1999].

  En este caso se puso un gran énfasis en la fase de análisis del problema, con un doble objetivo. Por un lado se obtuvo un modelo conceptual de la estructura de una página HTML y por otro se obtuvo un modelo de la interacción de una persona ciega con esa página HTML. El prototipo final tiene un diseño orientado a objetos, con interfaz adaptada a personas ciegas y que utiliza un componente de librería para la interpretación de páginas HTML.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema pequeño.

- *Criterio 2 – Paradigmas de diseño*: arquitectura multinivel orientada a objetos con interfaz adaptada usando componentes estándar.
- *Criterio 3 – Nivel de experiencia*: analistas noveles.
- *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.

- **Caso 9: proyecto GESTLAB**. Este fue un proyecto interno de CETTICO, consistente en una aplicación de ayuda a la gestión de recursos del laboratorio (ordenadores, periféricos, software, libros, etc.), concluyendo su desarrollo en 1999 [García, 2000] [Moya, 1999].

  El proyecto fue diseñado como una aplicación cliente – servidor con soporte basado en tecnologías Java y HTML. Se utilizó SETCM para analizar fundamentalmente los datos que había que manejar (dado que las operaciones eran muy sencillas: altas, bajas y modificaciones), con lo que se obtuvo un modelo conceptual de la gestión interna del laboratorio.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema pequeño.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos en arquitectura cliente – servidor, con un modelo complejo de Base de Datos.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles y medios.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles y medios.

- **Caso 10: proyecto ALBOR**. El nombre de este proyecto es un acrónimo de "Acceso Libre de Barreras al Ordenador". Fue realizado para el IMSERSO, en colaboración con el CEAPAT (Centro Estatal de Autonomía Personal y Ayudas Técnicas) y ATAM (asociación dependiente de telefónica), terminando su desarrollo en el año 2000 [Amigo, 2000] [Frutos, 2000]. El objetivo inicial de ALBOR era la construcción de un Sistema Inteligente operativo en Internet capaz de asesorar a los profesionales tanto en la evaluación de personas con discapacidades (en lo relativo a su capacidad de acceso al ordenador), como en la elección de las adaptaciones tecnológicas más adecuadas para posibilitar dicho acceso.

  Para el análisis de ALBOR se decidió usar SETCM con el fin de estudiar el problema en detalle independientemente de su diseño final, que no era conocido al comenzar el proyecto. Una vez concluido el análisis se decidió que ALBOR tendría una arquitectura híbrida, con un sistema inteligente multiagente acompañado de componentes orientados a objetos para realizar las tareas más sencillas.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema muy complejo.
  - *Criterio 2 – Paradigmas de diseño*: diseño basado en agentes inteligentes distribuidos en Internet.
  - *Criterio 3 – Nivel de experiencia*: analistas expertos.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en expertos.

- **Caso 11: proyecto ALBOR-II**. Este proyecto consiste en la continuación del caso anterior, incorporando dos grandes funcionalidades. Por un lado se desarrolló un sistema distribuido de mantenimiento de los conocimientos de ALBOR, que permite realizar actualizaciones desde ordenadores conectados a Internet. Por otro lado se desarrolló una comunidad virtual (la red ALBOR) que permite a los profesionales del área comunicarse y colaborar mediante una serie de herramientas como foros, noticias, chat, etc. El proyecto terminó en 2001 y actualmente está utilizándose en la Consejería de Educación de la Comunidad Autónoma de Madrid [Alonso et al., 2001] [Frutos, 2001] [González, 2001].

  El análisis de las nuevas funcionalidades fue realizado en una versión muy reciente de SETCM, tomando como punto de partida el modelo generado en el caso de estudio anterior, e incorporando aquellos elementos necesarios para los nuevos subsistemas.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema muy complejo.
  - *Criterio 2 – Paradigmas de diseño*: diseño basado en agentes inteligentes distribuidos en Internet.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles y medios.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles y medios.
  - *Criterio 5 – Capacidad de ampliación*: ampliación de los modelos del caso 10.
- **Caso 12: proyecto Estenotipia**. Fue desarrollado en el año 2001 para la ONCE y consistió en la realización de un prototipo con el sistema software de una máquina de estenotipia que sustituyera a la que entonces utilizaban las personas ciegas: la máquina *Stenokey*, de origen búlgaro [Picazo, 2001]. El sistema incluye la interpretación de las pulsaciones según el método de estenotipia, así como un editor de textos para facilitar la corrección de las transcripciones realizadas.

  El análisis del sistema fue realizado en una versión reciente de SETCM, con un gran nivel de detalle. El sistema final fue implementado en MS-DOS (por restricciones del hardware de la futura máquina) con un diseño orientado a objetos.

  Este caso contribuye a la evaluación de los siguientes objetivos:

  - *Criterio 1 – Tipo de problema*: problema medio.
  - *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos en sistema con pocos recursos hardware.
  - *Criterio 3 – Nivel de experiencia*: analistas noveles.
  - *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.
- **Caso 13: proyecto Subtítulos**. Este proyecto se encuadra dentro de una serie de convenios entre la Universidad Politécnica de Madrid y el IMSERSO. El objetivo era la realización de un sistema de subtitulado en directo para televisión, basado en la utilización de estenotipia computerizada, y el proyecto terminó en el año 2002 [García, 2003].

El análisis de este sistema se realizó con una versión prácticamente definitiva de SETCM y su implementación se realizó con una arquitectura multiproceso y un diseño orientado a objetos.

Este caso contribuye a la evaluación de los siguientes objetivos:

- *Criterio 1 – Tipo de problema*: problema medio.
- *Criterio 2 – Paradigmas de diseño*: diseño orientado a objetos en sistema multiproceso.
- *Criterio 3 – Nivel de experiencia*: analistas noveles.
- *Criterio 4 – Curva de aprendizaje*: aprendizaje en noveles.

El lector habrá observado que en ninguno de los casos de estudio se ha mencionado el último objetivo experimental, a saber, la capacidad de integración de SETCM dentro de los procedimientos de una organización. En realidad sí se ha tenido en cuenta, dado que todos los proyectos han sido realizados dentro de la misma organización, el laboratorio SETIAM (Sección de Transferencia Informática en Apoyo a las Minusvalías) de CETTICO. Por lo tanto, el conjunto de todos los casos de estudio permitirá evaluar si es posible adoptar SETCM dentro de los procesos de una organización concreta.

A modo de resumen, la Tabla 5.1 recoge la correspondencia entre los objetivos experimentales, sus variantes y los casos de estudio que acaban de describirse.

Tabla 5.1 Correspondencia entre objetivos experimentales y casos de estudio

Objetivo	Variante	Casos de estudio
Problemas de distinto tamaño	Pequeño	2, 7, 8, 9
	Medio	5, 6, 12, 13
	Grande	1, 3, 4, 10, 11
Paradigmas de diseño	Estructurado	1
	Objetos	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
	Cliente – Servidor	9, 10, 11
	Agentes inteligentes	10, 11
	Dirigido por control	10, 11, 12, 13
	Dirigido por eventos	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13
	Uso de componentes	2, 4, 8, 9
	Creación componentes	3, 4
Nivel de experiencia	Noveles	1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13
	Medios	1, 3, 4, 9, 11
	Expertos	3, 10
Curva de aprendizaje	Noveles	1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13
	Medios	1, 3, 4, 9, 11
	Expertos	3, 10
Capacidad de ampliación de modelos		4, 11
Procedimientos de organización		1 – 13

En esta tabla pueden apreciarse algunas tendencias de los casos de estudio cuyas razones conviene explicar:

- La gran mayoría de los casos de estudio tienen un diseño orientado a objetos con un control dirigido por eventos (interfaz gráfica de usuario). Esto es así debido a que es

el tipo de aplicación más habitual desarrollado en SETIAM: aplicaciones de tamaño medio o pequeño con una interfaz especialmente adaptada a las capacidades de los usuarios finales.

- Respecto a los tipos de analistas, puede observarse como una gran mayoría de los sistemas han contado con la participación de analistas noveles. Esto es así por tratarse de un ambiente académico, ya que en el laboratorio SETIAM hay una gran participación de estudiantes de últimos cursos de la carrera.

En los apartados siguientes cada caso de estudio va a dividirse en dos partes. En primer lugar se dará una descripción con los aspectos más relevantes de su análisis, utilizando preferentemente la notación gráfica. En segundo lugar se mostrará la evaluación obtenida para los distintos objetivos experimentales. Los detalles de los casos de estudio se recogen en el Anexo C.

Respecto a la evaluación de cada caso, se ha seguido un proceso de experimentación cualitativa basada en cuestionarios que contestaron los participantes en los proyectos. La evaluación mostrada en este apartado es un resumen de los resultados de los cuestionarios.

Una vez concluida la descripción de los casos de uso se procederá a establecer los resultados generales de la experimentación.

El autor es consciente de que la descripción de los casos de estudio aquí recogida es muy extensa, pero se ha incluido en el cuerpo de la Tesis, en lugar de en forma de anexo, porque se considera muy relevante dado lo profundo y riguroso del procedimiento de experimentación seguido. Por esta razón, y como se ha mencionado antes, se ha dividido cada caso de estudio de forma que el lector pueda acceder rápidamente a cada parte de la experimentación.

5.1.2 Caso 1: proyecto WINDI

5.1.2.1 Descripción

Dentro de este caso de estudio lo más relevante fue el modelado conceptual de los elementos de Windows 3.1x [CETTICO, 1995a] y las tareas de un usuario en el entorno [CETTICO, 1995b].

El modelo de los elementos de Windows consta de 48 conceptos que representan ventanas, elementos de diálogo, etc., para cuya definición se trató de usar una terminología más descriptiva que la utilizada en el propio entorno, con el fin de que fuera más fácil de entender para personas ciegas. Entre los conceptos se definieron tres tipos básicos de relaciones:

1. *Clasificación*: un elemento puede ser de varios tipos mas específicos. Por ejemplo, un botón puede ser de texto o gráfico. La Figura 5.3 recoge todas las relaciones de clasificación recogidas en [CETTICO, 1995a], siendo todas ellas completas y disjuntas.
2. *Desencadena*: la actuación del usuario en el elemento origen provoca la aparición del elemento destino. La Figura 5.4 recoge todas las relaciones de activación recogidas en [CETTICO, 1995a].

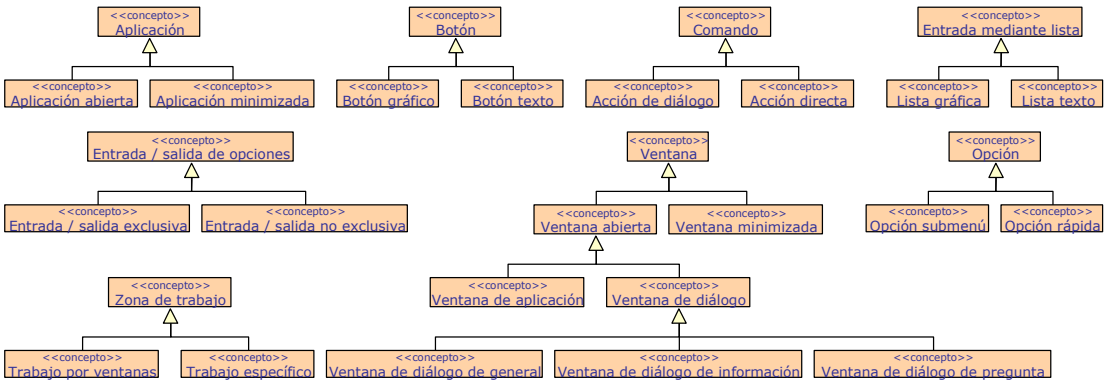


Figura 5.3 Clasificaciones de los elementos de Windows 3.1

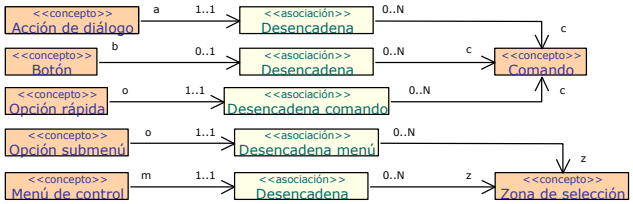


Figura 5.4 Relaciones de activación entre elementos de Windows 3.1

3. *Contiene*: un elementos (origen) contiene elementos de otro tipo (destino). Por ejemplo, una ventana de diálogo general puede contener botones. La Figura 5.5 refleja las relaciones de contenido establecidas en [CETTICO, 1995a]. Este diagrama es complejo debido a la gran cantidad de relaciones de contenido entre los componentes de Windows. Se han repetido dos conceptos ("Botón" y "Entrada / salida combinada", marcados con *) con el fin de evitar cruces de líneas.

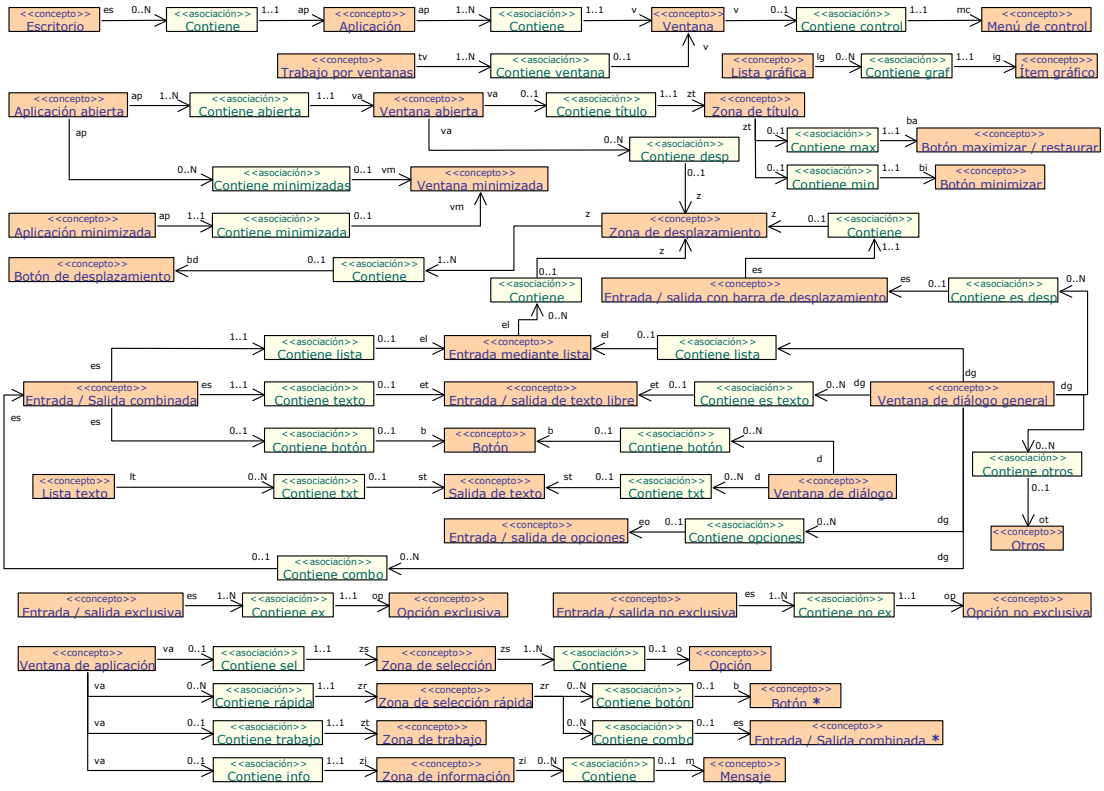


Figura 5.5 Relaciones de contenido entre elementos de Windows 3.1

El modelo de tareas de usuario consta de un total de 97, divididas en cuatro grandes grupos (Figura 5.6, en la que por razones de espacio y dado que cada una de las tareas tenía asignado un único método, no se representan los métodos de tarea):

- *Tareas generales*: son tareas que puede hacer el usuario independientemente del tipo de aplicación que esté utilizando. Este grupo contiene 56 tareas.
- *Tareas con parámetro*: es un conjunto de tareas de relativo bajo nivel (por ejemplo, seleccionar un objeto con el ratón) que tienen como parámetro el tipo de elemento con el que interactúa el usuario. Este grupo consta de 11 tareas.
- *Tareas del Administrador de Programas*. Dada la importancia de esta aplicación en el entorno Windows 3.1, se decidió hacer un estudio detallado de la interacción del usuario con esta aplicación. En este grupo hay 7 tareas.
- *Tareas del Administrador de Archivos*. También se estudiaron con especial interés las tareas que puede realizar el usuario con esta aplicación. Hay 23 tareas dentro de este grupo.

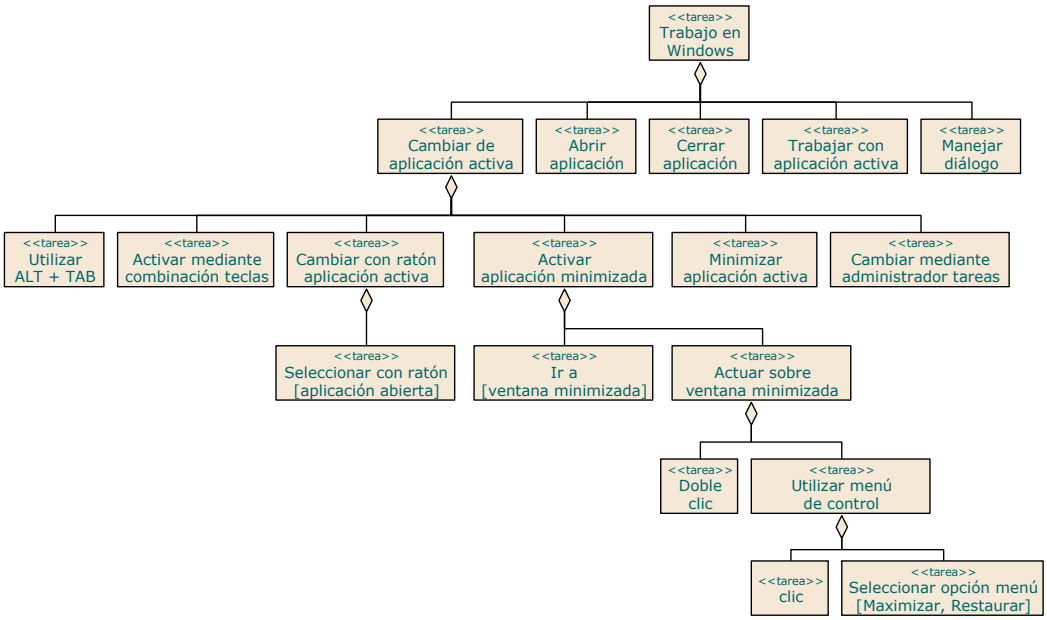


Figura 5.6 Diagrama parcial de descomposición de tareas de WINDI

El análisis de las tareas del usuario fue realizado con un gran nivel de detalle, pero de manera informal, ya que el objetivo perseguido era una primera aproximación a las tareas que podía realizar un usuario para su posterior adaptación mediante síntesis de voz y línea Braille. En el Anexo C se recoge algún ejemplo de especificación de tareas y métodos, mientras que el modelo completo puede encontrarse en [CETTICO, 1995b].

5.1.2.2 Evaluación

La Tabla 5.2 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.2 Evaluación de los objetivos experimentales para el caso 1

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Grande	7	Se pudo analizar la parte estática con el grado de detalle deseado, pero la parte dinámica fue más complicada de estudiar.

Objetivo	Variante	Nota	Comentarios
Paradigmas de diseño	Estructurado	8	No hubo problemas en la posterior transición a un diseño estructurado.
	Dirigido por eventos	4	Esta versión no facilitaba la representación de eventos, por lo que fue necesario un esfuerzo extra en diseño.
Niveles de experiencia	Noveles	7	Los analistas noveles pudieron centrarse en el problema, sin pensar en paradigmas de diseño, pero la notación requería de gran esfuerzo.
	Medios	7,5	Los analistas medios indicaron una gran facilidad para estudiar el problema, independientemente de las técnicas que ya conocieran con anterioridad. Estos analistas también detectaron defectos en la notación que se utilizaba.
Curva de aprendizaje	Noveles	8	Los analistas noveles aprendieron la técnica con considerable rapidez.
	Medios	7	A los analistas medios tenían que cambiar de hábitos para usar partes de la técnica.
Procedimientos de organización		5	Dado el éxito parcial obtenido se decidió comenzar una fase de pruebas de la técnica en futuros proyectos, para lo cual sería necesario trabajar más en la notación utilizada.

5.1.3 Caso 2: proyecto WINLEE

5.1.3.1 Descripción

El modelo estructural de WINLEE (Figura 5.7) maneja un número reducido de elementos (9 conceptos y 15 asociaciones), que pueden dividirse en dos grandes grupos:

- *La gestión del documento*. Esta categoría incluye los conceptos documento, posición, página, acción y las asociaciones correspondientes.
- *La gestión de parámetros de funcionamiento*. Esta categoría incluye los conceptos aplicación, impresora, línea Braille, OCR, síntesis de voz y las asociaciones correspondientes.

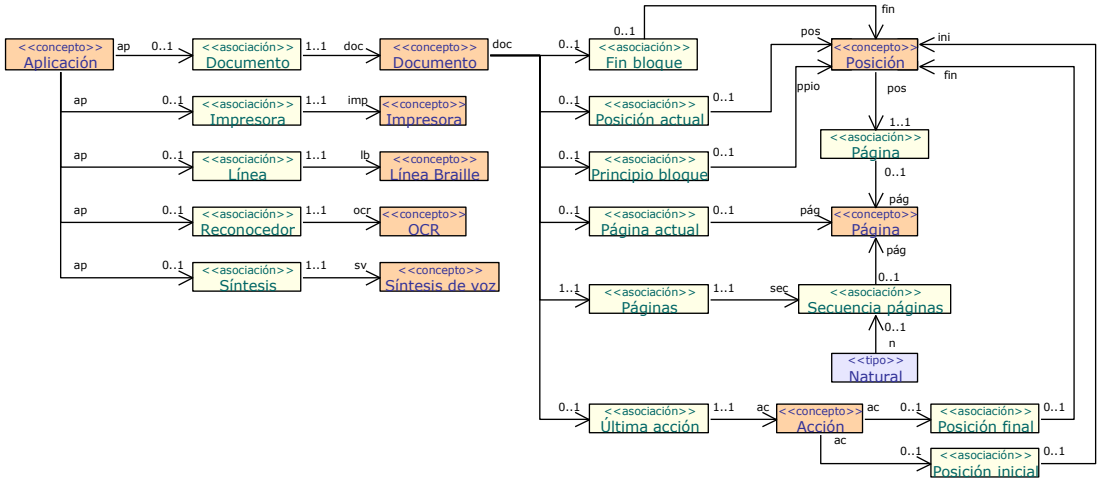


Figura 5.7 Modelo estructural de WINLEE

El modelo estructural completo está detallado en el Anexo C. Sin embargo en este apartado cabe destacar la existencia de dos asociaciones en las que uno de sus participantes no es un concepto:

1. *Secuencia páginas*: es una asociación entre el tipo de los números naturales y el concepto "Página", de forma que representa una secuencia ordenada de páginas:

```
ASOCIACIÓN Secuencia páginas
DESCRIPCIÓN:
  Representa una secuencia de páginas, de forma que a cada página se le asigna un
  número
ORIGEN:
  n : Natural (0:1)
DESTINO:
  pag : Página (0:1)
TIPO:
  función (parcial)
INVARIANTE
  -- La secuencia debe asignar páginas a números consecutivos: para todo número
  -- de la secuencia, o existe su inmediato superior o no existe ninguno mayor
  SEA sec DE AS"Secuencia páginas"
    EXISTE sec2 DE AS"Secuencia páginas"
      sec2.n = sec.n + 1
    FIN CONDICIÓN
    OR PARA TODO sec3 DE AS"Secuencia páginas"
      SI
        sec3 <> sec
      ENTONCES
        sec3.n < sec.n
      FIN SI
    FIN CONDICIÓN
  FIN CONDICIÓN
FIN ASOCIACIÓN
```

2. *Páginas*: es una asociación entre el concepto "Documento" y la asociación anterior, de forma que a cada documento se le asigna una secuencia ordenada de páginas:

```
ASOCIACIÓN Páginas
DESCRIPCIÓN:
  Relaciona un documento con su secuencia de páginas
ORIGEN:
  doc : Documento (1:1)
DESTINO:
  sec : (Natural).Secuencia páginas (1:1)
TIPO:
  función (completa)
FIN ASOCIACIÓN
```

Respecto al modelo de comportamiento, en WINLEE se identificaron 31 tareas, que representan las acciones que puede realizar el usuario del sistema, tal y como muestra la Figura 5.8, de nuevo sin los métodos de tarea.

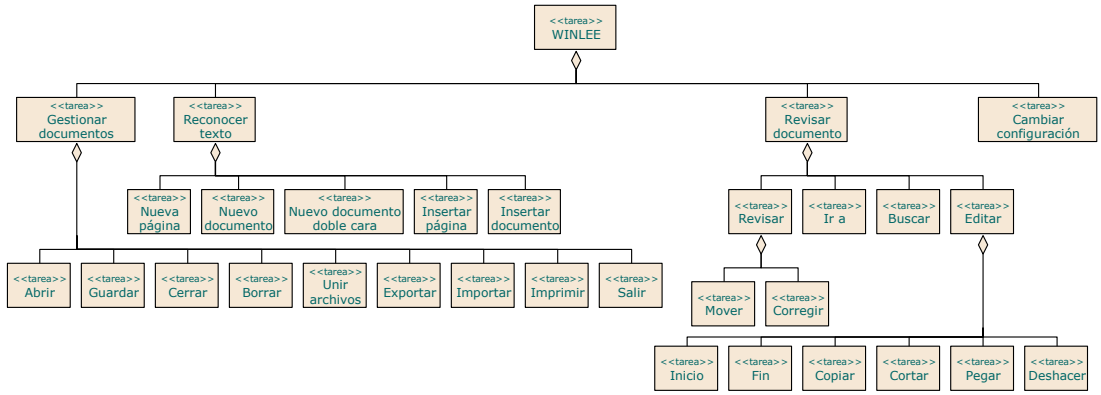


Figura 5.8 Diagrama de descomposición de tareas de WINLEE

La descomposición de tareas realizada en WINLEE partió de considerar una tarea global que representa toda la aplicación, dividida en cuatro tareas que representan los grandes grupos de funciones de la aplicación: gestión de documentos, reconocimiento de texto en papel, revisión del documento y cambios en la configuración. Estas tareas, a su vez, se dividen hasta llegar a las tareas elementales.

Respecto a los métodos de las tareas hay que indicar que el control de WINLEE es dirigido por eventos, como el de la mayoría de aplicaciones que funcionan en entorno Windows. En los dos primeros niveles se representan las subtareas como caminos de ejecución posiblemente paralelos (por ejemplo, la Figura 5.9 recoge el control del método de la tarea principal). En los siguientes niveles se representan gestiones de eventos dentro de caminos paralelos, como una forma de representar las opciones que tiene un usuario en cualquier momento (la Figura 5.10 recoge como ejemplo el control de la tarea "Editar").

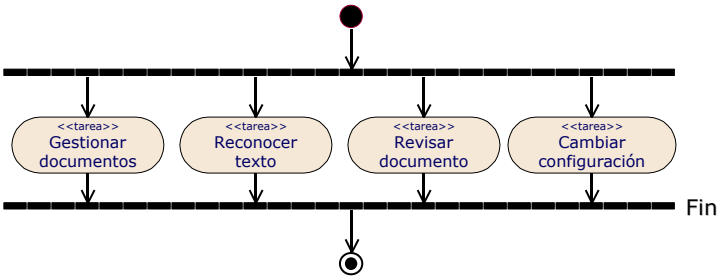


Figura 5.9 Diagrama de control del método principal de WINLEE

En el anexo C puede encontrarse la especificación de algunas tareas y métodos del proyecto WINLEE, mientras que la descripción completa está recogida en [Martínez, 2000], documento dedicado al desarrollo del núcleo interno del sistema. El otro Trabajo Fin de Carrera relacionado con este proyecto, [López, 2000], está dedicado al desarrollo de la interfaz de usuario adaptada para personas ciegas, tomando como punto de partida el modelo conceptual aquí presentado y estudiando a partir de ahí las necesidades específicas de la interfaz para comunicarse mediante voz o braille con el usuario.

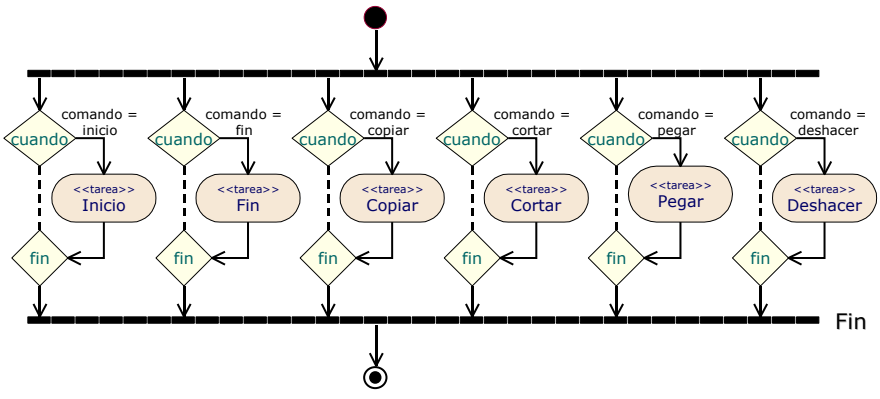


Figura 5.10 Diagrama de control del método "Editar" en WINLEE

5.1.3.2 Evaluación

La Tabla 5.3 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.3 Evaluación de los objetivos experimentales para el caso 2

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Pequeño	6	Se pudo analizar todo el problema con el grado de detalle deseado, pero suponía un exceso de esfuerzo para el tamaño del problema.
Paradigmas de diseño	Orientado a objetos	6	Hubo que diseñar unos pasos para la transición a objetos, que se basaron en la asignación de tareas a conceptos. El análisis estaba un poco orientado hacia el diseño estructurado.
	Dirigido por eventos	4	La versión utilizada empezó a tener en cuenta los programas dirigidos por eventos, pero aún no era adecuada para representar este tipo de control.
	Uso de componentes	8	El hecho de que en el diseño se usaran componentes para el OCR y el escáner no supuso ninguna dificultad en el análisis.
Niveles de experiencia	Noveles	7	Los analistas noveles pudieron centrarse en el problema, sin pensar en paradigmas de diseño, pero la notación prevista todavía requería demasiado esfuerzo.
Curva de aprendizaje	Noveles	8	Los analistas noveles no tuvieron dificultades para aprender a usar la técnica.
Procedimientos de organización		6	Este fue uno de los primeros proyectos piloto para la implantación de la técnica en el laboratorio. Seguía siendo necesario trabajar más algunos aspectos importantes.

5.1.4 Caso 3: proyecto MEHIDA-PC

5.1.4.1 Descripción

En el caso del proyecto MEHIDA-PC cabe destacar que se realizó el modelo de un sistema de tutoría inteligente genérico, por lo que en el modelo estructural se incluyen conceptos que representan las tareas concretas que realiza cada sistema de tutoría, como por ejemplo los procesos que pueden realizarse con los distintos elementos de la interfaz de usuario.

Por todo ello el modelo estructural de MEHIDA es muy complejo, con 6 tipos, 30 conceptos y 119 asociaciones. Teniendo en cuenta esta complejidad, en este apartado se mostrarán algunos de los elementos más relevantes (el modelo estructural está recogido de forma completa en el anexo C usando el lenguaje descrito en el capítulo de solución, y en [González, 1996] usando la notación definida en aquel momento).

En primer lugar, la Figura 5.11 muestra la estructura del método didáctico. Una unidad didáctica contiene una secuencia de actividades, cada una de las cuales es realizada por un alumno, mediante una interfaz, usando unos elementos de actividad. Además una actividad tiene una serie de procesos para su gestión.

El segundo fragmento de modelo estructural recogido en este apartado (Figura 5.12) muestra un elemento básico, la "Plantilla", que representa una pantalla de la interfaz de usuario.

Una plantilla tiene un rectángulo que define su aspecto, puede tener una imagen de fondo y contiene una secuencia de controles. Por otro lado, una plantilla tiene asociados 10 procesos que realizan tareas específicas sobre la plantilla, como mostrar todos los

controles, pintar el fondo, instalar la gestión de eventos de Windows, etc. Por último una plantilla puede tener una plantilla padre y una serie de plantillas a los que puede ceder el control en un momento dado.

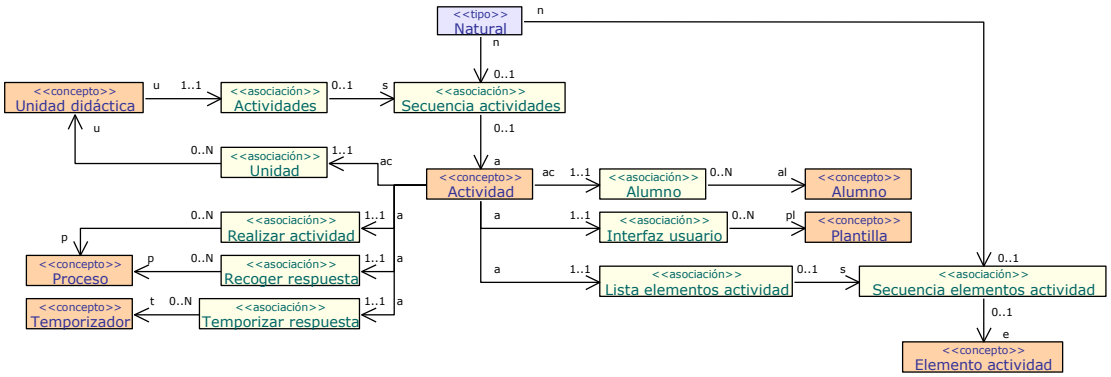


Figura 5.11 Diagrama estructural de actividades en MEHIDA-PC

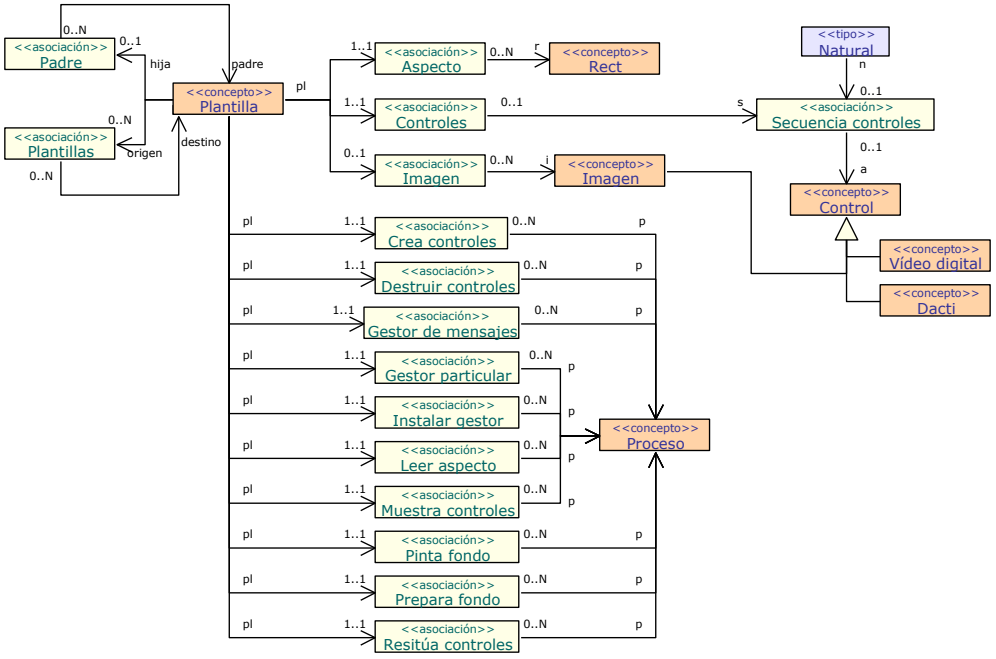


Figura 5.12 Diagrama estructural de una "Plantilla" en MEHIDA-PC

En ese mismo diagrama se muestra que en el modelo conceptual de MEHIDA se consideran tres tipos específicos de controles (además de los controles estándar de Windows, que no fueron modelados): una imagen, un vídeo digital y una animación de una palabra en formato dactilológico.

Respecto al modelo de comportamiento de MEHIDA, se identificaron 41 tareas, cada una de las cuales tiene un único método. En la Figura 5.13 se recoge el diagrama de descomposición de tareas en el que se han omitido la representación de métodos.

La mayoría de los métodos asociados a estas tareas presentan un control dirigido por eventos, similar al expuesto en el caso de WINLEE. Sin embargo hay tareas con métodos cuyo control no es dirigido por eventos, como el método de la tarea "Traducir a dactilológico", cuyo diagrama de control se muestra en la Figura 5.14.

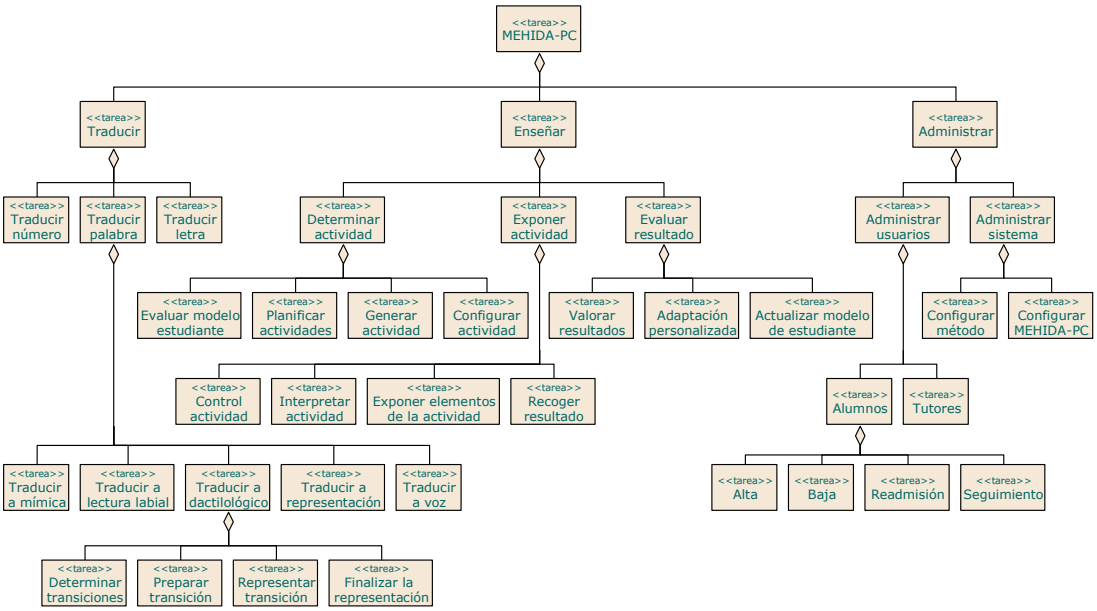


Figura 5.13 Diagrama de descomposición de tareas de MEHIDA-PC

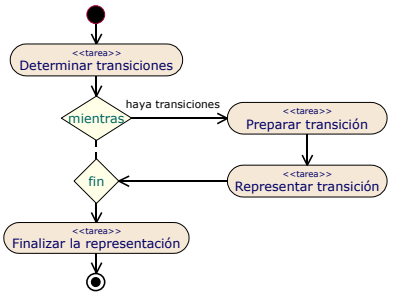


Figura 5.14 Diagrama de control de "Traducir a dactilológico" en MEHIDA-PC

Esta tarea requiere descomponer una palabra en sus transiciones (pares de letras), luego se realiza un bucle en el que se preparan y muestran todas las transiciones. Para terminar se finaliza la representación.

En el anexo C se recoge la especificación de esta tarea y su método, mientras que el modelo de comportamiento completo de MEHIDA-PC puede consultarse en [González, 1996].

5.1.4.2 Evaluación

La Tabla 5.4 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.4 Evaluación de los objetivos experimentales para el caso 3

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Grande	7	Se pudo analizar todo el problema con el grado de detalle deseado, pero la falta de soporte automatizado provocó un exceso de trabajo.
Paradigmas de diseño	Objetos	4	Los analistas trataron de hacer un modelo orientado a objetos desde el principio, ajustando algunos aspectos del método.
	Dirigido por eventos	4	Esta versión no facilitaba la representación de eventos, por lo que fue necesario un esfuerzo extra en diseño.

Objetivo	Variante	Nota	Comentarios
	Creación de componentes	8	Se pudieron modelar adecuadamente las características de los componentes que había que desarrollar (elementos de actividad y controles).
Niveles de experiencia	Medios	7	Los analistas medios pudieron centrarse en el problema, aunque presentaron una tendencia hacia modelar objetos. Era necesario mejorar la notación.
	Expertos	8	Los analistas expertos mostraron una gran facilidad para estudiar el problema, independientemente de las técnicas que ya usaran con anterioridad. Era necesario mejorar cuestiones pragmáticas.
Curva de aprendizaje	Medios	7	Los analistas medios tuvieron que perder ciertos hábitos. De hecho trataban de realizar un análisis orientado a objetos.
	Expertos	8	A los analistas expertos no les supuso mucho esfuerzo el aprender esta nueva técnica.
Procedimientos de organización		5,5	Se vieron algunos problemas derivados de la tendencia de los analistas a usar conceptos de diseño. Aún así se apreciaba una evolución interesante del método.

5.1.5 Caso 4: proyecto TUTOR

5.1.5.1 Descripción

Durante el desarrollo del proyecto TUTOR se trabajó fundamentalmente en crear los elementos de actividad necesarios para que el sistema desarrollado en MEHIDA-PC pudiera realizar todas las actividades de aprendizaje solicitadas por el IMSERSO [de la Flor, 2000]. Para ello el análisis del problema fue realizado en dos etapas:

1. *Análisis de un elemento de actividad genérico*. De esta forma se construyó un modelo con las características comunes de todos los elementos de actividad.
2. *Análisis de los elementos específicos*. Tomando como punto de partida el modelo genérico, se fueron concretando los detalles de cada uno de los elementos de actividad específicos. En TUTOR se desarrollaron los siguientes elementos, de forma que cuatro primeros son elementos básicos y los restantes utilizan alguno de los elementos más básicos:
   a. *Dactilológico*. El elemento dactilológico pretende mostrar una sucesión de letras en dactilológico. La representación dactilológica será la animación de una mano que vaya deletreando lo que se le indique.
   b. *Cursor*. El elemento Cursor es un elemento que aparece en pantalla señalando a otro tipo de elementos. En principio, los elementos a los que puede señalar son de cualquier tipo, aunque lo normal es que sean zonas de edición.
   c. *Letra*. La finalidad de este elemento es representar a las letras en las actividades. Las letras aparecen en las primeras actividades de enseñanza de la lectoescritura en las que se explican éstas en distintas representaciones para que el alumno empiece a conocerlas.
   d. *Número*. El elemento número representa a los números en las actividades. Las características de este elemento son muy parecidas a las del elemento letra, pues

ambos trabajan con conceptos que están dentro de un rango y se debe poder avanzar por ellos y ambos tienen diversas representaciones.

e. *Sílaba*. El elemento sílaba sirve para mostrar sílabas en las actividades en tres posibles representaciones: en minúscula, en mayúscula o en dactilológico.

f. *Palabra*. Este elemento muestra palabras en distintas representaciones: imagen, dactilológico vídeo, lengua de signos, texto, labial, dactilológico.

g. *Palabra sílaba*. Este elemento sirve para enseñar al alumno cómo se divide una palabra en sílabas. Para ello, se debe utilizar una palabra en texto escrito con una serie de separadores entre cada una de las letras. El alumno deberá pulsar sobre los separadores para hacer que se activen o no según crea que existe en ese punto una división de sílabas.

h. *Palabra edición*. La finalidad de este elemento es la de enseñar al alumno a escribir palabras. Para permitir que el alumno pueda escribir en la Actividad se usarán casillas, una para cada letra. Para que el alumno pueda moverse entre las distintas letras, se utilizará un cursor.

i. *Conjunto imagen*. El elemento Conjunto Imagen se utiliza para enseñar a contar al alumno, y para ello muestra un cierto número de imágenes para que el alumno las cuente y diga cuántas hay.

Teniendo en cuenta estas dos etapas de análisis, en este apartado se va a mostrar el análisis de un elemento genérico, seguido por el análisis de uno de los elementos concretos.

La Figura 5.15 muestra el modelo estructural de un elemento de actividad genérico. Un elemento tiene una posición y un aspecto, está descrito mediante un lenguaje de especificación y describe un concepto (término). Además los elementos de actividad gestionan notificaciones que son enviadas a receptores, entre los que puede estar el propio elemento.

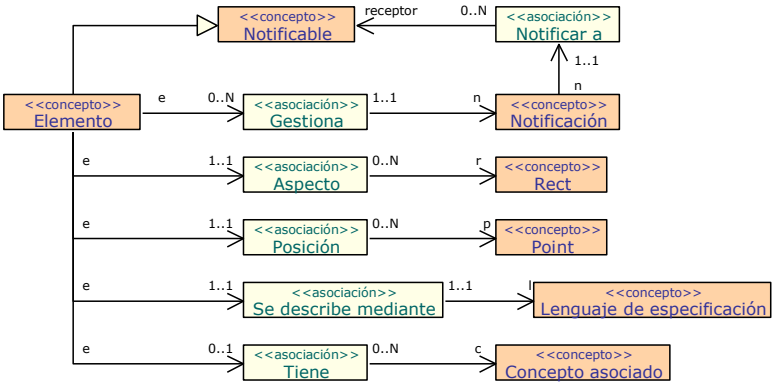


Figura 5.15 Modelo estructural de un elemento genérico de actividad en TUTOR

El modelo de comportamiento de un elemento genérico se muestra en la Figura 5.16. Puede observarse como hay una tarea (Notificar) que es subtarea de dos tareas distintas (Proporcionar servicio común) y (Gestionar notificaciones).

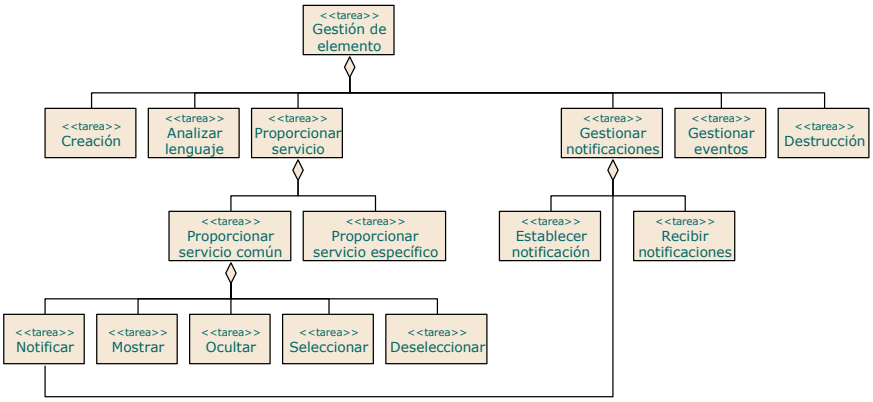


Figura 5.16 Descomposición de tareas del elemento genérico en TUTOR

El control del método principal de un elemento (Gestión de elemento) está dirigido por eventos, tal y como se recoge en la Figura 5.17.

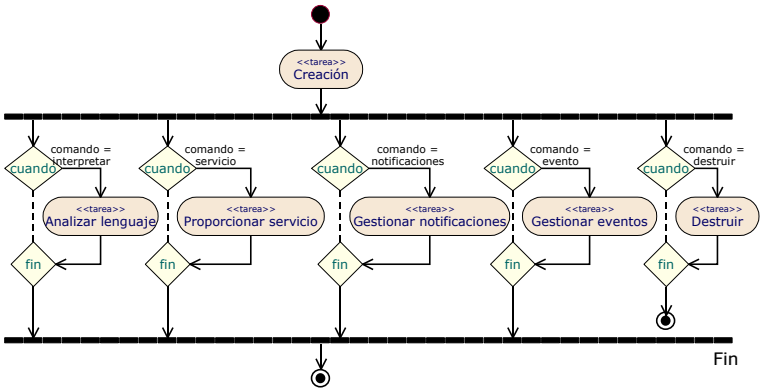


Figura 5.17 Diagrama de control principal de un elemento genérico en TUTOR

La gestión del elemento comienza con su creación, para posteriormente realizar las subtareas en función de los eventos que se produzcan en el entorno, que se corresponden con órdenes enviadas por el gestor de actividades del sistema TUTOR. El proceso termina cuando se ordena destruir el elemento.

Con esto concluye la descripción resumida del modelo de un elemento de actividad genérico. Como se ha mencionado anteriormente, en TUTOR se consideraron 9 elementos concretos (Figura 5.18). De todos ellos se va a mostrar únicamente el elemento "Palabra sílaba".

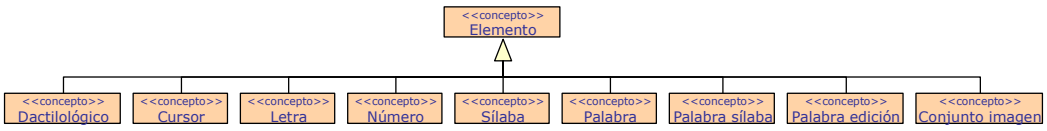


Figura 5.18 Jerarquía de elementos de actividad de TUTOR

En el modelo estructural para el elemento "Palabra sílaba" se añaden los conceptos y asociaciones representadas en la Figura 5.19: un concepto evaluación (que permite evaluar la división en sílabas), un concepto formato texto (para representar los atributos del texto) y un concepto fuente (para los atributos de texto que usa Windows), más las correspondientes relaciones.

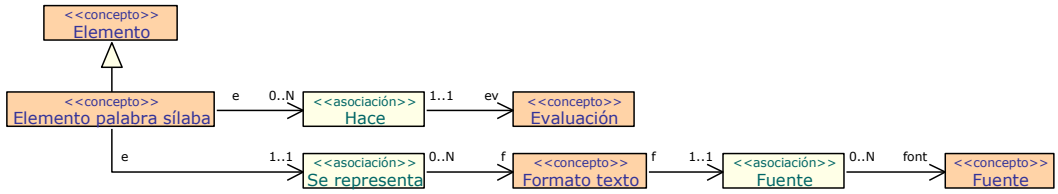


Figura 5.19 Ampliación del modelo estructural de TUTOR para "palabra sílaba"

En el caso del lenguaje de especificación se define una instancia con la gramática correspondiente:

```
INSTANCIA LS Palabra sílaba (Lenguaje de especificación [0, INFINITO])
ATRIBUTOS:
  gramática = "Identificador_elemento, Inicialmente Visible, Concepto
                [ POSICION = Coordenada X, Coordenada Y ]
                [ FUENTE = Fuente,Cursiva,Negrita,Subrayada,Ancho ]
                [ COLOR_TEXTO = R, G, B ]
                [ COLOR_FONDO = ( TRASPARENTE | OPACO, R, G, B)]
              FIN ELEMENTO PALABRA SILABA"
FIN INSTANCIA
```

En cuanto al modelo de comportamiento, los únicos cambios están dentro de la tarea "Proporcionar servicio específico", tal y como se recoge en la Figura 5.20. Se añade una nueva tarea "evaluar sílabas" encargada de comprobar si el alumno ha realizado correctamente la división en sílabas.

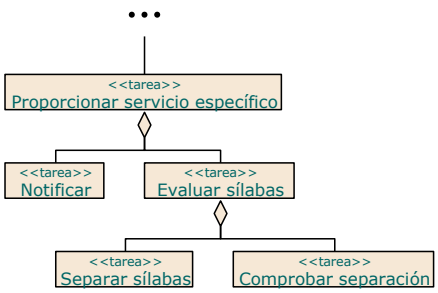


Figura 5.20 Diagrama de descomposición de tareas para "Palabra sílaba"

5.1.5.2 Evaluación

La Tabla 5.5 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.5 Evaluación de los objetivos experimentales para el caso 4

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Grande	8	Se pudo analizar todo el problema con el grado de detalle deseado, favorecido por el hecho de que se estaba ampliando el modelo de MEHIDA-PC.
Paradigmas de diseño	Objetos	8	El análisis de TUTOR era más conceptual y menos dependiente del diseño. Aún así la transición al diseño fue sencilla, dado que la arquitectura del sistema ya era conocida.
	Dirigido por eventos	7	La versión de la técnica utilizada en TUTOR ya consideraba los eventos de una forma más adecuada.
	Creación de componentes	9	Se pudieron modelar con facilidad las características de los nuevos componentes que había que desarrollar.
	Uso de componentes	8,5	Se pudo reutilizar el modelo conceptual de los componentes ya existentes.

Objetivo	Variante	Nota	Comentarios
Niveles de experiencia	Noveles	8,5	Los analistas noveles usaron la técnica para centrarse en el problema, sin pensar en la solución.
	Medios	8,5	Los analistas medios mostraron facilidad para usar el método, independientemente de las técnicas que ya usaran con anterioridad.
Curva de aprendizaje	Noveles	9	Los analistas noveles no tuvieron ningún problema en aprender a usar esta técnica.
	Medios	8,5	A los analistas medios no les supuso apenas esfuerzo el aprender esta nueva técnica.
Capacidad de ampliación de modelos		8	La técnica permitía ampliar el sistema desde el análisis del problema, lo cual permitió ahorrar muchos esfuerzos.
Procedimientos de organización		7,5	La reutilización desde el análisis mejoraba en gran medida la sensación sobre la adecuación del uso de esta técnica en el laboratorio.

5.1.6 Caso 5: sistema de música para invidentes

5.1.6.1 Descripción

Este sistema supuso abordar un dominio diferente a los que habitualmente se consideraban en el laboratorio y que además era muy desconocido por gran parte de los miembros del equipo de trabajo: la música, su representación en partitura y su informatización gracias a MIDI. Por ello el modelo estructural, aunque relativamente sencillo, tuvo una gran importancia en el desarrollo del sistema, ya que facilitó la comprensión de un dominio poco habitual.

Este modelo consta de 28 tipos, 15 conceptos y 28 asociaciones y puede dividirse en tiene dos partes bien diferenciadas. En la primera se refleja cómo se representa una partitura musical y sus componentes (Figura 5.21). Una partitura tiene una definición de compás (4/4 o similar), un fichero donde se almacena, una secuencia de notas y una posición actual. La posición actual identifica la nota actual, el instante de tiempo y la lista de notas activas en ese punto. Una nota tiene una serie de características (tono, fuerza, etc.), un instante de inicio y una duración.

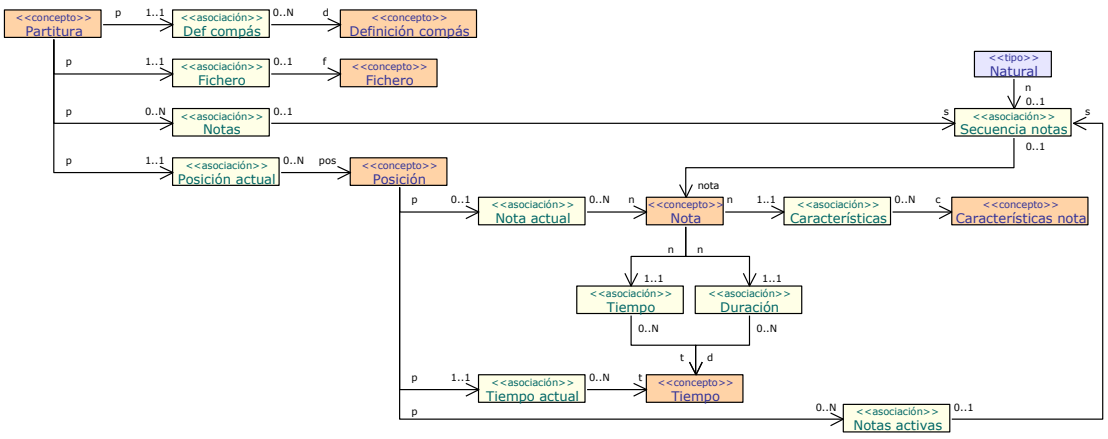


Figura 5.21 Modelo estructural del sistema musical para invidentes: la partitura

La segunda parte del modelo estructural recoge la información relativa a la propia aplicación: sistema para componer partituras, con interfaz adaptada para su uso por

personas ciegas. Esta segunda parte es menos relevante y su especificación puede verse en el Anexo C.

En cuanto al modelo de comportamiento, recoge las tareas que puede realizar el usuario con el sistema de composición musical (Figura 5.22).

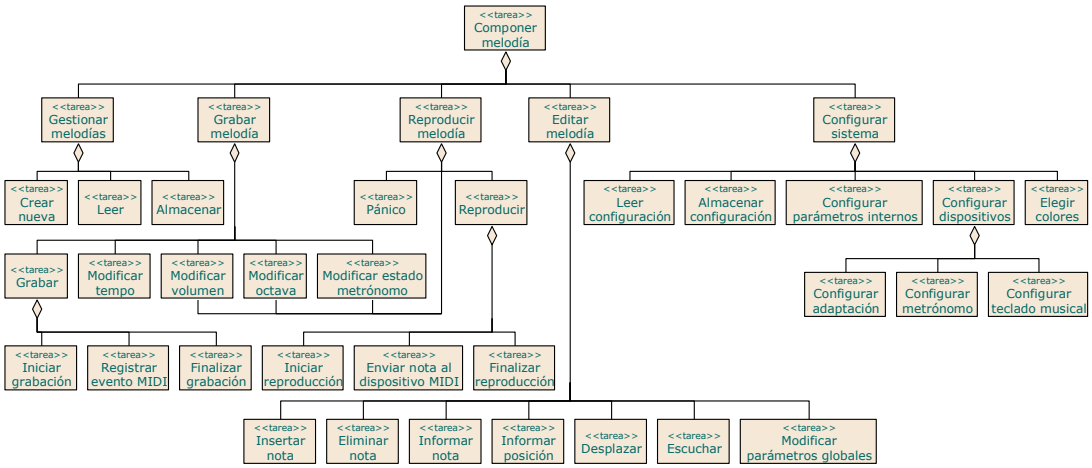


Figura 5.22 Descomposición de tareas del sistema de composición musical

5.1.6.2 Evaluación

La Tabla 5.6 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.6 Evaluación de los objetivos experimentales para el caso 5

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Medio	8	Se pudo analizar todo el problema con el grado de detalle deseado, aunque el modelo de comportamiento exigió demasiado nivel de detalle.
Paradigmas de diseño	Objetos	7,5	La transición al diseño orientado a objetos fue relativamente cómoda, aunque exigía un esfuerzo extra.
	Dirigido por eventos	6	La versión de la técnica utilizada consideraba los eventos, aunque no de la forma más adecuada.
Niveles de experiencia	Noveles	8	Los analistas noveles usaron la técnica con mayor facilidad que los métodos orientados a objetos que habían aprendido en clase.
Curva de aprendizaje	Noveles	9	Los analistas noveles aprendieron la técnica con mucha facilidad.
Procedimientos de organización		6,5	Este fue uno de los proyectos piloto para la implantación de la técnica en el laboratorio. Seguía siendo necesario trabajar más los aspectos pragmáticos del método.

5.1.7 Caso 6: proyecto DELE

5.1.7.1 Descripción

Dentro de este proyecto hay que destacar la importancia del modelo estructural, ya que en su mayor parte representa un modelo de un diccionario enciclopédico, modelo que ha sido reutilizado y modificado en proyectos posteriores de tipo similar desarrollados en el laboratorio, como por ejemplo la enciclopedia para invidentes L2000 [Casado, 2002] [García, 2002] [Hernández, 2000].

En la Figura 5.23 se recoge el modelo estructural del diccionario, que fue utilizado para crear la base de datos [Sánchez, 2001] y, posteriormente, la aplicación que permitía realizar su consulta [Rodríguez, 2002].

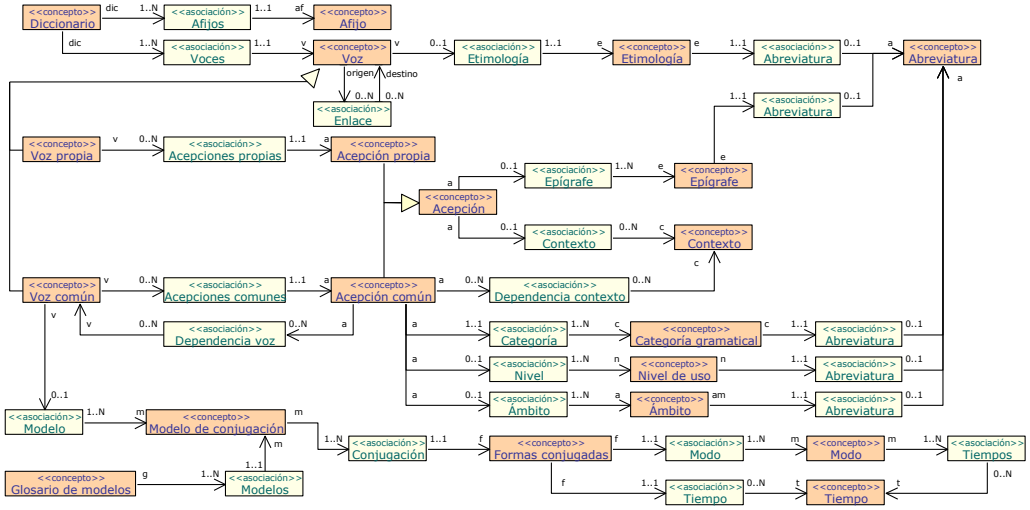


Figura 5.23 Modelo estructural del diccionario en DELE

En este modelo un diccionario contiene afijos (abreviaturas) y voces, que pueden ser comunes o propias. Las voces contienen acepciones (significados), con estructura diferente en función del tipo de voz. Por último, las voces comunes que sean verbos pueden tener asociados modelos de conjugación, en los que aparecen todos los modos y tiempos verbales.

Para la mencionada aplicación de consulta se desarrolló el correspondiente modelo de comportamiento, que consta de un total de 59 tareas y cada una tiene un único método. La Figura 5.24 muestra parte de este modelo, con el detalle completo de la búsqueda de voces. En este diagrama falta el detalle de la búsqueda de modelos de conjugación.

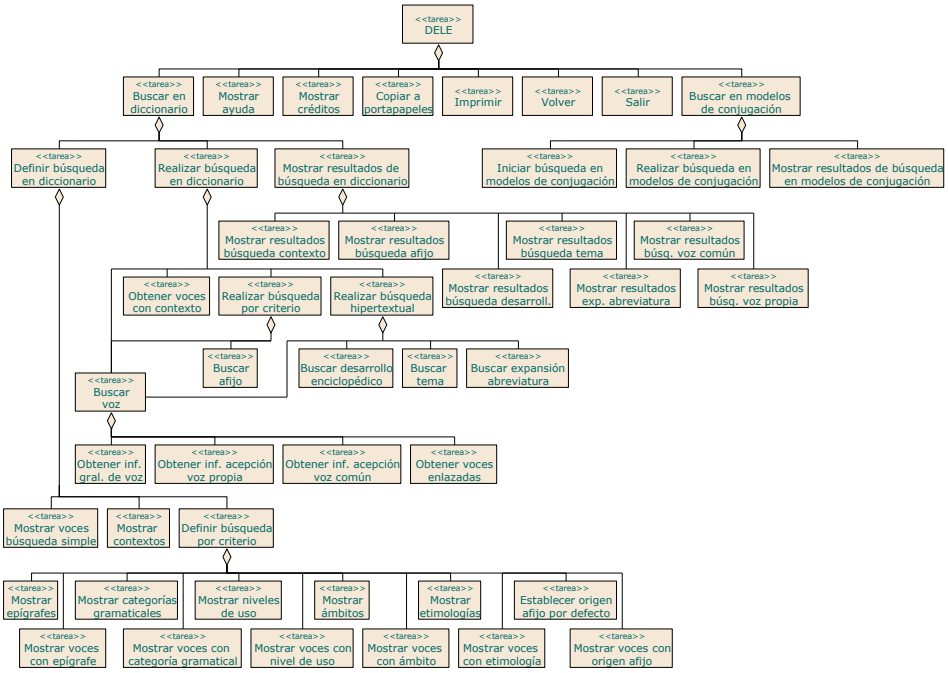


Figura 5.24 Descomposición de tareas (parcial) de la consulta en DELE

En este modelo de comportamiento puede destacarse que existe una tarea, en concreto, "Buscar voz", que se utiliza en los métodos de tres tareas diferentes: "Realizar búsqueda en diccionario", "Realizar búsqueda por criterio" y "Realizar búsqueda hipertextual".

5.1.7.2 Evaluación

La Tabla 5.7 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.7 Evaluación de los objetivos experimentales para el caso 6

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Medio	8	Se pudo analizar todo el problema con el grado de detalle deseado, aunque el modelo de comportamiento exigió demasiado nivel de detalle.
Paradigmas de diseño	Objetos	7,5	La transición al diseño orientado a objetos fue relativamente cómoda, aunque exigía un esfuerzo extra.
	Dirigido por eventos	6	La versión de la técnica utilizada consideraba los eventos, aunque no de la forma más adecuada.
Niveles de experiencia	Noveles	8	Los analistas noveles usaron la técnica con mayor facilidad que los métodos orientados a objetos que habían aprendido en clase.
Curva de aprendizaje	Noveles	9	Los analistas noveles aprendieron la técnica con mucha facilidad.
Procedimientos de organización		6,5	Este fue uno de los proyectos piloto para la implantación de la técnica en el laboratorio. Seguía siendo necesario trabajar más los aspectos pragmáticos del método.

5.1.8 Caso 7: el ahorcado – juego educativo para niños ciegos

5.1.8.1 Descripción

El modelo estructural de este caso es muy sencillo. Contiene 8 tipos, 8 conceptos y 11 asociaciones, que representan la información manejada por el sistema: la partida, la palabra que se está usando, los jugadores, los diccionarios de palabras, la configuración y las mejores puntuaciones obtenidas hasta ahora (Figura 5.25).

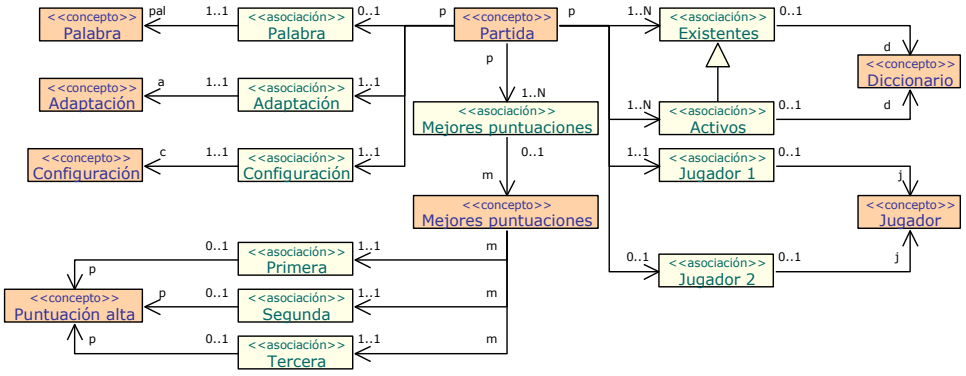


Figura 5.25 Modelo estructural del juego del ahorcado

En este modelo destaca la aparición de una clasificación de asociaciones. La asociación "Activos" (diccionarios que se usan en un momento dado) es una subasociación de "Existentes" (todos los diccionarios instalados en el sistema).

En cuanto al modelo de comportamiento, tiene 36 tareas, todas ellas con un único método (Figura 5.26).

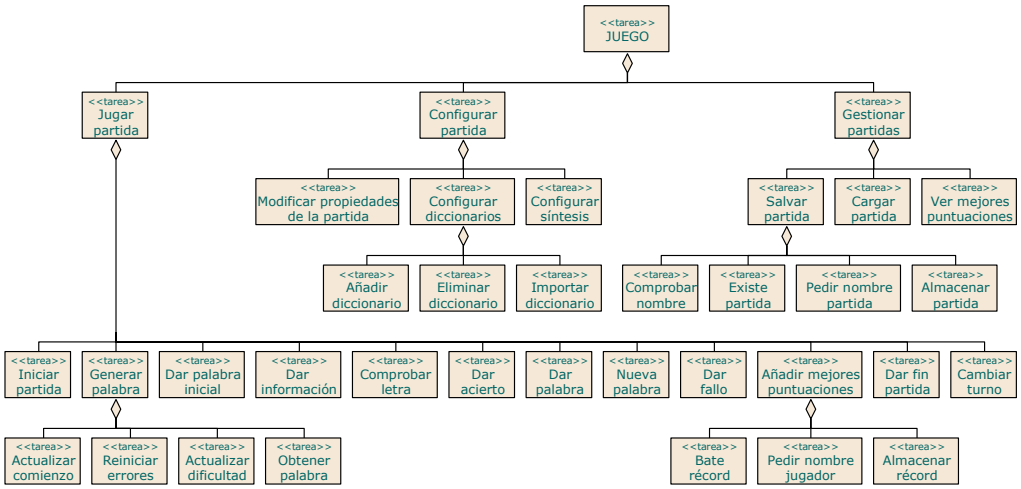


Figura 5.26 Modelo de comportamiento del juego del ahorcado

La tarea más compleja de este modelo de comportamiento es "Jugar partida", cuyo método tiene 12 subtareas directas y que presenta además un control mixto: mezcla un control dirigido por eventos que reacciona a las solicitudes del usuario (letra nueva, pedir situación o terminar partida), junto con un control explícito sobre lo que hay que hacer cada vez que el usuario pulsa una letra. El diagrama de control de este método se muestra en la Figura 5.27. Puede observarse que es un diagrama muy complejo, lo cual hace pensar que si el análisis hubiera sido realizado por personal más experto, se habrían dividido las tareas de otra forma que implicara un control más sencillo.

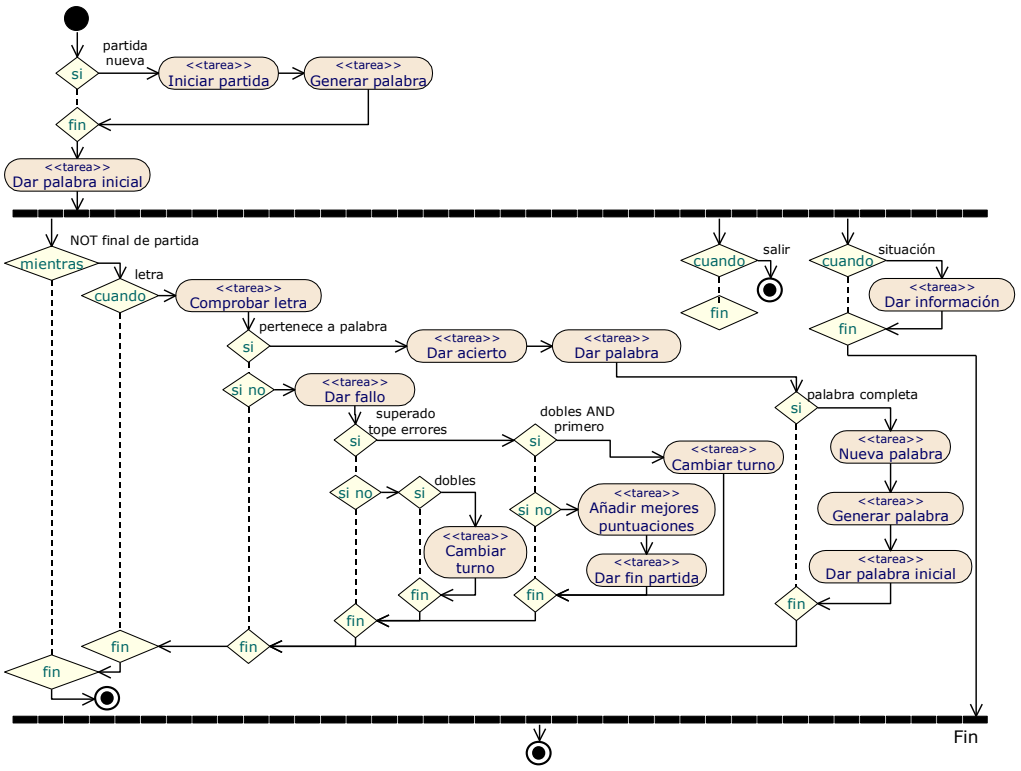


Figura 5.27 Diagrama de control de "Jugar partida" en el juego del ahorcado

5.1.8.2 Evaluación

La Tabla 5.8 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.8 Evaluación de los objetivos experimentales para el caso 7

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Pequeño	7	Se pudo analizar el problema con el grado de detalle deseado, aunque el modelo de comportamiento exigió demasiado esfuerzo para ser un problema pequeño.
Paradigmas de diseño	Objetos	8	No hubo problemas de transición al diseño y pudo decidirse la mejor forma de hacerlo (orientación a objetos) una vez conocido el problema.
	Dirigido por eventos	7	La versión de la técnica utilizada ya consideraba los eventos, aunque no de la forma más adecuada.
Niveles de experiencia	Noveles	8,5	Los analistas noveles usaron correctamente la técnica para centrarse en el problema, sin pensar en la solución.
Curva de aprendizaje	Noveles	9	Los analistas noveles aprendieron en poco tiempo a usar esta nueva técnica.
Procedimientos de organización		8	Los resultados de este proyecto demostraban el éxito progresivo de la implantación del método en el laboratorio.

5.1.9 Caso 8: navegador para personas ciegas

5.1.9.1 Descripción

La aportación principal de este proyecto fue el modelo estructural de una página Web, que fue estudiado en detalle con el fin de poder determinar la forma más adecuada de transmitir su contenido a una persona ciega (Figura 5.28). A este modelo estructural se le añadieron posteriormente conceptos y asociaciones necesarios para definir el modelo de comportamiento.

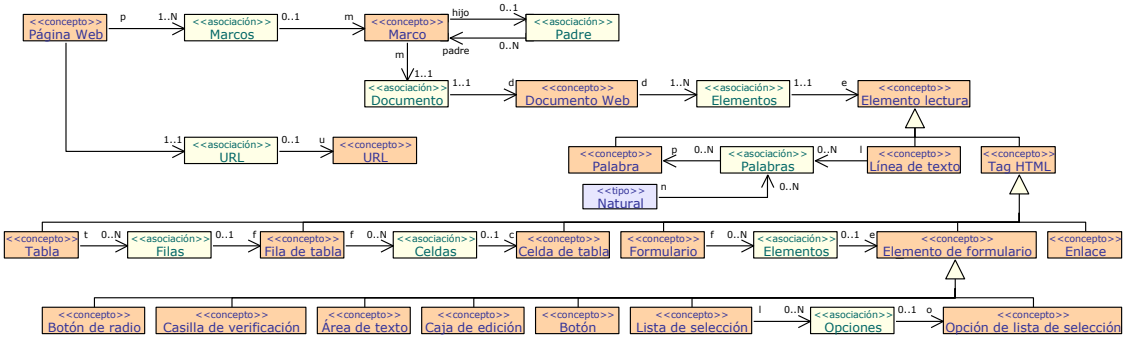


Figura 5.28 Modelo conceptual de un documento Web

A diferencia de los casos de estudio anteriores, este modelo estructural presenta una jerarquía de clasificaciones en tres niveles: elementos de lectura, marcadores de HTML y elementos de formulario.

A este modelo estructural se añadieron algunos conceptos que permitieron modelar el comportamiento de un navegador para personas ciegas, cuya descomposición de tareas se muestra en la Figura 5.29.

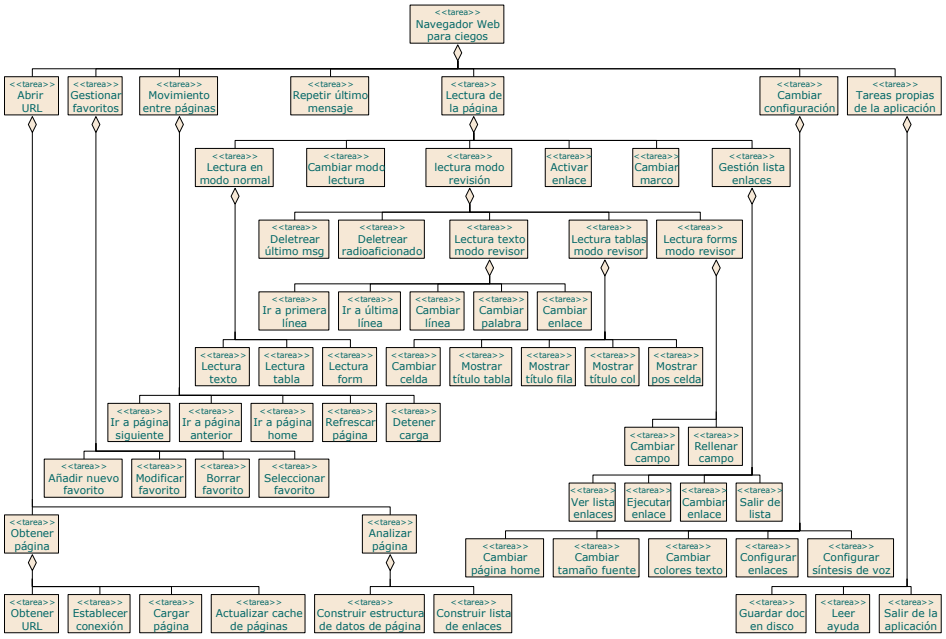


Figura 5.29 Descomposición de tareas del navegador Web

5.1.9.2 Evaluación

La Tabla 5.9 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.9 Evaluación de los objetivos experimentales para el caso 8

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Pequeño	7,5	Se pudo analizar el problema con el grado de detalle deseado, aunque el modelo de comportamiento exigió mucho esfuerzo en relación con el problema.
Paradigmas de diseño	Objetos	8	No hubo problemas de transición al diseño y pudo decidirse la mejor forma de hacerlo (orientación a objetos) una vez conocido el problema.
	Dirigido por eventos	7	La versión de la técnica utilizada ya consideraba los eventos, aunque no de la forma más adecuada.
	Uso de componentes	9	Se pudo analizar el problema de forma independiente al uso posterior de componentes para parte de la implementación.
Niveles de experiencia	Noveles	8	Los analistas noveles usaron la técnica adecuadamente, aunque el grado de detalle fue excesivo en algunas partes del modelo.
Curva de aprendizaje	Noveles	9	Los analistas noveles no tuvieron ningún problema en aprender a usar esta nueva técnica.
Procedimientos de organización		7,5	Respecto al proyecto anterior (ahorcado) se rebajaron un tanto las expectativas, aunque los resultados seguían siendo positivos.

5.1.10 Caso 9: proyecto GESTLAB

5.1.10.1 Descripción

Dentro de este proyecto la parte más relevante fue el modelo estructural que representa todos los elementos que se deben gestionar dentro de laboratorio. Este modelo consta de 13 tipos, 39 conceptos, 30 asociaciones, 9 clasificaciones de concepto y 3

clasificaciones de asociación, por lo que se procederá a mostrarlo en partes. La esencia del modelo es la representación de un equipo (ordenador) y los elementos que contiene (Figura 5.30).

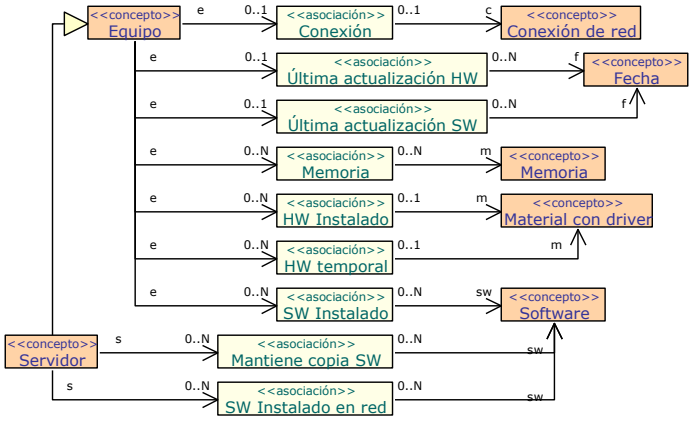


Figura 5.30 Un equipo y sus componentes en GESTLAB

Para un equipo se recoge información sobre su conexión a red, fechas de últimas actualizaciones y componentes (memoria, hardware y software). Si además es un servidor, entonces puede mantener copias de software o tenerlas instaladas en red, para que en otros ordenadores sólo haga falta instalar la parte cliente del programa.

En el modelo estructural de GESTLAB aparecen asociaciones con atributos, como por ejemplo, la relación entre un equipo y el software que se instala en ese equipo:

```
ASOCIACIÓN SW Instalado
DESCRIPCIÓN:
  Relaciona un equipo con el software instalado.
ORIGEN:
  e : Equipo (0:N)
DESTINO:
  sw : Software (0:N)
TIPO:
  no función (parcial)
ATRIBUTOS:
  inactivo : Booleano = falso "Indica si el software está inactivo en el sistema."
  path local : Cadena "Indica el path donde está instalado el software en el
  equipo"
FIN ASOCIACIÓN
```

Además de los conceptos principales reflejados en la figura anterior, en el modelo estructural de GESTLAB existen varias jerarquías de clasificación: tipos de soportes software, tipos de modelos de software, tipos de hardware concreto y tipos de modelos a los que pertenece cada componente hardware. A modo de ejemplo, la Figura 5.31 recoge las jerarquías de hardware y modelos de hardware, señalándose que todo material con *driver* tiene asignado un modelo.

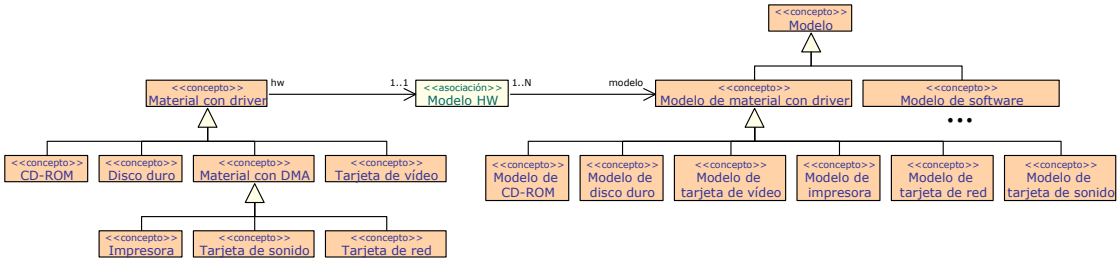


Figura 5.31 Jerarquías de hardware y sus modelos en GESTLAB

La existencia de estas jerarquías de hardware y modelos, junto con la asociación entre ambos, implica que en el modelo estructural de GESTLAB aparezcan también clasificaciones de asociaciones. Así, la Figura 5.32 muestra la jerarquía de la asociación "Modelo HW".

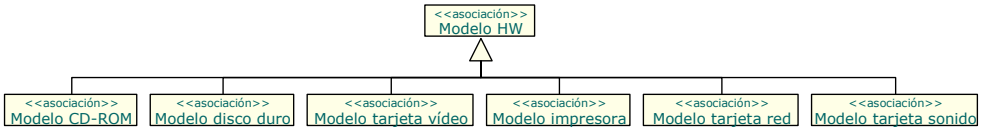


Figura 5.32 Una jerarquía de asociaciones en GESTLAB

En contraste, el modelo de comportamiento de GESTLAB fue muy sencillo, ya que simplemente había que hacer altas, consultas, modificaciones y bajas sobre toda la información recogida en el modelo estructural.

Cabe recordar que GESTLAB fue implementado con una arquitectura cliente – servidor. El modelo estructural aquí presentado sirvió para desarrollar la base de datos que utiliza el servidor del sistema [Moya, 1999], mientras que el modelo de comportamiento fue el punto de partida de la aplicación cliente [García, 2000].

5.1.10.2 Evaluación

La Tabla 5.10 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.10 Evaluación de los objetivos experimentales para el caso 9

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Pequeño	8	Se pudo analizar la parte estática con el grado de detalle deseado, aunque en la parte dinámica no se llegó a usar por su sencillez (gestión de datos).
Paradigmas de diseño	Objetos	8	No hubo problemas de transición al diseño y pudo decidirse la mejor forma de hacerlo (orientación a objetos) una vez conocido el problema.
	Dirigido por eventos	7	La versión de la técnica utilizada consideraba los eventos, aunque no de forma totalmente adecuada.
	Cliente – Servidor	8	El modelo de análisis permitió concentrarse en el problema sin preocuparse de la arquitectura final.
	Uso de componentes	9	Se pudo analizar el problema de forma independiente al uso posterior de componentes para parte del trabajo (conexiones cliente → servidor → BD).
Niveles de experiencia	Noveles	8	Los analistas noveles usaron correctamente la técnica para centrarse en el problema, sin pensar en exceso en la solución.
	Medios	9	Los analistas medios usaron la técnica de forma adecuada para estudiar el problema.
Curva de aprendizaje	Noveles	9	Los analistas noveles no tuvieron ningún problema en aprender a usar esta nueva técnica.
	Medios	9	Los analistas medios aprendieron fácilmente la técnica.
Procedimientos de organización		8	Este proyecto pretendía servir de campo de pruebas para técnicas de diseño que luego se utilizaron en proyectos reales (como ALBOR). La utilización de SETCM para el análisis permitió centrarse sin problemas en las distintas variantes del diseño.

5.1.11 Caso 10: proyecto ALBOR

5.1.11.1 Descripción

En esta primera iteración del proyecto ALBOR se desarrolló el sistema inteligente distribuido, que es el núcleo de la red que se completó en fases posteriores. En primer lugar se va a mostrar, como ejemplo de diagrama de tipos, el conjunto de los tipos derivados de cadena de su modelo estructural (Figura 5.33).

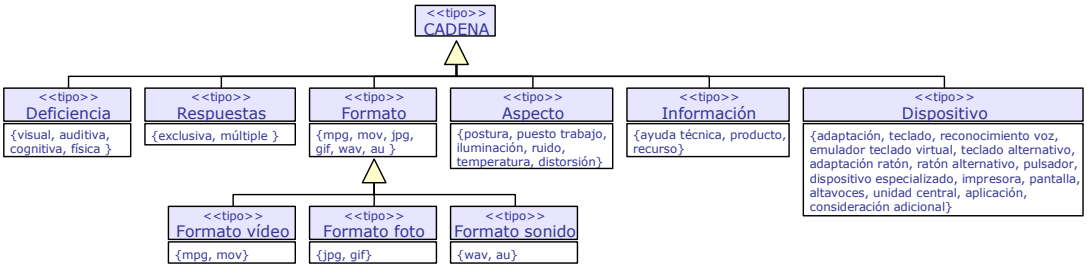


Figura 5.33 Los tipos derivados de cadena de ALBOR

Estos tipos se utilizan en los atributos de los conceptos y asociaciones del modelo estructural. Este modelo queda recogido en la Figura 5.34, en la que, a diferencia de los diagramas de casos de estudio anteriores, se muestran los atributos de los conceptos y asociaciones.

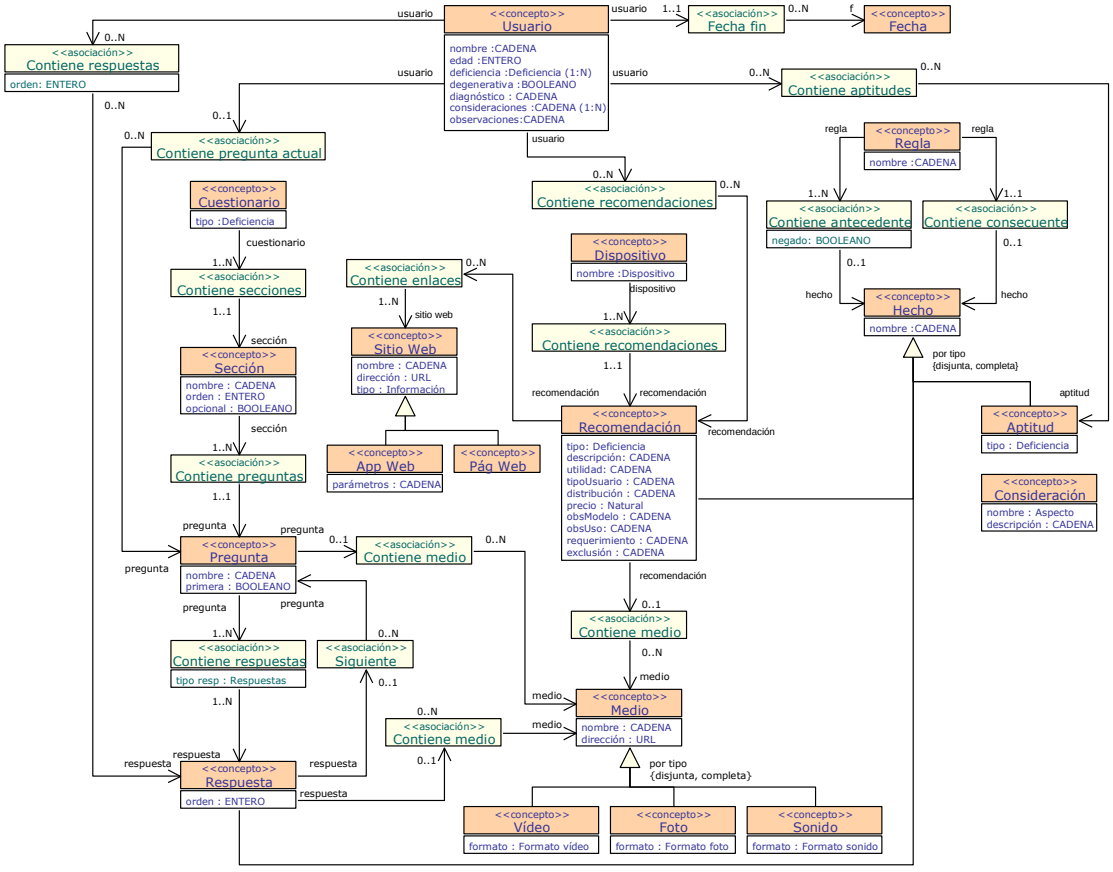


Figura 5.34 Conceptos y asociaciones del modelo estructural de ALBOR

Dentro de este modelo estructural hay que destacar que hay muchos conceptos y asociaciones cuyas instancias son conocidas por el sistema:

- Por una parte se conocen todos los cuestionarios, secciones, preguntas y respuestas, así como la pregunta siguiente para cada respuesta.
- Por otro lado se conocen todas las reglas, con sus antecedentes y consecuentes, que son aptitudes, respuestas o recomendaciones.
- Por último se conocen todos los dispositivos, medios y sitios Web.

En todas estas instancias reside el conocimiento experto del sistema. Por eso no es de extrañar que el modelo de comportamiento de ALBOR sea muy sencillo (Figura 5.35), ya que únicamente contiene 16 tareas.

En este modelo de comportamiento hay que señalar que no existe una tarea que sea "padre" de todas las demás. Esto es así ya que las tareas principales (iniciar sesión, identificar usuario, analizar usuario y mostrar informe) son relativamente independientes entre sí. Esta distribución de tareas fue una de las razones por las que en la fase de diseño se decidió definir una arquitectura distribuida basada en la colaboración de agentes inteligentes.

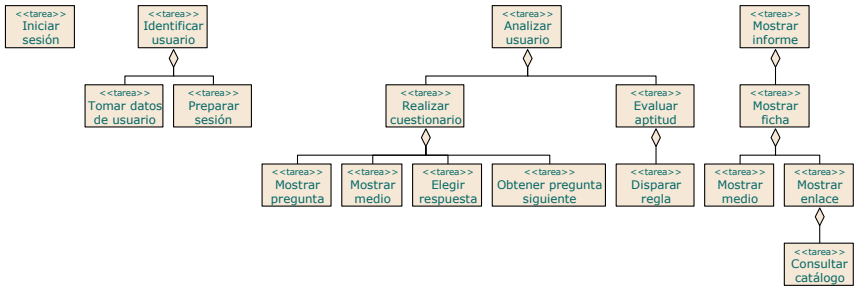


Figura 5.35 Descomposición de tareas de ALBOR

5.1.11.2 Evaluación

La Tabla 5.11 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.11 Evaluación de los objetivos experimentales para el caso 10

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Grande	8,5	Se pudo analizar el problema con el grado de detalle deseado, representándose incluso las reglas y cuestionarios del sistema.
Paradigmas de diseño	Objetos	9	No hubo problemas de transición al diseño y pudo decidirse la mejor forma de implementar los agentes (orientación a objetos) una vez conocido el problema.
	Dirigido por control	8,5	Hay grandes partes del problema que son dirigidas por el control, como la evaluación del cuestionario, y pudieron ser modeladas de forma correcta.
	Dirigido por eventos	8	La versión de la técnica utilizada consideraba de forma más adecuada los eventos, para aquellas partes del problema que lo requerían
	Cliente – Servidor	9	El modelo de análisis permitió concentrarse en el problema sin tener que preocuparse de la arquitectura final del problema.
	Agentes inteligentes	8,5	Se pudo analizar el problema sin pensar en posibles soluciones y una vez conocido se decidió desarrollar un sistema multiagente.

Objetivo	Variante	Nota	Comentarios
Niveles de experiencia	Expertos	9,5	Los analistas expertos pudieron estudiar el problema usando el vocabulario propio del dominio y sin verse influenciados por el futuro diseño del sistema.
Curva de aprendizaje	Expertos	9	Los analistas expertos no tuvieron ningún problema en aprender a usar esta nueva técnica.
Procedimientos de organización		9	Este proyecto marcó el punto de decisión sobre la aplicación de la técnica en el laboratorio. A partir de aquí SETCM forma parte del catálogo de procesos que se usan en el trabajo cotidiano.

5.1.12 Caso 11: proyecto ALBOR-II

5.1.12.1 Descripción

Dentro de este caso se hicieron dos ampliaciones de ALBOR:

1. Por un lado se desarrolló el sistema de mantenimiento, que permite actualizar los cuestionarios y reglas del sistema de forma remota.
2. Por otro lado se desarrolló la "Red ALBOR", un espacio virtual en Internet que sirve de herramienta básica a los profesionales de la evaluación de personas con discapacidad y la adaptación de su acceso al ordenador.

En ambos casos se partió del modelo conceptual de ALBOR para realizar los cambios desde el análisis. Por cuestiones de espacio, en este apartado se va a mostrar sólo la primera parte (el sistema de mantenimiento). Los detalles sobre la red ALBOR pueden encontrarse en [Alonso et al., 2001] [Frutos, 2001] [González, 2001].

En el modelo estructural se tuvieron que añadir nuevos conceptos y asociaciones para poder reflejar de forma adecuada las actuaciones de mantenimiento realizadas por usuarios del sistema (Figura 5.36): informes sobre elementos ya existentes o bien fichas para dar de alta nuevos elementos.

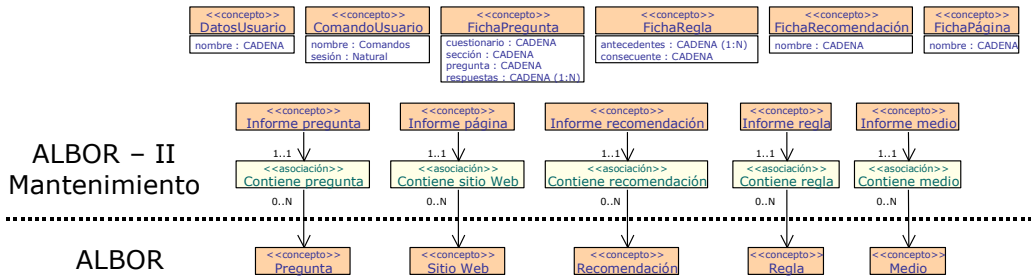


Figura 5.36 Nuevos elementos del modelo estructural de ALBOR-II: mantenimiento

En cuanto al modelo de comportamiento, se añadieron las tareas de actualización del sistema mostradas en la Figura 5.37.

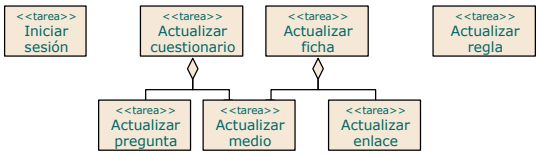


Figura 5.37 Tareas de mantenimiento de ALBOR-II

5.1.12.2 Evaluación

La Tabla 5.12 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.12 Evaluación de los objetivos experimentales para el caso 11

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Grande	9	Se pudo analizar correctamente el problema.
Paradigmas de diseño	Objetos	9	No hubo problemas de transición a programas OO.
	Dirigido por control	9	El análisis de la parte dirigida por el control fue realizado sin problemas.
	Dirigido por eventos	9	La versión de la técnica utilizada consideraba de forma adecuada los eventos.
	Cliente – Servidor	9	El modelo de análisis permitió concentrarse en el problema aunque la arquitectura fuera cliente - servidor.
	Agentes inteligentes	9,5	Se pudo analizar correctamente el problema e identificar posteriormente nuevos agentes.
Niveles de experiencia	Medios	9,5	Los analistas medios pudieron estudiar fácilmente las ampliaciones de los modelos.
	Noveles	9,5	Los analistas noveles colaboraron con éxito.
Curva de aprendizaje	Medios	9	Los analistas medios aprendieron fácilmente su uso.
	Noveles	9,5	Los analistas noveles aprendieron muy fácilmente el uso del método.
Capacidad de ampliación		9,5	Fue muy sencillo ampliar el sistema desde el análisis.
Procedimientos de organización		9,5	La aplicación continuada de SETCM en el proyecto resultó ser muy positiva.

5.1.13 Caso 12: proyecto Estenotipia

5.1.13.1 Descripción

Dentro del proyecto de estenotipia se realizaron tres aplicaciones [Picazo, 2001]:

1. *Aplicación de estenotipia*. Aplicación principal del proyecto, encargada de recoger las pulsaciones realizadas sobre la máquina de estenotipia, almacenar en memoria el texto al que corresponden, mostrarlo en pantalla y finalmente guardarlo en un fichero después de haber sido tratado, expandiendo abreviaturas y traduciendo códigos especiales para su posterior revisión por parte del estenotipista, revisión que podrá realizarse con la misma aplicación.
2. *Aplicación de códigos especiales*. Aplicación encargada de la creación, consulta y modificación de diccionarios de códigos especiales que podrán ser usados durante una sesión de trascripción.
3. *Aplicación de comunicaciones*. Esta aplicación permitirá enviar al exterior lo que se está escribiendo en la máquina de estenotipia, sin tratar abreviaturas pero sí códigos especiales. Este envío se realizará por uno de los puertos serie de la máquina de estenotipia, para poder recibirlas en un ordenador que dispondrá de un software adicional, responsable de comunicarse con, por ejemplo, un sistema de subtitulado.

En este apartado se va a describir únicamente el análisis de la primera aplicación: el sistema de estenotipia. Su modelo estructural es relativamente sencillo, con 12 conceptos y 18 asociaciones, tal y como muestra la Figura 5.38.

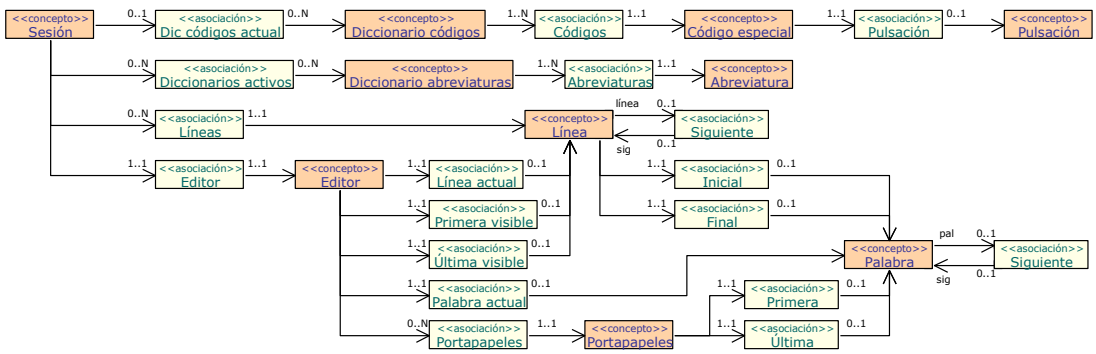


Figura 5.38 Modelo estructural del proyecto Estenotipia

Los elementos principales son la sesión (documento activo), sus líneas y palabras. En un segundo plano está el editor, con todos los parámetros necesarios para su funcionamiento. Por último se tienen diccionarios de códigos especiales (pulsaciones rápidas) y de abreviaturas.

Respecto al modelo de comportamiento, en este proyecto se decidió realizar un análisis exhaustivo del mismo, incluyendo las tareas del editor. Esto es así ya que el sistema debía implementarse para poder funcionar en un ordenador con pocos recursos (sistema operativo MS-DOS) y no se podrían utilizar librerías de programación ni para la interfaz de usuario ni para el editor.

Este análisis exhaustivo provocó la definición de un gran número de tareas, de las cuales la Figura 5.39 muestra sólo los primeros niveles de descomposición. En total hay más de 290 tareas, siendo muchas de ellas muy cortas (con métodos de una o dos sentencias). El modelo de comportamiento completo puede consultarse en [Picazo, 2001].

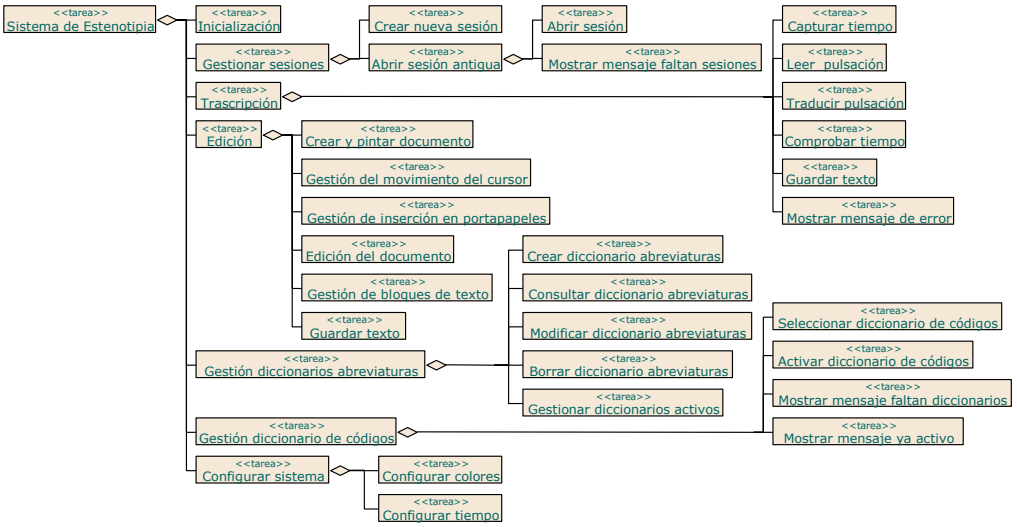


Figura 5.39 Descomposición de tareas (parcial) de Estenotipia

5.1.13.2 Evaluación

La Tabla 5.13 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.13 Evaluación de los objetivos experimentales para el caso 12

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Medio	9,5	Se pudo analizar el problema con un gran nivel de detalle en la parte de comportamiento.
Paradigmas de diseño	Objetos	9,5	No hubo problemas de transición al diseño OO.
	Dirigido por control	9,5	La notación textual permitió reflejar cómodamente todos los métodos de tarea.
Niveles de experiencia	Noveles	9	Los analistas noveles usaron correctamente la técnica para centrarse en el problema.
Curva de aprendizaje	Noveles	9,5	Los analistas noveles no tuvieron ningún problema en aprender a usar esta nueva técnica.
Procedimientos de organización		9'5	La técnica ya estaba instaurada en el laboratorio.

5.1.14 Caso 13: proyecto Subtítulos

5.1.14.1 Descripción

El análisis del sistema de subtítulos fue relativamente sencillo, dado que las mayores dificultades del problema estaban en cuestiones relacionadas con el diseño e implementación.

El modelo estructural consta de 3 tipos, 9 conceptos y 8 asociaciones (tres de ellas ternarias para representar secuencias de elementos), y queda reflejado en la Figura 5.40.

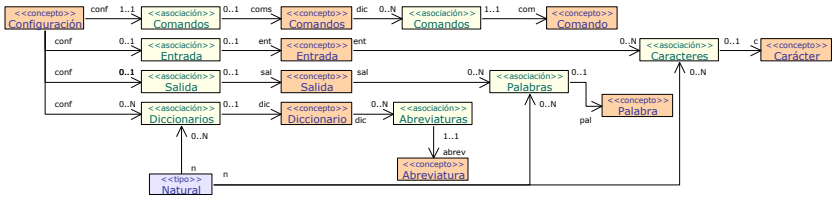


Figura 5.40 Modelo estructural del sistema de subtítulos

El modelo de comportamiento, por otro lado, tiene un total de 52 tareas, muchas de ellas muy sencillas, organizadas según el diagrama de descomposición de tareas que recoge la Figura 5.41.

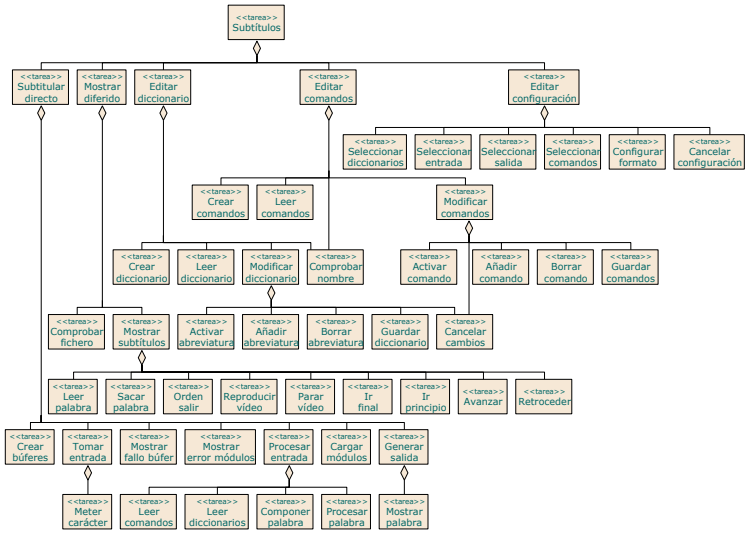


Figura 5.41 Descomposición de tareas del sistema de subtítulos

En este diagrama de descomposición de tareas se puede observar que hay dos tareas ("comprobar nombre" y "cancelar cambios") que son utilizadas desde más de un método de tarea.

5.1.14.2 Evaluación

La Tabla 5.14 recoge la valoración realizada de los objetivos experimentales.

Tabla 5.14 Evaluación de los objetivos experimentales para el caso 13

Objetivo	Variante	Nota	Comentarios
Problemas de distinto tamaño	Medio	9,5	Se pudo analizar el problema con un gran nivel de detalle en la parte de comportamiento.
Paradigmas de diseño	Objetos	9,5	No hubo problemas de transición al diseño OO.
	Dirigido por control	9,5	La notación textual permitió reflejar cómodamente todos los métodos de tarea.
	Dirigido por eventos	9	La notación permitió reflejar adecuadamente las partes del problema dirigidas por eventos.
Niveles de experiencia	Noveles	9	Los analistas noveles usaron correctamente la técnica para centrarse en el problema.
Curva de aprendizaje	Noveles	9,5	Los analistas noveles no tuvieron ningún problema en aprender a usar esta nueva técnica.
Procedimientos de organización		9'5	La técnica ya estaba instaurada en el laboratorio.

5.1.15 Resultados globales de la experimentación

El número de casos de estudio, siendo elevado desde el punto de vista procedimental, no permite realizar un estudio estadístico completo de los resultados de la evaluación. Sin embargo sí ofrece resultados concluyentes al analizar las tendencias en las valoraciones obtenidas según avanzaba el tiempo y se mejoraban aspectos de la técnica.

En primer lugar, la Figura 5.42 muestra las tendencias de valoración de la *adecuación del método a problemas de distinto tamaño*. Puede apreciarse como durante la evolución de la técnica propuesta ha mejorado la evaluación de este objetivo por parte de los participantes de cada proyecto, independientemente del tipo de problema. También puede observarse que las notas más altas son para problemas medios y grandes, dado que el análisis de problemas pequeños, que no tendrían ni que analizarse de forma detallada, produce una sobrecarga de trabajo.

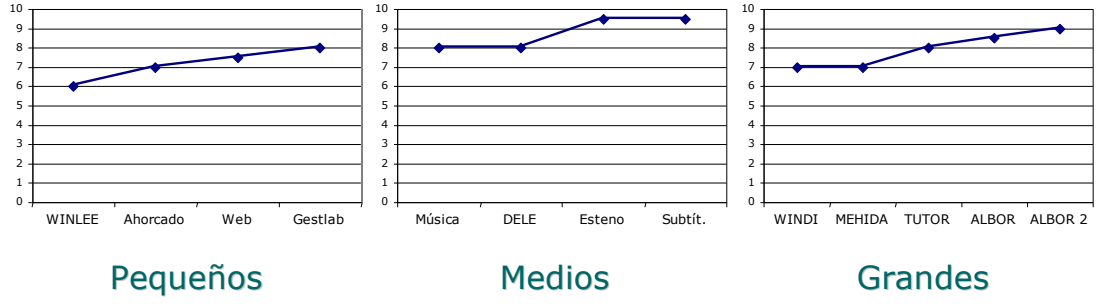


Figura 5.42 Tendencias en la evaluación de la técnica para tamaños de problemas

El segundo objetivo aborda los paradigmas de diseño y su relación con el análisis. Éste es el objetivo con mayor número de variantes (8 en total), por lo que interesa mostrar los resultados obtenidos en cada una de ellas (Tabla 5.15). En esta tabla se puede

observar la evolución de la calificación obtenida por SETCM al ser aplicada a sistemas con diseños de tipos diferentes.

Tabla 5.15 Evaluación del criterio de paradigmas de diseño

Variante	WINDI	WINLEE	MEHIDA	Música	DELE	TUTOR	Ahorcado	WWW	Gestlab	ALBOR	ALBOR II	Esteno	Subtit.
Estructurado	8												
Objetos		6	4	7,5	7,5	8	8	8	8	9	9	9,5	9,5
Cliente – servidor									8	9	9		
Agentes inteligentes										8,5	9,5		
Dirigido por control										8,5	9	9,5	9,5
Dirigido por eventos	4	4	4	6	6	7	7	7	7	8	9		9
Uso de componentes		8				8,5		9	9				
Creación de componentes			8			9							

Dentro de este objetivo es interesante representar las tendencias de las dos variantes con más casos de estudio: diseño orientado a objetos y dirigido por eventos (Figura 5.43).

En el caso de problemas cuyo diseño fue orientado a objetos, puede apreciarse la tendencia creciente en las valoraciones, con un valor inicial relativamente alto, y la excepción del proyecto MEHIDA-PC, ya que sus analistas no siguieron el método propuesto sino que intentaron aplicar un análisis tradicional orientado a objetos usando las notaciones propuestas.

Respecto a los problemas con control dirigido por eventos, puede apreciarse que su análisis fue difícil al principio, obteniéndose evaluaciones muy bajas. Esto se debía a que las primeras versiones no incorporaban los mecanismos de control necesarios. El esfuerzo realizado para incorporar la gestión de eventos de forma natural en el método propuesto ha llevado a la mejora que puede apreciarse en la evaluación de este aspecto.

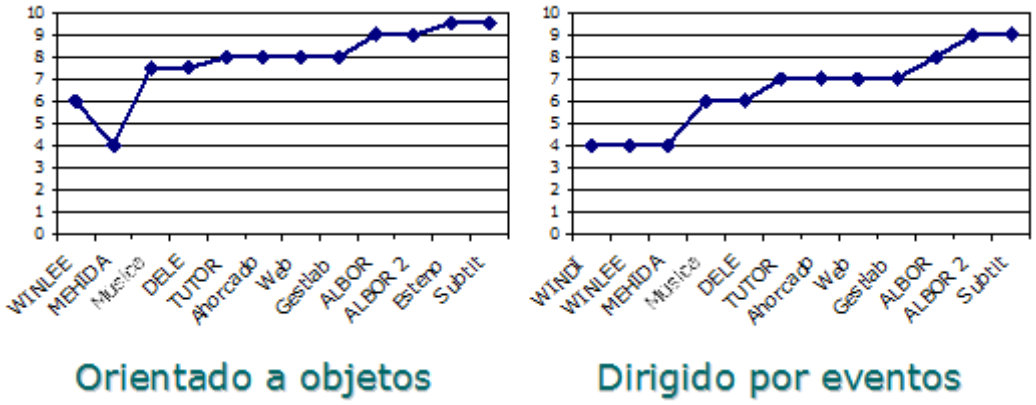


Figura 5.43 Tendencias de la evaluación respecto a paradigmas de diseño

Cabe destacarse las altas calificaciones obtenidas en las dos fases del proyecto ALBOR (con notas entre 8.5 y 9.5), un sistema que resuelve un problema complejo mediante un diseño híbrido basado en agentes inteligentes, con arquitectura cliente – servidor y programación orientada a objetos.

Los objetivos tercero (nivel de experiencia de los analistas) y cuarto (curva de aprendizaje) están centrados en los aspectos pragmáticos del método propuesto y por tanto procede su análisis conjunto. En la Figura 5.44 se muestra la tendencia de su evaluación, considerándose como valor de cada caso la media de los distintos tipos de analistas que trabajaron en ese proyecto (nótese que el rango de valores del gráfico va de 5 a 10, para facilitar la distinción entre las dos curvas).

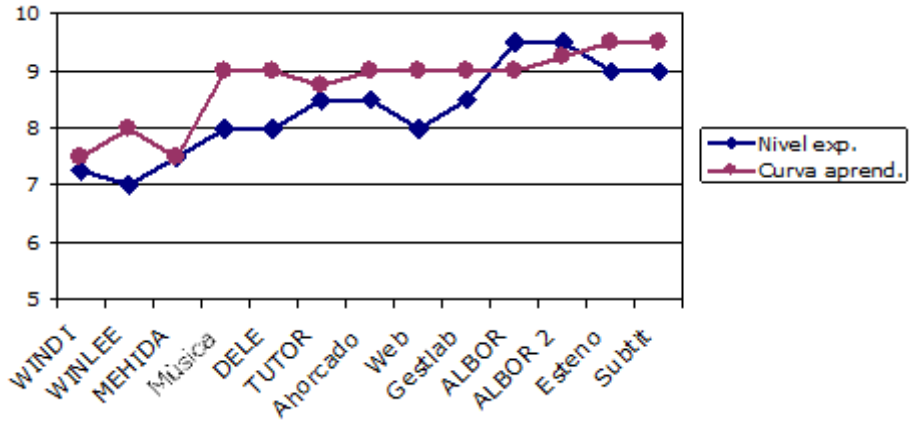


Figura 5.44 Tendencias en la valoración de los objetivos 3 y 4

En primer lugar puede comprobarse como, casi desde el principio, ambos objetivos han obtenido una alta valoración, lo que demuestra el pragmatismo de aplicación de la técnica propuesta. En cuanto a las diferencias entre los dos objetivos, puede observarse cómo, en general, ha sido un método más fácil de aprender que de aplicar, aunque siempre por escasa diferencia. Esto se debe a la sencillez de las notaciones propuestas y al ambiente académico en el que se han aplicado.

Por último debe estudiarse el objetivo de la adecuación del método propuesto para su integración dentro de los procedimientos de una organización. La tendencia de la evaluación de este objetivo se muestra en la Figura 5.45, acompañada por la media del resto de los objetivos aplicables en cada caso.

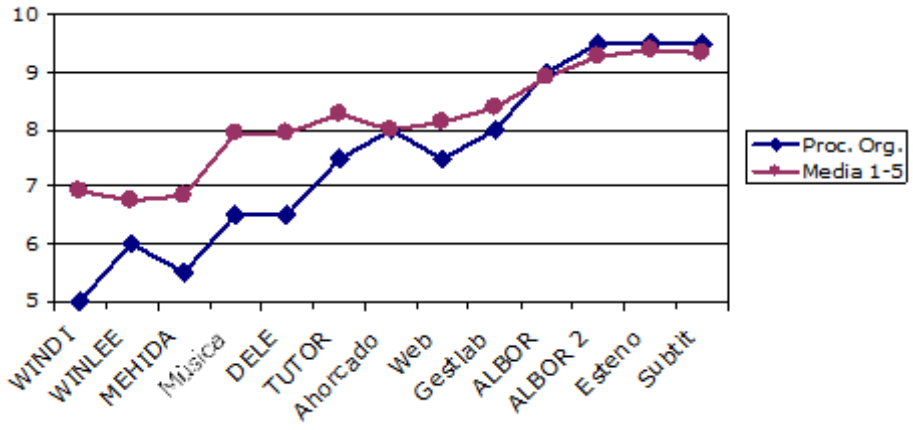


Figura 5.45 Evaluación del objetivo 6 comparada con la media del resto

Este objetivo experimental es muy importante dado que representa la opinión subjetiva de los actores participantes en cada proyecto, así como de sus gestores, que

normalmente tienen una visión más global y más centrada en el cumplimiento de objetivos (tiempos, costes, recursos, etc.).

Puede observarse cómo son curvas muy parecidas, lo que demuestra cómo el último objetivo es realmente una consecuencia del resto. En la mayoría de los casos, cuanto mejor es la media de los objetivos 1 a 5, mejor es la valoración de la adecuación de la técnica para su uso en el laboratorio.

En resumen, se puede constatar cómo la evaluación de los objetivos experimentales es creciente según ha ido evolucionando el método propuesto, obteniéndose valoraciones muy altas en los últimos casos de estudio.

5.2 HERRAMIENTA DE MODELADO CON VERIFICACIÓN AUTOMÁTICA

Consciente desde un principio de la importancia del soporte de herramientas para la pragmática del proceso de análisis, en paralelo con el desarrollo de esta Tesis se ha producido la construcción de dos versiones diferentes de una herramienta encaminada a crear y verificar modelos de análisis de la forma más sencilla posible.

La construcción de esta herramienta ha sido posible en gran medida debido a la base formal del método, lo que ha permitido especificar de forma no ambigua el comportamiento que debería tener la herramienta.

La **primera herramienta** [Andrés, 2001] comenzó a desarrollarse al mismo tiempo que empezaron a tener éxito las primeras ideas de un análisis de problemas independiente de técnicas de diseño.

Esta herramienta se desarrolló en lenguaje C++, en el entorno C++ Builder de Borland y utilizaba una base de datos en formato Microsoft Access para almacenar los modelos de análisis.

El énfasis principal del programa consistía en crear los modelos de forma gráfica para luego ir completando la información de cada elemento mediante cuadros de diálogo sencillos de manejar.

En aquel momento el método consistía en cuatro modelos (conceptos y relaciones, descomposición de problemas, control y actores) y utilizaba una notación gráfica propia. Esta notación gráfica se diseñó pensando en que sirviera como un mapa para recorrer los modelos y, por otro lado, fuera lo suficientemente distinta de notaciones de diseño como para facilitar que el analista se concentrara únicamente en el problema bajo estudio.

La Figura 5.46 muestra la pantalla principal de la aplicación con parte del modelo estructural del ejemplo de equipos de fútbol, en la que se puede apreciar la notación gráfica con la que se representaban los conceptos y sus relaciones, derivada de la notación propuesta en [Kilov et al., 1993].

Una vez creados los elementos de forma gráfica, la herramienta permite editar su especificación, con el fin de completar todos los detalles que no aparecen en los diagramas. La Figura 5.47 muestra uno de los cuadros de diálogo, en concreto aquél que permite editar las características de un concepto.

El desarrollo de esta primera versión de la herramienta se alargó mucho en el tiempo, debido sobre todo a las dificultades encontradas en tres aspectos de la implementación:

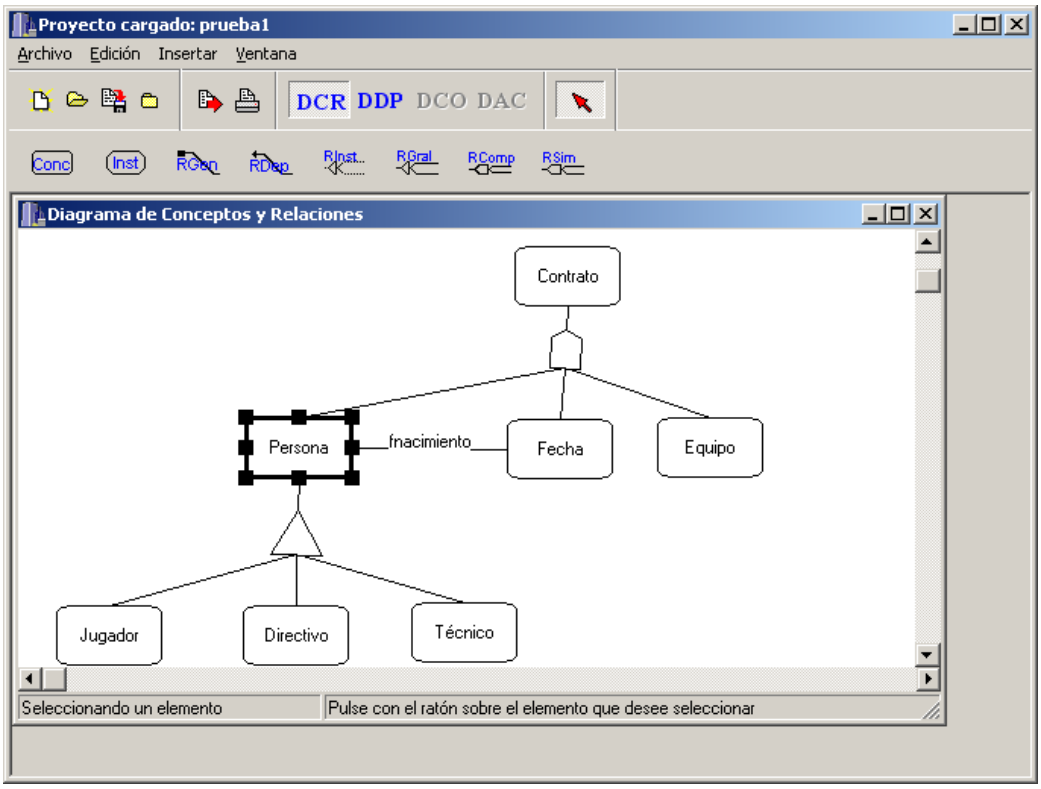


Figura 5.46 Pantalla principal de la primera versión de la herramienta

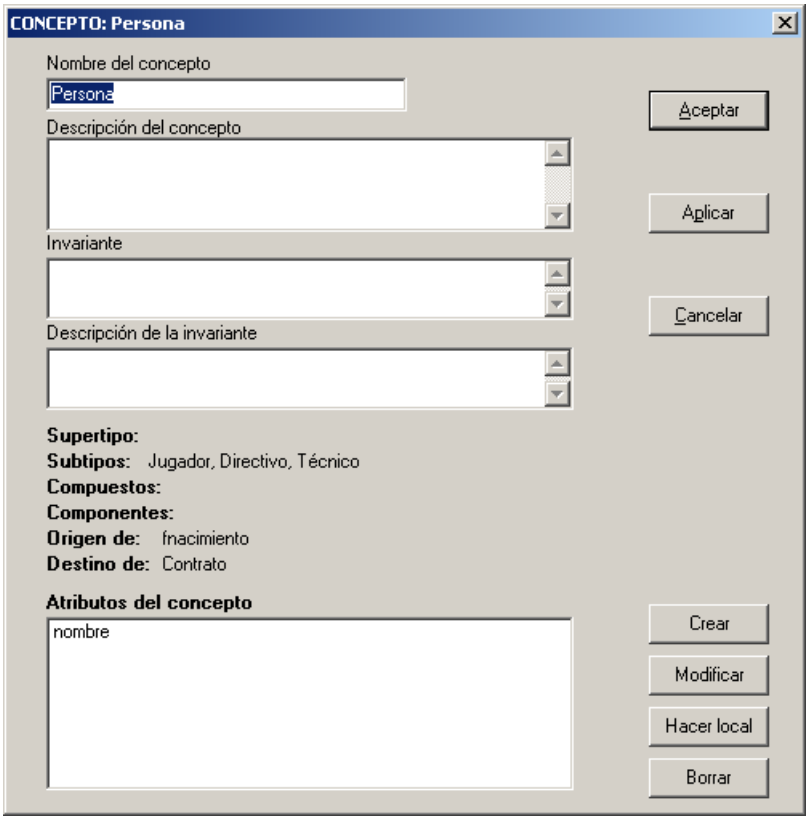


Figura 5.47 Edición de un concepto en la primera versión de la herramienta

- Por un lado la creación de una interfaz gráfica con un editor de diagramas que requirió dedicar muchos esfuerzos de programación de rutinas gráficas de bajo nivel.
- Por otro lado la utilización de una base de datos relacional que no gestiona de forma adecuada estructuras de información jerarquizadas y muy interrelacionadas como son los modelos propuestos.
- El uso del lenguaje C++ exigía mucho esfuerzo por parte de los programadores en aspectos que deberían ser inmediatos como la gestión de colecciones de datos de diferentes tipos.

Este largo tiempo de desarrollo llevó a que la evolución del método propuesto fuera mucho más rápida que el desarrollo de la herramienta, lo que llevó a tomar la decisión de parar ese esfuerzo de programación y desarrollar una **nueva versión**.

Las características de esta nueva versión, que aún está en desarrollo, derivan de la experiencia descrita anteriormente:

- Se decidió *abandonar la interfaz basada en diagramas* a la que en la versión anterior hubo de dedicar muchos esfuerzos no significativos para la completitud de la herramienta. El objetivo era concentrarse en lograr una cobertura completa del método antes de comenzar el desarrollo de un módulo de diagramas.
- Se decidió *no utilizar ninguna base de datos* para almacenar los modelos y utilizar la memoria principal para contenerlos durante la ejecución de la herramienta. Al terminar una sesión de trabajo los modelos se almacenan en ficheros XML, lo que facilita su transporte a otras herramientas.
- Se está utilizando el *lenguaje Java*, por ser mucho más cómodo de utilizar para la gestión de colecciones de datos y por permitir generar programas independientes de plataforma, que pueden ejecutarse prácticamente en cualquier ordenador.

La versión actual contempla en su totalidad el modelo estructural, estando próximo el comienzo de la incorporación del modelo de comportamiento.

La Figura 5.48 muestra la ventana principal de la nueva herramienta. Esta aplicación tiene un menú con comandos para editar el modelo, un árbol donde se muestran todos los elementos creados hasta el momento (tipos, conceptos, asociaciones, etc.) y una zona de texto donde se muestra el resultado de las operaciones solicitadas por el usuario. En esta zona puede observarse un listado parcial de los elementos del modelo estructural del ejemplo de equipos de fútbol.

La edición de los elementos del modelo se realiza bien mediante los menús del programa o bien mediante acciones en el árbol de elementos, junto con la interacción con los cuadros de diálogo que van apareciendo. Como ejemplo, la Figura 5.49 muestra el cuadro de diálogo que permite editar un concepto.

Esta herramienta facilita la creación de modelos correctos de dos formas distintas:

- Por un lado la herramienta reduce la posibilidad de crear modelos incorrectos, realizando comprobaciones cada vez que el usuario intenta crear nuevos elementos.

Como ejemplo puede mencionarse que la herramienta impide crear clasificaciones de concepto y asociación con circularidad.

- Por otro lado la herramienta realiza tareas de verificación cuando el usuario las solicita. Así, por ejemplo, se puede verificar el cumplimiento de las restricciones impuestas por clasificaciones, asociaciones, atributos, etc.

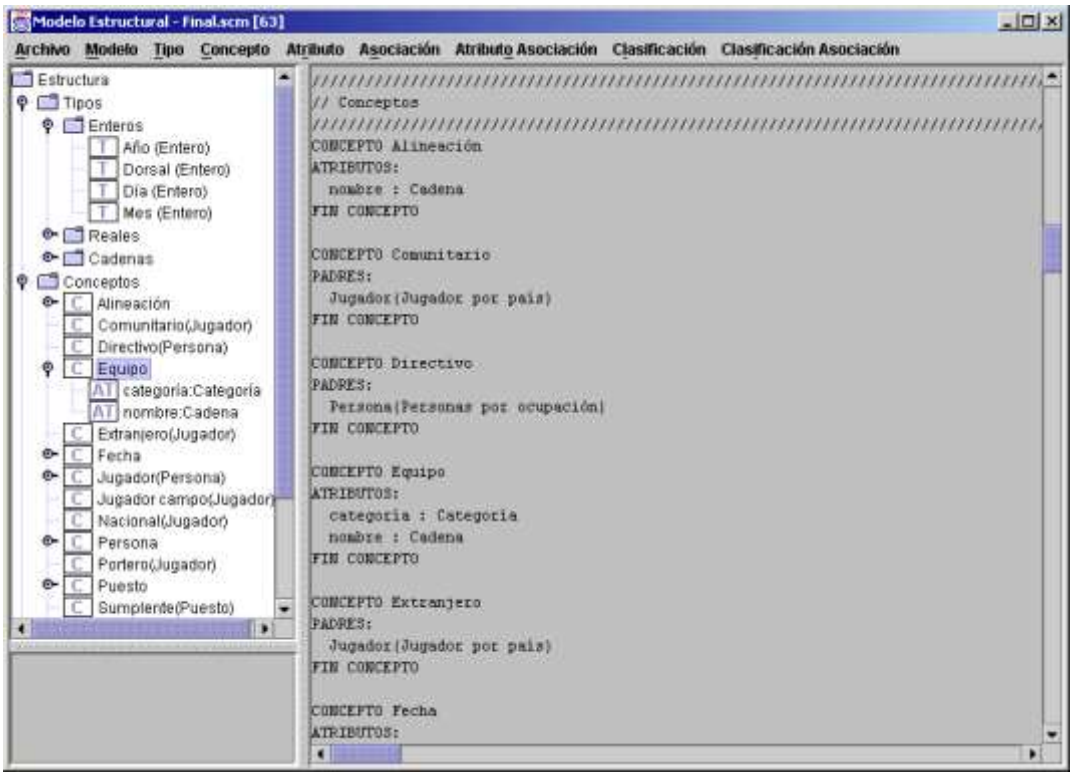


Figura 5.48 Pantalla principal de la nueva versión de la herramienta

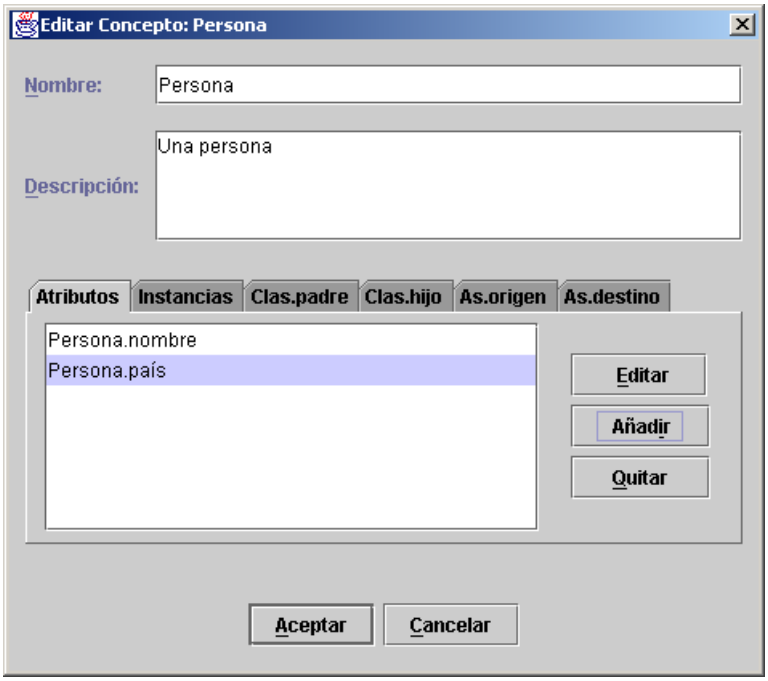


Figura 5.49 Edición de un concepto en la nueva herramienta

5.3 PUBLICACIONES

Como resultado del trabajo continuo que se ha venido desarrollando desde que el autor comenzó a trabajar en los primeros intentos de unificación de metodologías de desarrollo, se han generado una serie de publicaciones que incluyen artículos, ponencias en congresos, trabajos fin de carrera, tesis de master e incluso una tesis doctoral.

5.3.1 Artículos y Ponencias en Congresos

Las publicaciones aquí recogidas (en orden cronológico) muestran la evolución de las ideas que han culminado en esta Tesis, así como su aplicación práctica a proyectos reales.

- Alonso, F.; Fuertes, J.L.; Martínez, L.A., Montes, C. "A knowledge engineering software development methodology applied to a spiral/conical life cycle". Proceeding of the 9th International Conference on Software Engineering and Knowledge Engineering. SEKE'97. Págs. 32-37. 1997.
- Frutos, S.; González, A. L.; Martínez, L. A.; Montes, C. "Adapting Computer Human Interaction in Real Time". Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. IEEE SMC'98. 1998.
- Alonso, F.; de Antonio, A.; González, A. L.; Fuertes, J. L.; Martínez, L. A. "Towards a Unified Methodology for Software Engineering and Knowledge Engineering". Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. IEEE SMC'98. 1998.
- Alonso, F.; Fuertes, J. L.; Martínez, L. A.; Montes, C. "An Expert Systems Development Methodology" Proceedings of the 2nd International Conference on Intelligent Processing Systems. IEEE ICIPS'98. 1998
- Alonso, F; Barreiro, J. M.; Fuertes, J. L.; Martínez, L. A.; Montes, C. "Título: "An Incremental Prototyping Approach to Software Development in Knowledge Engineering". Proceedings of the Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information System Analysis and Synthesis (ISAS'99). 2. 76-82. 31 julio – 4 agosto 1999.
- Alonso, F., de Antonio, A., Martínez, L.A., Montes, C. "Unifying Software Development: Models for the Analysis Phase." Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information System Analysis and Synthesis (ISAS'99), Orlando, Florida.1999.
- Alonso, F.; Fuertes, J. L.; Martínez, L.; Montes, C. "An Incremental Solution for Developing Knowledge-Based Software: Its Application to an Expert System for Isokinetics Interpretation". Expert Systems with Applications. 18. 165-184. Abril 2000.
- Alonso, F., Frutos, S., Fuertes, J.L., Martínez, L.A., Montes, C. "ALBOR. An Internet-Based Advisory KBS with a Multi-Agent Architecture." International

Conference on Advances in Infrastructure for Electronic Business, Science, And Education on the Internet (SSGRR 2001), L'Aquila (Italia), 1-6.2001

5.3.2 Trabajos Fin de Carrera y Tesis de Master

La gran mayoría de los Trabajos aquí recogidos contienen la documentación detallada de la aplicación del método propuesto al desarrollo de proyectos reales.

- del Barrio Molina, Estrella. "Metodología para el desarrollo de asistentes de un traductor de lenguaje gráfico a texto". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Septiembre 1996.
- González Martínez, Ángel Lucas. "Análisis, Diseño e Implementación de un Modelo General de Comunicación Multimedia". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Diciembre 1996.
- Berengeno Lloret, María Jesús. "Modelo externo de un revisor de pantallas en entornos gráficos". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Diciembre 1997.
- Plaza Carrillo, Miguel Ángel. "Proyecto Ruiseñor. Análisis, Diseño e Implementación de un modelo de interfaz musical para invidentes". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Febrero 1998.
- Gil García-Rojo, Roberto. "Proyecto Ruiseñor. Desarrollo de un sistema de composición musical para personas ciegas". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Septiembre 1998.
- de la Cruz Orobio, Daniel. "Modelo de acceso a páginas Web para ciegos". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Julio 1999.
- Ortega Rodríguez-Madridejos, Pablo. "Juegos educativos para ciegos: el ahorcado". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Septiembre 1999.
- de la Flor González, Lorena. "Proyecto Tutor. Componentes adaptables para la interacción con el usuario en sistemas inteligentes de tutoría". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Marzo 2000.
- Frutos Cid, Sonia. "ALBOR: Sistema Basado en Conocimientos con Arquitectura Multi-Agente para Asesoría en Internet". Tesis de Máster en Ingeniería del Conocimiento. Facultad de Informática. Universidad Politécnica de Madrid. Abril 2000.
- García Martínez, David. "Proyecto GESTLAB: aplicación cliente". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Junio 2000.
- Martínez Olano, Juan José. "Proyecto Winlee: Análisis, diseño e implementación de un sistema de reconocimiento de caracteres y gestión documental". Julio 2000.
- Amigo López, Lorena. "Proyecto ALBOR. Modelo de cliente para cuestionarios inteligentes e interactivos en Internet". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Julio 2000.

- López Gutiérrez, Juan Ramón. "Proyecto Winlee: Modelado, diseño e implementación de la interfaz de usuario adaptada para invidentes". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Noviembre 2000.
- Andrés Henche, María Carmen. "Herramienta para la metodología de análisis independiente de problemas: especificación de modelos". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Febrero 2001.
- Picazo Benito, Marcos. "Sistema software de una máquina de estenotipia". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Marzo 2001.
- Sánchez Benítez, Juan Carlos. "Proyecto DELE. Diseño y gestión de la base de datos para el diccionario enciclopédico 'El pequeño Larousse Ilustrado'". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Julio 2001.
- Rodríguez Fernández, Elena. "Proyecto DELE: aplicación de consulta para el diccionario enciclopédico 'El Pequeño Larousse Ilustrado'". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Noviembre 2002.
- García Belmonte, Luis. "Programa de subtitulado para televisión en directo". Trabajo Fin de Carrera. Facultad de Informática. Universidad Politécnica de Madrid. Enero 2003.

5.3.3 Tesis doctorales

Uno de los resultados más destacables de esta Tesis Doctoral es la próxima presentación de otra Tesis Doctoral en esta misma facultad, concretamente la titulada "Modelo de Diseño de una Arquitectura Multi-Agente Basado en un Modelo de Sociedad de Agentes", cuya autora es Sonia Frutos Cid [Frutos, 2003].

Esta Tesis tiene su origen en la aplicación de SETCM al desarrollo del proyecto ALBOR, en sus sucesivas fases, que ya ha sido comentado en el apartado de experimentación.

Esta tesis consiste en un modelo de diseño de sistemas multiagente que toma como punto de partida el análisis del problema realizado con SETCM. A partir de ahí la técnica de la Tesis de Frutos permite estudiar el problema planteado para ver si conviene o no realizar un diseño de sistema con agentes. Una vez tomada la decisión, el método de Frutos describe todos los pasos necesarios para llegar al diseño final.

Desde el punto de vista de SETCM este resultado es extremadamente importante, ya que sirve para demostrar la adecuación de los resultados del método propuesto para el modelado de problemas cuyo diseño computacional puede adoptar, mezclar o compaginar paradigmas completamente distintos, e incluso puede servir como base a procesos de toma de decisión relativos a la arquitectura y diseño interno del sistema.

6 CONCLUSIONES

En esta tesis se ha presentado una **técnica de análisis** llamada SETCM que tiene las siguientes propiedades:

- Es independiente del tipo de problema planteado.
- Es independiente de las técnicas de diseño.
- Tiene una base formal consistente, apoyada en los conceptos elementales de la teoría de conjuntos.
- Ofrece aspectos pragmáticos, en concreto una notación textual y gráfica, que facilitan la comprensión y validación de los modelos generados.
- Es verificable, dada su base formal.
- Tiene soporte de herramientas, centradas en la actualidad en el modelo estructural.

Como se ha descrito en el capítulo de resultados, esta técnica ha sido **aplicada con éxito a sistemas cuyo diseño final presentaba mezcla de paradigmas**: declarativo, dirigido por eventos, orientado a objetos, estructurado, etc.

Gran parte del éxito de la técnica consiste en que **el proceso de formación para su uso es muy ágil**, lo que repercute en un menor coste de entrenamiento y un mayor aprovechamiento de la capacidad de las personas participantes en el proyecto.

En cuanto a su **utilización por profesionales de distinto nivel de experiencia**, se han apreciado las siguientes ventajas:

- En el caso de especialistas ha sido adoptado con naturalidad sin rechazo a la carga formal implícita en la técnica.
- En el caso de noveles les ha ayudado a pensar en el problema sin preocuparse de la solución, sabiendo además que el modelo resultante les permitiría elegir las técnicas de diseño más adecuadas para el problema.

En un proyecto concreto (ALBOR, [Alonso et al., 2001]) se ha podido comprobar que SETCM facilitó las labores de **reutilización y ampliación**. En concreto, al pasar de ALBOR-I a ALBOR-II, se pudieron incorporar las nuevas funcionalidades desde el modelo del problema independientemente de los recursos informáticos necesarios para su materialización.

Se ha comprobado que la **base formal** permite realizar de forma cómoda dos tareas fundamentales para asegurar la calidad de los resultados:

- En primer lugar, se puede realizar una **verificación completa** de los modelos representados con SETCM, comprobando que se cumplen todas las restricciones formales de los modelos. Dentro de este campo una característica muy interesante es que se puede comparar un estado concreto del sistema real con su modelo estructural, para detectar posibles inconsistencias en la especificación del modelo.
- En segundo lugar, y éste es un resultado no esperado al comenzar los trabajos, se ha comprobado que existen grandes posibilidades de **aplicar modelos de calidad** (como el expuesto en [Fuertes, 2003]) a los resultados del análisis. Este hecho permite adelantar en el tiempo la aplicación de la gestión de la calidad al proyecto, con los consiguientes beneficios en cuanto a costes y tiempos de desarrollo.

Siguiendo con aspectos relacionados con la calidad de los modelos, se ha comprobado que la utilización de SETCM **permite conservar con facilidad el lenguaje propio del dominio** de la aplicación, lo que facilita enormemente la validación de los modelos por parte de expertos en el problema.

En cuanto a la integración de SETCM en el proceso de desarrollo de software, se han observado dos resultados muy interesantes:

- La técnica propuesta en esta Tesis es **independiente del modelo de ciclo de vida** utilizado durante el desarrollo. Los proyectos a los que se ha aplicado esta técnica tenían diferentes ciclos de vida (en cascada, prototipado, espiral, etc.) en función de sus características.
- En todos los proyectos se ha producido una **derivación natural del análisis a diseño**:
  - Por un lado los modelos de análisis ofrecían una visión objetiva de todas las partes de los problemas sin sesgos derivados del propio método de análisis.
  - Por otro lado, los modelos tenían expresividad suficiente desde el punto de vista de cualquier paradigma.

Para terminar con las conclusiones de esta tesis se procede a destacar la **principal aportación realizada: la combinación de una base formal con aspectos pragmáticos**:

- Los aspectos pragmáticos permiten evitar rechazo a métodos formales y han llevado a su aplicación práctica sin incurrir en sobrecargas de formación o de esfuerzo.
- La base formal permite evitar la ambigüedad de métodos no formales. A modo de ejemplo, la notación UML ha desembocado en al menos dos proyectos internacionales cuyo objetivo es reducir su ambigüedad: "*Unambiguous UML (2U)*" y "*Precise UML (pUML)*".

A modo de colofón, la Figura 6.1 muestra gráficamente los logros de esta Tesis respecto a los objetivos planteados, utilizando tres medias: la de todos los casos de estudio, la de los casos de 1999 en adelante y la de los casos del año 2000 y siguientes. Puede observarse cómo se ha alcanzado un alto grado de cumplimiento de todos los objetivos, que es creciente según se consideran los resultados más recientes.

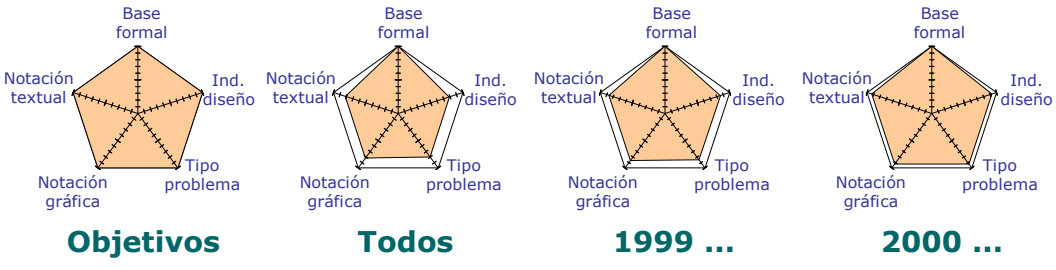


Figura 6.1 Cumplimiento de objetivos

7 LÍNEAS DE TRABAJO FUTURO

Esta Tesis abre tres grandes líneas de trabajo futuro con objetivos de gran relevancia, cada una de las cuales requerirá de grandes esfuerzos de investigación (y en algunos casos de desarrollo).

La primera línea consiste en trabajar sobre el **proceso de análisis**. El objetivo de esta línea sería el definir un proceso óptimo y detallado para llevar a cabo el análisis de problemas de la forma más económica y eficiente posible. Este proceso deberá cumplir con las características esbozadas en el capítulo de solución propuesta:

- Deberá ser iterativo e incremental.
- Deberá ser oportunista.
- Deberá permitir configurar el grado de detalle deseado para los modelos.
- Deberá incorporar criterios que ayuden a elegir técnicas de diseño a partir de las características del problema modelado.

Dentro de la línea del proceso hay que tener en cuenta que el método propuesto ha encajado en distintos ciclos de vida y ha permitido desarrollar sistemas con diferentes paradigmas de programación. El autor considera por tanto que este método podría encajar en cualquier metodología completa de desarrollo que tuviera una fase de análisis explícita. Para comprobar esta hipótesis deberían realizarse estudios encaminados a evaluar la integración de SETCM en metodologías de amplia utilización, como el Proceso Unificado [Jacobson et al., 1999] o, en el contexto español, MÉTRICA V3 [MAP, 2001].

Una segunda línea consiste en, partiendo de unos primeros resultados esperanzadores [Alonso et al., 2001], definir un **modelo de calidad para la fase de análisis** que permita evaluar los modelos aplicando criterios, factores y medidas de una forma estructurada y consistente. Un posible punto de partida sería el modelo de calidad para software orientado a objetos definido en [Fuertes, 2003].

La tercera línea, de marcado carácter teórico pero con fuertes implicaciones para mejorar la pragmática de la aplicación de SETCM, consiste en la **formalización de correspondencia entre resultados de análisis y metamodelos de diseño** y resolución de problemas. Esta formalización debería permitir sistematizar la transición entre el análisis y el diseño e incluso podría producir resultados encaminados a automatizar la transición análisis – diseño, lo que permitiría un gran salto cualitativo en el desarrollo de software, como propugna Blum en su libro "*Beyond programming*" ("Mas allá de la programación") [Blum, 1996].

En un menor grado de relevancia puede considerarse una última línea de trabajo futuro, consistente en la **mejora continuada del método** propuesto. Esta mejora se refiere tanto a aspectos pragmáticos (notación y lenguaje), como a aspectos esenciales. Así, por ejemplo, en el caso de las clasificaciones de concepto y de asociación podría pensarse en la incorporación de más tipos de especializaciones, como una que permitiera restringir el resto de los participantes de una asociación cuando participa en ella un subconcepto o subasociación dados.

8 BIBLIOGRAFÍA

[Abbot, 1990] Abbot, R.J. "Program Design by Informal English Descriptions." *Communications of the ACM*, 26(11), 882-894. 1990.

[Alonso et al., 1995] Alonso, F., de Antonio, A., Fuertes, J.L., Montes, C. "Teaching Communication Skills to Hearing-Impaired Children." *IEEE Multimedia*, 2(4), 55-67. 1995.

[Alonso et al., 1996] Alonso, F., Juristo, N., Maté, J.L., Pazos, J. "Software Engineering and Knowledge Engineering: Towards a Common Life Cycle." *Journal of Systems and Software*, 33. 1996.

[Alonso et al., 1999a] Alonso, F., de Antonio, A., Martínez, L.A., Montes, C. "Unifying Software Development: Models for the Analysis Phase." *Proc. Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information System Analysis and Synthesis (ISAS'99)*, Orlando, Florida. 1999.

[Alonso et al., 2001] Alonso, F., Frutos, S., Fuertes, J.L., Martínez, L.A., Montes, C. "ALBOR. An Internet-Based Advisory KBS with a Multi-Agent Architecture." *Proc. International Conference on Advances in Infrastructure for Electronic Business, Science, And Education on the Internet (SSGRR 2001)*, L'Aquila (Italia), 1-6. 2001.

[Alonso et al., 2002] Alonso, F., Martínez, L.A., Segovia, F.J. *Modelos de Desarrollo de Programas. Segunda Edición.*, Fundación General de la Universidad Politécnica de Madrid, Madrid. 2002.

[Alonso et al., 2003] Alonso, F., Martínez, L.A., Segovia, F.J. *Metodología Básica de Desarrollo Orientado a Objetos*, Fundación General de la Universidad Politécnica de Madrid, Madrid. 2003.

[Alonso, 1996] Alonso, C. *WINDI: Adaptación Tiflotécnica de los elementos estándar de Windows*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1996.

[Amigo, 2000] Amigo, L. *Proyecto ALBOR. Modelo de cliente para cuestionarios inteligentes e interactivos en Internet*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[Andrade, 2002] Andrade, J. *Un Marco Metodológico para el Modelado Conceptual*, Tesis Doctoral, Universidade da Coruña. 2002.

[Andrés, 2001] Andrés, C. *Herramienta para la metodología de análisis independiente de problemas: especificación de modelos.*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2001.

[Angele et al., 1996] Angele, J., Fensel, D., Studer, R. "Domain and task modeling in MIKE." *Domain Knowledge for Interactive System Design*, Sutcliffe, A.et al. (eds.), Chapman & Hall. 1996.

[Angele et al., 1998] Angele, J., Fensel, D., Landes, D., Studer, R. "Developing Knowledge-Based Systems with MIKE." *Journal of Automated Knowledge Engineering*, 5, 389-418. 1998.

[Angele, 1996] Angele, J. "Conceptual Modeling in KARL and G-KARL." *Proc. CASE Workshop.15th International Conference On Conceptual Modelling (ER-96)*, Alemania. 1996.

[Ares et al., 1998] Ares, J., Pazos, J. "Conceptual modelling: an essential pillar for quality software development." *Knowledge-Based Systems*, 11, 87-104. 1998.

[Arquímedes, 1912] Arquímedes. *The Works of Archimedes with the Method of Archimedes*, Dover Publications, New York. 1912.

[Barwise, 1989] Barwise, J. "Mathematical Proofs of Computer System Correctness." *Notices of the American Mathematical Society*, 36, 844-851. 1989.

[Bechtel, 1988] Bechtel, W. *Philosophy of Science: An Overview for Cognitive Science*, Lawrence Erlbaum Associates, Hillsdale. 1988.

[Berengeno, 1997] Berengeno, M.J. *Modelo externo de un revisor de pantallas en entornos gráficos*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1997.

[Blum, 1996] Blum, B.I. *Beyond Programming*, Oxford University Press, New York. 1996.

[Boehm, 1988] Boehm, B.W. "A Spiral Model of Software Development and Enhancement." *Computer*, 21(5). 1988.

[Bonfatti et al., 1994] Bonfatti, F., Monari, P.D. "Towards a General Purpose Approach to Object-Oriented Analysis." *Proc. International Symposium of Object-Oriented Methodologies and Systems, ISOOMS*, Italia, 108-122. 1994.

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I. *El Lenguaje Unificado de Modelado*, Addison-Wesley Iberoamericana, Madrid. 1999.

[Booch, 1986] Booch, G. "Object-Oriented Development." *IEEE Transactions on Software Engineering*, 12(2), 221. 1986.

[Booch, 1994] Booch, G. *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, Redwood City. 1994.

[Bowen, 2003] Bowen, J.P. *The World Wide Web Virtual Library: Formal Methods*. Centre for Applied Formal Methods, SCISM, South Bank University. London. http://www.afm.sbu.ac.uk. 2003.

[Brooks, 1987] Brooks, F.P.J. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer*, 20, 11. 1987.

[Bubenko et al., 1995] Bubenko, J.A., Kirikova, M. "World in requirements specifications by enterprise modelling." *Information Modelling and Knowledge Bases*, Kangassalo, H.et al. (eds.), IOS Press, 92-104. 1995.

[Buchanan et al., 1983] Buchanan, B.G., Barstow, D., Bechtel, R., Bennet, J., Clancey, W., Kulikowski, C., Mitchell, T., Waterman, D.A. "Constructing an Expert System." *Building Expert Systems*, Hayes-Roth, F., Waterman, D.A., Lenat, D.B., (eds.), Addison-Wesley, Reading. 1983.

[Bunge, 1998] Bunge, N. *Philosophy of Science*, Transaction Books, New Jersey. 1998.

[Bursmeister, 1996] Bursmeister, B. "Models and Methodology for Agent-Oriented Analysis and Design." *Proc. KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*, Alemania. 1996.

[Bush et al., 2001] Bush, G., Cranefield, S., Purvis, M. "The Styx Agent Methodology." *The Information Science Discussion Paper Series*, 2001/02. 2001.

[Butler et al., 2000] Butler, K.A., Bahrami, A., Esposito, C., Hebron, R. "Conceptual models for coordinating the design of user work with the design of information systems." *Data and Knowledge Engineering*, 33(2), 191-198. 2000.

[Butler, 1995] Butler, R.W. *An Elementary Tutorial on Formal Specification and Verification Using PVS 2*. Technical Memorandum 118991 (Revised), NASA Langley Research Center, Hampton. 1995.

[Butler, 1996] Butler, R.W. *An Introduction to Requirements Capture using PVS: Specification of a Simple Autopilot*. NASA Technical Memorandum 110255, NASA Langley Research Center, Hampton. 1996.

[Campbell, 1974] Campbell, D.T. "Evolutionary Epistemology." *The Philosophy of Karl Popper*, Schilpp, P.A., (ed.), Open Court, LaSalle, 413-463. 1974.

[Cantor, 1962] Cantor, G. *Gesammelte Abhandlugen*, Georg Olms. 1962.

[Casado, 2002] Casado, L. *Subsistema de consulta para la enciclopedia Larousse L2000*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2002.

[Cashman, 1989] Cashman, M. "Object-Oriented Domain Analysis." *ACM Software Engineering Notes*, 14(6), 67. 1989.

[CETTICO, 1995a] CETTICO. *Conceptos de Windows 3.1*. Informe técnico, Centro de Transferencia Tecnológica en Informática y Comunicaciones (CETTICO). FGUPM, Madrid. 1995a.

[CETTICO, 1995b] CETTICO. *Nivel Estratégico de Windows 3.1*. Informe Técnico, Centro de Transferencia Tecnológica en Informática y Comunicaciones (CETTICO). FGUPM, Madrid. 1995b.

[Chan et al., 1995] Chan, C.W., Toniwachwuthikul, P., Cercone, N. "Knowledge Engineering for a Process Design Domain." *International Journal of Expert Systems*, 8(1), 47-76. 1995.

[Chen, 1976] Chen, P.P. "The Entity Relationship Model: Towards a Unified View of Data." *ACM Transactions on Database Systems*, 1, 9-37. 1976.

[Coad et al., 1990] Coad, P., Yourdon, E. *Object-Oriented Analysis*, Prentice Hall. 1990.

[Coad et al., 1991] Coad, P., Yourdon, E. *Object-Oriented Design*, Prentice Hall, Englewood Cliffs. 1991.

[Collinot et al., 1995] Collinot, A., Carle, P., Zeghal, K. "Cassiopeia: A Method for Designing Computational Organizations." *Proc. First International Workshop on Decentralized Intelligent Multi-Agent Systems*, Krakow, Poland, 124-131. 1995.

[Collinot et al., 1996] Collinot, A., Drogoul, A., Benhamou, P. "Agent Oriented Design of a Soccer Robot Team." *Proc. 2nd International Conference on Multi-Agent Systems, ICMAS*, Japón, 41-47. 1996.

[Connell et al., 1989] Connell, J.L., Shafer, L. *Structured rapid prototyping: an evolutionary approach to software development*, Prentice Hall, Englewood Cliffs. 1989.

[Crow et al., 1995] Crow, J., Owre, S., Rushby, J., Shankar, N., Srivas, M. *A Tutorial Introduction to PVS*. SRI International. 1995.

[Dardenne et al., 1993] Dardenne, A., van Lamsweerde, A. "Goal-directed requirements acquisition." *Science of Computer Programming*, 20, 3-50. 1993.

[Davis, 1993] Davis, A.M. *Software Requirements: Objects, Functions and States*, Prentice Hall, Englewood Cliffs. 1993.

[Dawes, 1991] Dawes, J. *The VDM-SL Reference Guide*, Pitman. 1991.

[de la Cruz, 1999] de la Cruz, D. *Modelo de acceso a páginas Web para ciegos*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1999.

[de la Flor, 2000] de la Flor, L. *Proyecto TUTOR. Componentes adaptables para la interacción con el usuario en sistemas inteligentes de tutoría*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[del Barrio, 1996] del Barrio, E. *Metodología para el desarrollo de asistentes de un traductor de lenguaje gráfico a texto*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1996.

[DeLoach et al., 2001a] DeLoach, S.A., Wood, M.F., Sparkman, C.H. "Multiagent Systems Engineering." *International Journal of Software Engineering and Knowledge Engineering*, 11(3), 231-258. 2001a.

[DeLoach et al., 2001b] DeLoach, S.A., Wood, M.F. "Developing Multiagent Systems with agentTool." *Intelligent Agents VII (ATAL'2000), LNAI 1757*, Jennings, N.R., Lesperance, Y., (eds.), Springer Verlag, Berlin. 2001b.

[DeLoach, 1999] DeLoach, S.A. "Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems." *Proc. First Bi-Conference. Workshop on Agent-Oriented Information Systems (AOIS'99)*, Heidelberg, Alemania. 1999.

[DeMarco, 1979] DeMarco, T. *Structured Analysis and System Specificacion*, CRC Press, New Jersey. 1979.

[Descartes, 1969] Descartes, R. *Oeuvres de Descartes*, Paris. Francia. 1969.

[Descartes, 1977] Descartes, R. *Discurso del Método. Meditaciones Metafísicas. Reglas para la Dirección del Espíritu. Principios de la Filosofía*, Editorial Porrúa, S.A., México. 1977.

[Deutsch, 1997] Deutsch, D. *The Fabric of Reality*, Penguin Books, London. 1997.

[Díez et al., 1997] Díez, J.A., Moulines, C.U. *Fundamentos de Filosofía de la Ciencia*, Ariel, Barcelona. 1997.

[Dijkstra, 1968] Dijkstra, E.W. "Go To Statement Considered Harmful." *Communications of the ACM*, 11, 147-148. 1968.

[D'Souza et al., 1995] D'Souza, D.F., Wills, A.C. *CATALYSIS - Practical Rigor and Refinement*. ICON Computing, Inc. 1995.

[D'Souza et al., 1998] D'Souza, D.F., Wills, A.C. *Objects, Components, and Frameworks With UML: The Catalysis Approach*, Addison-Wesley. 1998.

[Dulac et al., 2002] Dulac, N., Viguier, T., Levenson, N., Storey, M.-A. "On the Use of Visualization in Formal Requirements Specification." *Proc. IEEE Joint International Conference on Requirements Engineering (RE'02)*. 2002.

[Einstein, 1978] Einstein, A. "Notas autobiográficas." *La Teoría de la Relatividad*, Einstein, A.et al. (eds.), Alianza Editorial S.A., Madrid. 1978.

[Elspas et al., 1979] Elspas, B., Green, M., Moriconi, R., Shostak, R. *A JOVIAL Verifier*. Technical Report, Computer Science Laboratory. SRI International. 1979.

[Empédocles et al., 1996] Empédocles, d.A., Anaxágoras, d.C. *Los Filósofos Presocráticos III*, Planeta DeAgostini, S.A., Barcelona. 1996.

[Episkopou et al., 1986] Episkopou, D.M., Wood-Harper, A.T. "Towards a Framework to Choose Appropriate IS Approaches." *Computer Journal*, 29, 222. 1986.

[Eriksson et al., 1995] Eriksson, H., Shahar, Y., Tu, S.W., Puerta, A.R., Musen, M.A. "Task modeling with reusable problem-solving methods." *Artificial Intelligence*, 79, 293-326. 1995.

[Euclides, 1944] Euclides. *Obras Completas*, UNAM, México. 1944.

[EVB, 1986] EVB. *Object-Oriented Design Handbook*. EVB Software Engineering, Inc., Rockville. 1986.

[EVB, 1989] EVB. *Object-Oriented Analysis Handbook*. EVB Software Engineering, Inc. 1989.

[Fenkam et al., 2002] Fenkam, P., Gall, H., Jazayeri, M. "Visual Requirements Validation: Case Study in a Corba-supported environment." *Proc. IEEE Joint International Conference on Requirements Engineering (RE'02)*. 2002.

[Fensel et al., 1996] Fensel, D., Erikson, H., Musen, M.A., Studer, R. "Conceptual and Formal Specifications of Problem-Solving Methods." *International Journal of Expert Systems*, 9(4), 507-532. 1996.

[Finkelstein, 1984] Finkelstein, A. "London Open CRIS Conference." *Computer Bull. 2nd Series*, 5. 1984.

[FIPA, 1996] FIPA. *About FIPA - Rationale*. Foundation for Intelligent Physical Agents (FIPA). http://cselt.stet.it/fipa. 1996.

[Firesmith et al., 1997] Firesmith, D., Henderson-Sellers, B., Graham, I. *OPEN Modeling Language (OML). Reference Manual*, SIGS Books, New York. 1997.

[Firesmith et al., 2001] Firesmith, D., Henderson-Sellers, B. *The OPEN Process Framework: An Introduction*, Addison-Wesley. 2001.

[Firesmith, 1993] Firesmith, D. *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, Wiley, New York. 1993.

[Fowler et al., 1998] Fowler, D.C., Swatman, P.A. "Building Information Systems Development Methods: Synthesising from a Basis in both Theory and Practice." *Proc. Australian Software Engineering Conference*, Australia, 110-117. 1998.

[Fowler et al., 1998] Fowler, D.C., Swatman, P.A. "Building Information Systems Development Methods: Synthesising from a Basis in both Theory and Practice." *Proc. Australian Software Engineering Conference*, Australia, 110-117. 1998.

[Frege et al., 1879] Frege, G., Begriffsschrift. *Eine der Arithmetischen Nachgebildete Formelsprache des Reinen Denkens*, Verlag von Louis Nebert, Halle, Alemania. 1879. En Español: "Un Lenguaje de Fórmulas para el Pensamiento Puro Modelado en el Lenguaje de la Aritmética".

[Frutos et al., 1998] Frutos, S., González, Á.L., Martínez, L.A., Montes, C. "Adapting Computer-Human Interaction in Real Time." *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, San Diego, 1038-1043. 1998.

[Frutos, 2000] Frutos, S. *ALBOR: Sistema Basado en Conocimientos con Arquitectura Multi-Agente para Asesoría en Internet*, Tesis de Máster, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[Frutos, 2001] Frutos, L. *Red ALBOR: mantenimiento distribuido en Internet con múltiples administradores*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2001.

[Frutos, 2003] Frutos, S. *Modelo de Diseño de una Arquitectura Multi-Agente Basado en un Modelo de Sociedad de Agentes*, Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Madrid, Madrid (pendiente de defensa). 2003.

[Fuertes, 1999] Fuertes, J.L. "Off Screen Model (OSM)." *Nuevas Tecnologías y Computación Avanzada en la Ayuda a Discapacitados (1997)*, Fundación Alfredo Brañas, Santiago de Compostela. 1999.

[Fuertes, 2003] Fuertes, J.L. *Modelo de Calidad para el Software Orientado a Objetos*, Tesis doctoral, Facultad de Informática, Universidad Politécnica de Madrid. 2003.

[Gabay, 1991] Gabay, J. *Aprender y Practicar MERISE*, Masson, S.A., Barcelona. 1991.

[Galileo, 1968] Galileo, G. *Le Opere di Galileo Galilei. Edizione Nazionale*, G. Barbèra, Firence. 1968.

[Gane et al., 1979] Gane, C., Sarson, T. *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, New Jersey. 1979.

[García, 2000] García, D. *Proyecto GESTLAB: aplicación cliente*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[García, 2002] García, A. *Proyecto L2000. Desarrollo de la interfaz adaptada para invidentes*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2002.

[García, 2003] García, L. *Programa de subtitulado para televisión en directo*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2003.

[Genesereth et al., 1986] Genesereth, M., Nilsson, N. *Logical Foundation of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., California. 1986.

[Gil, 1998] Gil, R. *Proyecto Ruiseñor. Desarrollo de un sistema de composición musical para personas ciegas*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1998.

[Glaser, 1996] Glaser, N. *Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*, PhD Thesis, L'Universtité Henri Poincaré, Nancy, Francia. 1996.

[Glaser, 1997] Glaser, N. "The CoMoMAS Methodology and Environment for Multi-Agent System Development." *Multi-Agent Systems - Methodologies and Applications, LNAI 1286*, Zhang, C., Lukose, D., (eds.), Springer Verlag, Berlin, 1-16. 1997.

[Glaser, 2002] Glaser, N. "Conceptual Modelling of Multi-Agent Systems: The CoMoMAS Engineering Environment." Weiss, G., (ed.), Kluwer Academic. 2002.

[Gomaa, 1984] Gomaa, H. "A software Design Method for Real-Time Systems." *Communications of the ACM*, 27. 1984.

[Gómez et al., 1997] Gómez, A., Juristo, N., Montes, C., Pazos, J. *Ingeniería del Conocimiento*, Centro de Estudios Ramón Areces, Madrid. 1997.

[Gómez et al., 2000] Gómez, A., Moreno, A., Pazos, J., Sierra-Alonso, A. "Knowledge maps: An essential technique for conceptualisation." *Data and Knowledge Engineering*, 33(2), 169-190. 2000.

[González, 1996] González, Á.L. *Análisis, Diseño e Implementación de un Modelo General de Comunicación Multimedia*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1996.

[González, 2001] González, P. *Red ALBOR: sistema inteligente distribuido para asesoría en Internet*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2001.

[Graham et al., 1997a] Graham, I., Henderson-Sellers, B., Younessi, H. *The OPEN Process Specification*, Addison-Wesley. 1997.

[Graham, 1991] Graham, I. *Object-Oriented Methods*, Addison-Wesley, Harlow. 1991.

[Graham, 1995] Graham, I. *Migrating to Object Technology*, Addison-Wesley, Harlow. 1995.

[Graham, 1996a] Graham, I. "Linking a system and its requirements." *Object Expert*, 1(3), 62-64. 1996.

[Graham, 1997] Graham, I. "Some problems with use cases ... and how to avoid them." *OOIS'96 Proceedings*, Patel, D., Sun, Y., Patel, S., (eds.), Springer Verlag, London. 1997.

[Güell, 2001] Güell, A.M. *Homo Faber, Homo Sapiens. La gestión del Capital Intelectual*, Editorial Planeta, S.A., Barcelona. 2001.

[Guilfoyle et al., 1994] Guilfoyle, C., Warner, E. *Intelligent Agents: the New Revolution in Software*, Ovum Limited, Caindell. 1994.

[Hammer, 1969] Hammer, P.C. "Chart of Elemental Mathematics." *Advances in Mathematical Systems Theory*, Hammer, P.C., (ed.), Pennsylvania State University Press, University Park. 1969.

[Hatley et al., 1987] Hatley, D.J., Pirbhai, I.A. *Strategies for Real-Time System Specification*, Dorset House Publishing, New York. 1987.

[Hayes et al., 1993a] Hayes, I.J., Jones, C.B., Nichols, J.E. *Understanding the differences between VDM and Z*. Technical Report UMCS-93-8-1, UMCS, Manchester. 1993a.

[Heisenberg, 1996] Heisenberg, W. "Diálogos sobre la Física Atómica." *Física Cuántica*, Heisenberg, W., Bohr, N., Schrodinger, E., (eds.), Círculo de Lectores, Barcelona. 1996.

[Henderson-Sellers et al., 1994] Henderson-Sellers, B., Edwards, J.M. *BOOK TWO of Object-Oriented Knowledge: The Working Object*, Prentice Hall International Ltd. 1994.

[Henderson-Sellers et al., 1999] Henderson-Sellers, B., Simons, A., Younessi, H. *The OPEN Toolbox of Techniques*, Addison-Wesley. 1999.

[Henderson-Sellers et al., 2000] Henderson-Sellers, B., Unhelkar, B. *Open Modeling with UML*, Addison-Wesley. 2000.

[Henderson-Sellers, 1996a] Henderson-Sellers, B. "Convergence Is in the Air." *Report on Object Analysis & Design*, 2(6), 47-49, 54. 1996a.

[Henderson-Sellers, 1996b] Henderson-Sellers, B. *OPEN Modeling Language (Light notation) Version 1.0*. School of Computer Science & Software Engineering. Swinburne University. 1996b.

[Heráclito et al., 1996] Heráclito, Parménides, Zenón, d.E., Meliso, d.S. *Los Filósofos Presocráticos II*, Planeta DeAgostini, S.A., Barcelona. 1996.

[Hernández, 2002] Hernández, C. *Sistema de Traducción y Modelo de Base de Datos para la Enciclopedia Larousse 2000*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2002.

[Hesse, 1980] Hesse, M.B. *Revolutions and Reconstructions in the Philosofy of Science*, Harvester Press, Brighton. 1980.

[Høydalsvik et al., 1993] Høydalsvik, G.M., Sindre, G. "On the Purpose of Object Oriented Analysis." *Proc. Conference on Object Oriented Programming, Systems, Languages and Applications, OOPSLA*, EEUU, 240-255. 1993.

[Hoyningen-Huene, 1993] Hoyningen-Huene, P. *Reconstructing Scientific Revolutions: Thomas S. Kuhn's Philosophy of Science*, University of Chicago Press, Chicago. 1993.

[IFAD, 1999] IFAD. *The IFAD VDM-SL Language*. IFAD. 1999.

[Iglesias et al., 1999] Iglesias, C.A., Garijo, M., González, J.C., Velasco, J.R. "Analysis and Design of Multiagent Systems using MAS-CommonKADS." *Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL97), LNAI 1365*, Singh, M.P., Rao, A., Wooldridge, M., (eds.), Springer Verlag, Berlin, 313-326. 1999.

[Iglesias, 1998] Iglesias, C.A. *Definición de una Metodología para el Desarrollo de Sistemas Multiagente*, Tesis Doctoral, Universidad Politécnica de Madrid. 1998.

[ISO, 1996] ISO. *ISO/IEC 13817-1. Information technology -- Programming languages, their environments and system software interfaces -- Vienna Development Method -- Specification Language -- Part 1: Base language.*, ISO. 1996.

[ISO, 2002] ISO. *ISO/IEC 13568:2002. Information technology -- Z formal specification notation -- Syntax, type system and semantics*, ISO. 2002.

[Jackson, 1995a] Jackson, M.A. *Software RER & SPEC*, Addison-Wesley, Reading. 1995a.

[Jackson, 1995b] Jackson, M. *Software Requirements and Specifications. A Lexicon of Practice, Principles and Prejudices*, ACM Press, New York. 1995b.

[Jackson, 2001a] Jackson, M. *Problem Frames: Analysing and Structuring Software Development Problems*, Addison-Wesley, Harlow, UK. 2001a.

[Jacobson et al., 1992] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. *Object-Oriented Software Engineering: a Use-Case Driven Approach*, Addison-Wesley, Reading. 1992.

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J. *The Unified Software Development Process*, Addison-Wesley, Reading. 1999.

[Kendall et al., 1996] Kendall, E.A., Malkoun, M.T. "A Methodology for Developing Agent Based Systems." *Distributed Articial Intelligence - Architecture and Modelling, LNAI 1087*, Zhang, C., Lukose, D., (eds.), Springer Verlag, Berlin, 85-99. 1996.

[Kendall et al., 1997a] Kendall, E.A., Malkoun, M.T. "Design Patterns for the Development of Multi-Agent Systems." *Multi-Agent Systems - Methodologies and Applications, LNAI 1286*, Zhang, C., Lukose, D., (eds.), Springer Verlag, Berlin. 1997.

[Khoshafian et al., 1990] Khoshafian, S., Abnous, R. *Object Orientation.- Concepts, Language, Databases, User Interfaces*, John Willey and Sons Ltd. 1990.

[Kilov et al., 1993] Kilov, H., Ross, J. *Information Modeling. An object-oriented approach*, Prentice Hall, Englewood Cliffs. 1993.

[Kinny et al., 1996] Kinny, D., Georgeff, M., Rao, A. "A Methodology and Modelling Technique for Systems of BDI Agents." *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW*, Alemania, 56-71. 1996.

[Kinny et al., 1997] Kinny, D., Georgeff, M. "Modelling and Design of Multiagent Systems." *Intelligent Agents III (ATAL'96), LNAI 1193*, Müller, J.P., Wooldridge, M., Jennings, N.R., (eds.), Springer Verlag, Berlin, 1-20. 1997.

[Kirikova et al., 1994a] Kirikova, M., Bubenko, J.A. "Software Requirements Acquisition Through Enterprise Modelling." *Proc. 6th International Conference on Software Engineering and Knowledge Engineering, SEKE*, Letonia, 20-27. 1994.

[Kirikova, 2000] Kirikova, M. "Explanatory capability of enterprise models." *Data and Knowledge Engineering*, 33(2), 119-136. 2000.

[Knight et al., 1997] Knight, J.C., DeJong, C.L., Gibble, M.S., Nakano, L.G. "Why Are Formal Metgods Not Used More Widely?" *Proc. Fourth NASA Formal Methods Workshop*, Hampton, Virginia. 1997.

[Kruchten, 2000] Kruchten. *Rational Unified Process - An Introduction. 2nd Edition.*, Addison-Wesley. 2000.

[Kuhn, 1970] Kuhn, T.S. *The Structure of Scientific Revolutions*, University of Chicago Press, Urbana. 1970.

[Lakatos, 1970] Lakatos, I. "Falsification and the Methodology of Scientific Research Programmes." *Criticism and the Growth of Knowledge*, Lakatos, I., Musgrave, A., (eds.), Cambridge University Press, Cambridge, 91-196. 1970.

[Lakatos, 1978a] Lakatos, I. "History of Science and Its Rational Reconstructions." *The Methodology of Scientific Programmes. Philosophic Papers of Imre Lakatos*, Warrall, J., Currie, G., (eds.), Cambridge University Press, Cambridge. 1978a.

[Lakatos, 1978b] Lakatos, I. *Philosofical Papers*, Cambridge University Press, Cambridge. 1978b.

[Lalonde et al., 1990] Lalonde, W.R., Pugh, R.J. *Inside Smalltalk*, Prentice Hall, USA. 1990.

[Landes et al., 1995] Landes, D., Studer, R. "The treatment of non-functional requirements in MIKE." *Proc. 5th European Software Engineering Conference (ESEC'95). LNCS 989.* 1995.

[Larousse, 1997] Larousse. *El Pequeño Larousse Ilustrado*, Larousse Planeta, Barcelona. 1997.

[Laudan et al., 1986] Laudan, L., Donovan, A., Laudan, R., Barker, P., Brown, H., Leflin, J., Thagard, P., Wykstra, S. "Scientific Change: Philosophical Models and Historical Research." *Synthese*, 69, 141-223. 1986.

[Lazimy, 1989] Lazimy, R. "E2R Model and Object-Oriented Representation for Data Management, Process Modeling, and Decision Support." *Proc. 8th Internation Conference on Entity-Relationship Approach*, Canada, 129-151. 1989.

[Lind, 1999] Lind, J. *MASSIVE: Software Engineering for Mulitagent Systems*, PhD Thesis, Der Technischen Fakultät der Universität des Saarlandes, Saarbrücken. 1999.

[Lind, 2001] Lind, J. "Iterative Software Engineering for Mulitagent Systems: The MASSIVE method." *LNCS - 1994*, Springer Verlag. 2001.

[Lindley, 1993] Lindley, D. *The End of Physics: The Myth of a Unified Theory*, Basic Books, New York. 1993.

[López, 2000] López, J.R. *Proyecto WINLEE. Modelado, diseño e implementación de la interfaz de usuario adaptada para invidentes*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[Lorensen, 1986] Lorensen, W. *Object-Oriented Design. CRD Software Engineering Guidelines.*, General Electric, Co. 1986.

[Loucopoulos et al., 1995] Loucopoulos, P., Karaostas, V. *Systems Requirements Engineering*, McGraw-Hill, Berkshire. 1995.

[Maes, 1995] Maes, P. "Artificial Life Meets Entertainment: Life Like Autonomous Agents." *Communications of the ACM*, 38(11), 108-114. 1995.

[Malitz, 1979] Malitz, J. *Introduction to Matematical Logic. Set theory, Computable functions, Model Theory*, Springer Verlag, New York. 1979.

[MAP, 2001] MAP. *Metodología de Planificación, Desarrollo y Mantenimiento de Sistemas de Información MÉTRICA v.3*, Ministerio de Administraciones Públicas de España, Madrid. 2001.

[Martin et al., 1992] Martin, J., Odell, J.J. *Object Oriented Analysis and Design*, Prentice Hall, New Jersey. 1992.

[Martínez, 2000] Martínez, J.J. *Proyecto WINLEE. Análisis, diseño e implementación de un sistema de reconocimiento de caracteres y gestión documental*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2000.

[Melliar-Smith et al., 1985] Melliar-Smith, P.M., Rushby, J. "The Enhanced HDM system for specification and verification." *ACM Software Engineering Notes*, 10(4), 41-43. 1985.

[Meyer, 1988] Meyer, B. *Object Oriented Software Construction*, Prentice Hall, New York. 1988.

[Mineau et al., 2000] Mineau, G.W., Missaoui, R., Godinx, R. "Conceptual modeling for data and knowledge management." *Data and Knowledge Engineering*, 33(2), 137-168. 2000.

[Morales et al., 1993] Morales, A., Segovia, F.J. *Programación Orientada a Objetos. Aplicaciones con Smalltalk.*, Paraninfo S.A., Madrid. 1993.

[Moulin et al., 1994] Moulin, B., Cloutier, L. "Collaborative Work Based on Multiagent Architectures: a Methodological Perspective." *Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence*, Aminzadeh, F., Jamshidi, M., (eds.), Prentice Hall, New Jersey. 1994.

[Moulin et al., 1996] Moulin, B., Brassard, M. "A Scenario-Based Design Method and an Environment for the Development of Multi-Agent Systems." *Distributed Articial Intelligence - Architecture and Modelling, LNAI 1087*, Zhang, C., Lukose, D., (eds.), Springer Verlag, Berlin. 1996.

[Moya, 1999] Moya, J.M. *Proyecto GESTLAB: servidor de bases de datos*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1999.

[Myers, 1985] Myers, W. "MCC: Planning the Revolution in Software." *IEEE Software*, 2, 72. 1985.

[Mylopoulos et al., 1990] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M. "TELOS: Representing Knowledge about Information Systems." *ACM Transactions on Information Systems*, 8(4), 325-362. 1990.

[NASA, 1997] NASA. *Formal Methods Specification and Verification Guidebook for Software and Computer Systems. Volume II: A Practitioners Companion*. 1997.

[NASA, 1998] NASA. *Formal Methods Specification And Verification Guidebook For Software And Computer Systems. Volume I: Planning And Technology Insertion*. 1998.

[Naur et al., 1969] Naur, P., Randell, B. *Software Engineering: Report on a Conference Sponsored by the NATO Science Comité*. División de Asuntos Científicos de la Organización del Tratado del Atlántico Norte (OTAN), Garmish. 1969.

[Neubert, 1993] Neubert, S. "Model Construction in MIKE (Model-based and incremental knowledge engineering)." *Knowledge acquisition for knowledge-based systems, Proc of the 7th European Workshop (EKAIW'93). LNAI 723*, Aussenac, N.et al. (eds.), Springer Verlag, Toulouse. 1993.

[Nguyen et al., 2000a Nguyen, L., Swatman, P.A. "Essential and Incidental Complexity in Requirements Models." *Proc. 4th International Conference on Requirements Engineering*, 130-139. 2000.

[Nissen et al., 1996] Nissen, H.E., Jeusfeld, M., Jarke, M., Zemanek, G., Huber, H. "Managing Multiple Requirements Perspectives with Metamodels." *IEEE Software*, 13(2), 37-48. 1996.

[Nwana et al., 1996] Nwana, H.S., Wooldridge, M. "Software Agent Technologies." *British Telecomunications Technology Journal*, 14(4), 68-78. 1996.

[OGC, 2000] OGC. *Business Systems Development Series with SSADM*, The Stationery Office. 2000.

[OGC, 2001] OGC. *Business and Operational Guidance - Business Systems Development*, Office of Government Commerce. 2001.

[OMG, 2003] OMG. *Unified Modeling Language Specification. Version 1.5*. formal/03-03-01, Object Management Group. 2003.

[Omicini, 2001] Omicini, A. "SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems." *Agent-Oriented Software Engineering (LNAI 1957)*, Ciancarini, P., Wooldridge, M., (eds.), Springer Verlag, 185-194. 2001.

[ONCE, 1996] ONCE. *LEE. Manual de Usuario. Versión 2.0.*, Organización Nacional de Ciegos Españoles (ONCE), Madrid. 1996.

[Ortega, 1999] Ortega, P. *Juegos Educativos para ciegos: El Ahorcado*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1999.

[Owre et al., 2001a] Owre, S., Shankar, N., Rushby, J., Stringer-Calvert, D.W.J. *PVS Language Reference. Version 2.4*. SRI International, Menlo Park. 2001a.

[Owre et al., 2001b] Owre, S., Shankar, N., Rushby, J., Stringer-Calvert, D.W.J. *PVS System Guide. Version 2.4*. SRI International, Menlo Park. 2001b.

[Paradela, 2001] Paradela, L.F. *Una Metodología para la Gestión de Conocimientos*, Tesis Doctoral, Universidad Politécnica de Madrid, Madrid. 2001.

[Parnas, 1972] Parnas, D.L. "On the Criteria To Be Used in Decomposing Systems into Modules." *Communications of the ACM*, 15, 1053-1058. 1972.

[Pazos, 2003a] Pazos, J. "Gestión del Conocimiento Computable: Desde la Filosofía a la Empresa." Ávila. 2003a.

[Picazo, 2001] Picazo, M. *Sistema software de una máquina de estenotipia*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2001.

[Platón, 1944] Platón. *Obras Completas*, UNAM, México. 1944.

[Plaza, 1998] Plaza, M.Á. *Proyecto Ruiseñor. Análisis, Diseño e Implementación de un Modelo de Interfaz Musical para Invidentes*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1998.

[Plexousakis, 1993] Plexousakis, D. "Semantial and ontological considerations in TELOS: a language for knowledge representation." *Computational Intelligence*, 9(1). 1993.

[Popper, 1959] Popper, K.R. *The Logic of Scientific Discovery*, Harper & Row, New York. 1959.

[Popper, 1965] Popper, K.R. *Conjectures and Refutations: The Growth of Scientific Knowledge*, Basic Books, New York. 1965.

[Popper, 1974] Popper, K.R. "Campbell on the Evolutionary Theory of Knowledge." *The Philosophy of Karl Popper*, Schilpp, P.A., (ed.), Open Court, LaSalle, 1059-1065. 1974.

[Popper, 1985] Popper, K.R. *Post Scritum a la Lógica de la Investigación Científica. Vol 1. Realismo y el Objetivo de la Ciencia.*, Tecnos, Madrid. 1985.

[Pressman, 1997] Pressman, R.S. *Software Engineering. A practitioner's approach. 4th Edition.*, McGraw-Hill. 1997.

[Puerta et al., 1992] Puerta, A.R., Egar, J.W., Tu, S.W., Musen, M.A. "A multiple-method knowledge acquisition shell for the automatic generation of knowledge acquisition tools." *Knowledge Acquisition*, 4, 171-196. 1992.

[Quine, 1961] Quine, W.V.O. "Two Dogmas of Empirism." *From a Logical Point of View*, Quine, W.V.O., (ed.), Harper & Row, New York, 20-46. 1961.

[Quine, 1969] Quine, W.V.O. *Ontological Relativity and Other Essays*, Columbia University Press, New York. 1969.

[Quine, 1975] Quine, W.V.O. "The Nature of Natural Language." *Mind and Language*, Guttenplan, S., (ed.), Clarendon Press, Oxford. 1975.

[RAE, 2001] RAE. *Diccionario de la Lengua Española. Vigésima segunda edición.*, Editorial Espasa Calpe, S.A., Madrid. 2001.

[Rational, 2002] Rational. *Rational Unified Process. Version 2002.05.00*. Rational Software Corporation. 2002.

[Reenskaug, 1992] Reenskaug, T. "OORASS: seamless support for the creation and maintenance of object oriented systems." *Journal of Object Oriented Programming*, 5(6), 27-41. 1992.

[Robinson et al., 1979] Robinson, L., Levitt, K.N., Silverberg, B.A. *The HDM Handbook*. Computer Science Laboratory. SRI International. 1979.

[Rodríguez, 1996] Rodríguez, L.A. *WINDI: Windows para invidentes. Adaptación Tiflotécnica de los elementos no estándar de Windows*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 1996.

[Rodríguez, 2002] Rodríguez, E. *Proyecto DELE: aplicación de consulta para el diccionario enciclopédico 'El Pequeño Larousse Ilustrado'*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2002.

[Royce, 1970] Royce, W.W. "Managing the Development of Large Software Systems: Concepts and Techniques." *Proc. WESCON*. 1970.

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. *Modelado y Diseño Orientado a Objetos. Metodología OMT*, Prentice Hall, Madrid. 1991.

[Rumbaugh, 1997a] Rumbaugh, J. "Modeling through the Years." *Journal of Object Oriented Programming*, Julio - agosto, 16-19. 1997a.

[Rumbaugh, 1997b] Rumbaugh, J. "Models trough the development process." *Journal of Object Oriented Programming*, Mayo, 5-8, 14. 1997b.

[Rushby et al., 1991] Rushby, J., von Henke, F., Owre, S. *An introduction to formal specification and verification using EHDM*. Technical Report SRI-CSL-91-2, Computer Science Laboratory. SRI International. 1991.

[Rushby et al., 1996] Rushby, J., Stringer-Calvert, D.W.J. *A Less Elementary Tutorial for the PVS Specification and Verification System*. Technical Report CSL-95-10, Computer Science Laboratory. SRI International, Menlo Park CA. 1996.

[Russell, 1953] Russell, B. *Problemas de la Filosofía*, Labor, S.A., Barcelona. 1953.

[Sánchez, 2001] Sánchez, J.C. *Proyecto DELE. Diseño y gestión de la base de datos para el diccionario enciclopédico 'El Pequeño Larousse Ilustrado'*, Trabajo Fin de Carrera, Facultad de Informática, Universidad Politécnica de Madrid, Madrid. 2001.

[Schreiber et al., 1994] Schreiber, G., Wielinga, B.J., De Hoog, R., Akkermans, H., van de Velde, W. "CommonKADS: a Comprehensive Methodology for KBS Development." *IEEE Expert*, 28-37. 1994.

[Schreiber et al., 1999] Schreiber, G., Akkermans, H., Anjewierden, A., De Hoog, R., Shadbolt, N., van de Velde, W., Wielinga, B.J. *Knowledge Engineering and Management: the CommonKADS Methodology*, MIT Press, Massachusetts. 1999.

[Seemann, 1958] Seemann, O. *Mitología Clásica Ilustrada*, Vergara Editorial, S.A., Barcelona. 1958.

[Shapiro, 1997] Shapiro, S. "Splitting the difference: the historical necessity of synthesis in software engineering." *IEEE Annals of the History of Computing*, 19(1), 20-54. 1997.

[Shostak et al., 1982] Shostak, R., Schwartz, R., Melliar-Smith, P.M. "STP: A mechanized logic for specification and verification." *6th International Conference on Automated Deduction (CADE). LNCS 138.*, Loveland, D., (ed.), Springer Verlag, New York. 1982.

[Snowdon, 1996] Snowdon, B. "Active models and process support." *Proc. 5th European Workshop on Software Process Technology, EWSPT'96*, 93-98. 1996.

[Spinellis, 2003] Spinellis, D. "On the Declarative Specification of Models." *IEEE Software*, 96, 94, 95. 2003.

[Spitzen et al., 1978] Spitzen, J.M., Levitt, K.N., Robinson, L. "An example of hierarchical design and proof." *Communications of the ACM*, 21(12), 1064-1075. 1978.

[Spivey, 1992] Spivey, J.M. *The Z Notation: A Reference Manual*, Prentice Hall International Ltd., UK. 1992.

[Stevens et al., 1974] Stevens, W., Myers, G., Constantine, L. "Structured Design." *IBM Systems Journal*, 13, 115-139. 1974.

[Studer et al., 1998] Studer, R., Benjamins, V.R., Fensel, D. "Knowledge Engineering: Principles and methods." *Data and Knowledge Engineering*, 25, 161-198. 1998.

[Suppe, 1974] Suppe, F. "The Search for Philosophic Understanding of Scientific Theories." *The Structure of Scientific Theories*, Suppe, F., (ed.), University of Chicago Press, Urbana, 3-241. 1974.

[Swatman et al., 1998] Swatman, P.A., Fowler, D.C. *Requirements Engineering for High Quality Information Systems. The FOOM Approach*, Addison-Wesley. 1998.

[Tu et al., 1995] Tu, S.W., Eriksson, H., Gennari, J.H., Shahar, Y., Musen, M.A. "Ontology-based configuration of problem-solving methods and generation of knowledge acquisition tools: The application of PROTÉGÉ-II to protocol-based decision support." *Artificial Intelligence in Medicine*, 7(5). 1995.

[van Dalen et al., 1978] van Dalen, D., Doets, H.C., de Swart, H. *Sets: Naive, Axiomatic, and Applied*, Pergamon Press. 1978.

[van Lamsweerde et al., 1991] van Lamsweerde, A., Dardenne, A., Delcourt, B., Dubisy, F. "The KAOS Project: Knowledge Acquisition in autOmated Specification of Software." *Proc. American Association for Artificial Intelligence Spring Symposium Series*, EEUU, 59-62. 1991.

[Vitt et al., 1996] Vitt, J., Hooman, J. "Assertional Specification and Verification using PVS of the Steam Boiler Control System." *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control, LNCS 1165*, Springer Verlag, 453-472. 1996.

[Wielinga et al., 1992] Wielinga, B.J., Schreiber, A.T., Breuker, J.A. "KADS: a Modelling Approach to Knowledge Engineering." *Knowledge Acquisition*, 4(1), 5-54. 1992.

[Wirth, 1971] Wirth, N. "Program Development by Stepwise Refinement." *Communications of the ACM*, 14, 221-227. 1971.

[Wood et al., 2000] Wood, M.F., DeLoach, S.A. "An Overview of the Multiagent Systems Engineering Methodology." *Proc. First International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, Limerick, Irlanda. 2000.

[Wood, 2000] Wood, M.F. *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*, MS Thesis, School of Engineering. Air Force Institute of Technology, Ohio. 2000.

[Wooldridge et al., 1999] Wooldridge, M., Jennings, N.R., Kinny, D. "A Methodology for Agent-Oriented Analysis and Design." *Proc. Third International Conference on Autonomous Agents (Agents '99)*, 69-76. 1999.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N.R., Kinny, D. "The Gaia Methodology for AgentOriented Analysis and Design." *International Journal of Autonomous Agents and MultiAgent Systems*, 3(3), 285-312. 2000.

[Yourdon, 1975] Yourdon, E. *Techniques of Program Structure and Design*, Prentice Hall, Englewood Cliffs. 1975.

[Yourdon, 1989] Yourdon, E. *Modern Structured Analysis*, Prentice Hall, New Jersey. 1989.

[Yu et al., 1994] Yu, E., Mylopoulos, J. "Understanding "why" in software process modelling, analysis and design." *Proc. 16th International Conference on Software Engineering*. 1994.

[Zubiri, 1999] Zubiri, X. *Naturaleza, Historia, Dios.*, Alianza Editorial S.A., Madrid. 1999.