UNIVERSIDAD DE ALMERÍA
DEPARTAMENTO DE LENGUAJES Y COMPUTACIÓN

InSCo.

Un modelo de proceso para la integración entre la ingeniería del software y la ingeniería del conocimiento

Memoria presentada por **Isabel María del Águila Cano**
Para optar al grado de doctor por la Universidad de Almería

Directores:
Dr. Samuel Túnez Rodríguez
Dr. José Tomás Palma Méndez

Almería, Junio 2010

Agradecimientos

El trabajo que se presenta en esta Memoria de Tesis Doctoral se ha logrado gracias al esfuerzo y apoyo que me han brindado a lo largo de los años numerosas personas. Me gustaría por tanto, expresar de algún modo mi gratitud no sólo a aquellos con los que he trabajado directamente, sino también a los que me dieron aliento y consuelo en los momentos más difíciles.

En primer lugar quiero dar mi más sincero agradecimiento a mis directores de Tesis, D. Samuel Túnez Rodríguez y D. José Tomás Palma Méndez, sin su apoyo, dirección y paciencia no hubiese llevado a término este trabajo.

También quiero agradecer al profesor D. Roque Marín Morales sus indicaciones y consejos para la definición de las líneas de trabajo de esta Tesis.

A todo el Grupo de investigación "Ingeniería de Datos, del Conocimiento y del Software (DKSE)". Especialmente a Jose Joaquín y Javi con los que tan estrechamente he colaborado.

A todos mis compañeros del departamento de Lenguajes y Computación de la Universidad de Almería por su ayuda en la realización y presentación de esta tesis.

En particular, quiero dar las gracias a José del Sagrado por sus recomendaciones durante los muchos ratos compartidos.

Agradecer al Ministerio de Ciencia y Tecnología la concesión de del proyecto (TIN2004-05694), y al Proyecto de excelencia de la Junta de Andalucía (P06-TIC-02411.02), así como a los convenios de colaboración con la Dirección General de la Producción Agraria de la Consejería de Agricultura y Pesca de la Junta de Andalucía, que han otorgado la financiación necesaria para la consecución de este trabajo.

"Es mejor cojear por el camino que avanzar a grandes pasos fuera de él. Pues quien cojea en el camino, aunque avance poco, se acerca a la meta, mientras que quien va fuera de él, cuanto más corre, más se aleja." San Agustín.

A Isa, Ana y José Miguel
y por estar ahí siempre, a Jose.

Índice general

Índice de figuras

Índice de tablas

Índice de ejemplos

Lista de Acrónimos

A	Actividad fundamental Análisis
ANA	Modelo de Análisis
BN	Red Bayesiana
C	Actividad fundamental Conocimiento
CASE	Computer Aided Software Engineering
CIM	Computational Independent Model
CK	CommonKADS
CML	Conceptual Modeling Language
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
CommonKADS	Commprensive Methodology for KBS Development
CON	Modelo de Conocimiento
D	Actividad fundamental Diseño
DFD	Diagrama de Flujo de Datos
DIS	Modelo de Diseño
DOC_ARQ	Documento de definición de la arquitectura
DOC_DS	Documento de definición del sistema
DOC_ERS	Documento de especificación de requisitos
DTE	Diagrama de Transición de Estados
GPS	General Problem Solver
HCI	Interfaz Hombre-máquina
I	Actividad fundamental Implementación
IA	Inteligencia Artificial

IEEE	Institute de Electric and Electronic Engineering
IEEE/EIA	Institute of Electrical and Electronics Engineers Electronic Industries Association
IMP	Modelo de Implementación
InCo	Ingeniería del Conocimiento
InRe	Ingeniería de los Requisitos
InSo	Ingeniería del Software
IOC	Initial Operational Capacity
ISO/IEC	International Organization of Standarization /International Electrotecnical Commission
JAD	Joint Application Development
JDS	Jackson Development System
JSP	Jackson Structured Programming
KADS	Knowledge Acquisition and Design Structuring
KARL	Knowledge Acquisition Representation Language
KLIC	Knowledge Based System Life Cycle
LCA	LifeCycle Architecture
LCO	LifeCycle Objectives
MAS	Multi-Agent System
MAS-CommonKADS	Multi-Agente CK
MaSE	Multiagent System Engineering
MDA	Model Driven Arquitecture
MIKE	Model-based and Incremental Knowledge Engineering
ML2	Modeling Language
MSC	Message Sequence Charts
MVC	Modelo-Vista-Controlador
NEG	Modelo de Negocio
O	Actividad fundamental Organización
OMG	Object Management Group
OMT	Object Modeling Technique

OOSE	Object Oriented Software Engineering
OTAN	Organización del tratado del Atlántico Norte
PAC	Presentación abstracción y Control
PDA	Personal Digital Assistant (Asistente Digital Personal)
PIM	Platform Independent Model
POO	Programación Orientada a Objetos
PRU	Modelo de Prueba
PSM	Platform Specif Model
R	Actividad fundamental Requisitos
RUP	Rational Unified Process
SAD	Structured Analysis and Design
SADT	Structured Analysis and Design Technique
SAEPI	Sistema de asesoramiento experto para la producción integrada en Agricultura
SAVIA	Sistema de Ayuda Experta a la Vigilancia Integrada en Agricultura
SBC	Sistema Basado en Conocimiento
SBSE	Search Based Software Engineering
SDL	Specification and Description Language
SEI	Software Engineering Institute
SER	Modelo de Servicios
SREM	Software Requirement Engineering Methodology
STRIPS	Stanford Research Institute Problem Solver System
SWEBOK	Software Engineering Body of Knowledge
UML	Unified Modeling Language
XP	Extreme Programming

Introducción general y resumen

El objetivo general del trabajo de investigación presentado en esta memoria es la propuesta de un modelo de proceso para el desarrollo de un sistema híbrido, que llamaremos modelo de proceso InSCo; siendo un sistema híbrido aquel que presenta un comportamiento integrado donde colaboran componentes software basadas en conocimiento y no basadas en conocimiento. La propuesta del modelo de proceso InSCo define las actividades fundamentales y artefactos generados en la ejecución del proceso, prestando atención a los modelos construidos y técnicas utilizables en cada actividad. Aunque se hace una propuesta global que abarca todo el proceso de desarrollo, el trabajo de esta tesis se centra en describir detalladamente las actividades del nivel conceptual.

El modelo InSCo permitirá la fusión de metodologías heredadas de la Ingeniería del Conocimiento (InCo) y de la Ingeniería del Software (InSo), teniendo por objetivo integrar bajo una misma descripción las técnicas y métodos utilizados y aprendidos por los profesionales de ambas ingenierías. No se plantea la construcción de una nueva metodología, porque los ingenieros son ya usuarios de alguna de ellas y trasladarse a un nuevo enfoque creemos que no es una propuesta viable por el coste que supone el cambio. De esta manera se facilitará la implantación de InSCo.

Este modelo guiará la construcción de sistemas software híbridos en los que las soluciones deben venir de la mano de software basado en conocimiento, puesto que se han de resolver problemas poco estructurados con requerimientos subjetivos y donde la incertidumbre, la incompletitud y la inconsistencia son el común denominador; pero además tenemos que integrar en la solución

funcionalidad no basada en conocimiento, imprescindible para el éxito de la aplicación. Desde la perspectiva de los usuarios de estos sistemas toda la funcionalidad debe ser homogénea y la tecnología (basada o no en conocimiento) debe serle trasparente.

El material presentado en esta memoria se organiza en siete capítulos. En el primero, capítulo 1, se realiza el planteamiento global del problema abordado en esta tesis, cuyo primer paso hacia la solución es el modelo de proceso InSCo que se propone como resultado principal del trabajo de investigación. Este problema puede ser enunciado como la ausencia de una metodología integrada que sea aplicable a la construcción de sistemas software que presenten prestaciones tanto basadas en conocimiento, como no. Finaliza este capítulo, indicando que el alcance de esta tesis es la definición de un modelo de proceso completo para esta metodología integradora y la descripción detallada de las actividades del nivel conceptual.

El capítulo 2 recopila y estudia las metodologías InSo e InCo más relevantes prestando especial interés a su adecuación para el desarrollo de un sistema software híbrido. Se revisa el modelo de proceso propuesto en cada una de ellas y como podría adaptarse para emplearlo en este tipo de sistemas, haciendo especial hincapié en soluciones específicas aportadas para el desarrollo de sistemas híbridos. Las propuestas de los diferentes autores ha sido analizadas indicando sus deficiencias y como pueden ser resueltas con la propuesta InSCo. Este análisis sirve de base para identificar las características deseables del modelo de proceso. Puesto que el objetivo es la integración, se seleccionan tres de las metodologías más ampliamente utilizadas en InSo e InCo que se llamarán metodologías soporte, para que el modelo de proceso propuesto en este trabajo pueda ser adoptado tanto por los ingenieros del software como por los ingenieros del conocimiento experimentados en estas metodologías.

En el capítulo 3 se plantea un metamodelo a partir del cual se realiza la descripción de las actividades y los artefactos que se manejan y se generan en el modelo de proceso InSCo. El modelo viene definido por las actividades fundamentales: Organización, Requisitos, Conocimiento, Análisis, Diseño e Implementación. Estas actividades suponen una cobertura de las metodologías soporte y de los estándares. El principal artefacto generado durante la ejecución del modelo de proceso InSCo es el conjunto de modelos: Modelo de Negocio,

Modelo de Servicios, Modelo de Conocimiento, Modelo de Análisis, Modelo de Diseño, Modelo de Implementación y Modelo de Prueba.

Como se ha indicado, se acota en la descripción detallada al nivel conceptual, es decir sólo los aspectos independientes de la implementación, que se tratan en los siguientes capítulos de la memoria.

El capítulo 4 describe la actividad fundamental Organización en la que se genera el Modelo de Negocio (NEG) y el documento de definición del sistema (DOC_DS). Este modelo tiene por objetivo principal la compresión y obtención de información sobre del contexto organizacional, donde el sistema software híbrido será implantado. Se realiza la descripción de la estructura y contenido del modelo de negocio, su descomposición en actividades o tareas y la descripción breve de las técnicas más apropiadas para la ejecución de esta actividad fundamental.

Las actividades fundamentales de Requisitos y Conocimiento se describen en el capítulo 5. Cualquier sistema software híbrido se debe abordar desde dos perspectivas de modelado: desde el modelo de servicios (SER) y desde el modelo de conocimiento (CON). Esto da lugar a que se generen dos puntos de vista diferentes del sistema. Este capítulo recoge la estructura y contenido de ambos modelos, planteando también como usar el documento estándar de la IEEE Std 830-1998 o su equivalente electrónico como resultado de estas actividades fundamentales (DOC_ERS). Para demostrar su aplicabilidad con otros mecanismos de representación del conocimiento, diferentes a los habituales en las metodologías soporte, se describe de forma breve como instanciar la actividad fundamental Conocimiento para el modelado con redes Bayesianas.

La actividad fundamental Análisis, descrita en el capítulo 6, es aquella donde se realiza el modelado conceptual desde el punto de vista de los desarrolladores y está estrechamente ligada al paradigma y la plataforma de desarrollo, siendo el punto de conexión con la implementación. En el capítulo 6 se propone la utilización de técnicas que midan la similaridad entre el modelo de conocimiento y los servicios que debe ofrecer el sistema, para definir el alcance de las componentes basadas en conocimiento del sistema software híbrido. Se trata de una primera aproximación a la ejecución de esta tarea que debe desarrollarse en profundidad en futuros trabajos de investigación.

Para concluir en el capítulo 7 se recogen las conclusiones del trabajo de investigación presentado en esta memoria, junto con la descripción de los

previsibles trabajos futuros. También se relacionan las publicaciones relevantes generadas a partir de este trabajo.

En esta memoria también se han incluido los siguientes apéndices: Apéndice A, con la descripción de un proyecto sencillo aplicando InSCo, Apéndice B, artefactos de un caso complejo de la gestión del protocolo de producción integrada en agricultura, Apéndice C, descripción breve de la herramienta InSCo Requisite.

1 Motivación y alcance de la tesis

En este capítulo se realiza el planteamiento global del problema abordado en esta tesis cuyo primer paso hacia la solución es el modelo de proceso InSCo que se propone como resultado principal de este trabajo de investigación. Este problema puede ser enunciado como la ausencia de una metodología integrada que sea aplicable a la construcción de sistemas software que presenten prestaciones tanto basadas en conocimiento como aquellas no basadas en él.

En la sección 1.2 "Necesidad de integración del InSo e InCo" se justifica porque es necesaria la colaboración de la ingeniería del software y del conocimiento en la definición de una metodología integradora, para en las secciones 1.3 "Acotación del concepto de metodología" y 1.4 "Hipótesis de partida y objetivos" fijar los límites de este trabajo de investigación en la definición de un modelo de proceso completo de la metodología, y en la descripción detallada de las actividades del nivel conceptual, indicando la hipótesis de partida y los objetivos específicos a conseguir con el trabajo de tesis. En la sección 1.5 "Contexto del trabajo investigador" se describen los proyectos, las líneas de trabajo del grupo de investigación bajo las que se ha desarrollado esta tesis. El capítulo finaliza con la presentación de los ejemplos que se tratarán a lo largo de esta memoria, y la herramienta CASE (Computer Aided Software Engineering) que se ha construido para algunas de las actividades fundamentales propuestas para el modelo de proceso InSCo.

1.1 Ingeniería del software e ingeniería del conocimiento

El software está presente en sistemas de todo tipo: transportes, sanidad, telecomunicaciones, militares, procesos industriales, entretenimiento, productos de oficina, etc.; la lista cubre todas las actividades humanas. No podemos ser ajenos que en el epicentro de la evolución que ha sufrido la sociedad actual, hasta llegar a lo que se ha denominado sociedad de la información, se encuentra la evolución acaecida en la construcción y utilización de las aplicaciones informáticas. Estas aplicaciones conducen a la toma de decisiones comerciales, sirven como base a la investigación científica moderna, contribuyen a la resolución de problemas de ingeniería y son el factor clave que diferencia los productos y servicios modernos. Son muchos los casos en los que las personas dejan su trabajo, bienestar, seguridad, entretenimiento, decisiones y sus propias vidas en manos de software informático. Todo esto ha llevado a un cambio en la percepción pública del software.

El software debe desarrollarse como cualquier otro producto industrial aplicando métodos de ingeniería que aseguren la calidad del producto resultante. Aparece así la ingeniería del software (InSo) que el Institute of Electrical and Electronics Engineers (IEEE), define *como "la aplicación de una aproximación sistemática, disciplinada y cuantificable para el desarrollo, operación y mantenimiento de software, es decir, la aplicación de la ingeniería al software"* (Institute of Electrical and Electronics Engineers Staff , 1990). La InSo como disciplina ha ido evolucionando desde que se acuñó el término en la comisiones promovidas por el comité científico de la Organización Nacional del Tratado del Atlántico Norte (OTAN) (Naur & Randell, 1969) y (Randell & Naur, 1970), y se puede refinar la definición original ofrecida por el IEEE cómo: *"es una disciplina que adopta principios de ingeniería como son metodologías conocidas, procesos, herramientas, estándares, métodos de gestión y de organización, sistemas de control de calidad y cualquier cosa que necesite el desarrollo de software a gran escala para obtener un resultado de alta productividad, bajo coste, calidad controlada, y una planificación del desarrollo medible"* (Wang & Patel, 2000).

Pero la InSo comprende un conjunto de actividades y unos dominios de aplicación lo suficientemente diversos como para considerar la necesidad de otras ingenierías, que se complementen con la propia InSo de cara a mejorar la productividad, el grado de satisfacción y de calidad del software construido. Esto ocurre con la llamada ingeniería de requisitos (Kotonya & Sommerville, 1998),

ingeniería de los sistemas en tiempo real (Shumate & Keller, 1992) o ingeniería Web (Institute of Electrical and Electronics Engineers Staff, 1999). También es importante destacar la colaboración que existe entre la InSo y otra ingeniería con la que mantiene una relación histórica, la ingeniería del conocimiento (InCo), disciplina que tiene un objetivo muy similar: *"Construir sistemas basados en conocimiento (SBC) de manera controlable, lo que requiere un análisis y mantenimiento del propio proceso de construcción y el desarrollo de métodos apropiados, lenguajes y herramientas para el desarrollo sostenible de estos sistemas"* (Studer, Fensel, Decker, & Benjamins, 1999). Quienes han trabajado e investigado en ambas disciplinas han llegado a la conclusión de que no sólo son complementarias, sino que cualquier ingeniero informático debería conocer los procedimientos, técnicas, métodos y herramientas tanto de la una como de la otra, para poder desarrollar adecuadamente su labor (Juristo & Acuña, 2002). Esta convergencia es esencial para evitar la duplicidad de esfuerzos y un objetivo importante sería el desarrollo de una metodología común para la producción de software que sirva coherentemente a ambas ingenierías. Esto permitiría, además de optimizar recursos, producir más eficientemente un producto más robusto, fiable y barato, que es en definitiva el fin último de cualquier ingeniería.

1.2 Necesidad de integración entre InSo e InCo

El desarrollo alcanzado en la Inteligencia Artificial (IA) en las últimas décadas, ha propiciado la evolución de la InCo como disciplina. Su objetivo principal ha sido en de convertir el proceso de desarrollo de SBC de un arte en un proceso de ingeniería, al igual que en su día ocurrió con la InSo. Esto lleva asociado la definición de métodos apropiados, lenguajes y herramientas especializadas para el desarrollo este software, entendidos estos como el software cuyo método de resolución es más heurístico que algorítmico (Gómez, Juristo, Montes, & Pazos, 1997).

La evolución que ha sufrido la InCo es similar a la que ha sufrido la InSo, teniendo en cuenta la década de adelanto a favor de la InSo. Puesto que ambas proponen la construcción de software utilizando principios de ingeniería, deberían existir semejanzas entre los métodos, técnicas y herramientas empleadas en ambas. Sin embargo, a lo largo de su evolución estás disciplinas se han ignorado mutuamente, lo que va en contra de los algunos de principios

básicos de cualquier ingeniería como la reutilización, partición de trabajo o cooperación (Wang & Patel, 2000).

El motivo básico del desencuentro de ambas, es el producto resultante, en un caso es software basado en conocimiento y en otro no. La gran diferencia entre estos tipos de software viene dada por los problemas a los que se aplican. Para la InSo son básicamente problemas procedimentales y sistemáticos trabajando con datos y algoritmos; y para la InCo son heurísticos y declarativos, siendo los conocimientos su material de trabajo esencial. En la tabla 1.1 se muestra un resumen de algunas de las diferencias entre ambos tipos de sistemas software (Alonso, Juristo, & Pazos, Trends in Life-cycle Models for SE and KE proposal for Spiral-Conical life-cycle Approach, 1995).

Tabla 1.1 Diferencia entre software basado y no basado en conocimiento.

Parámetro de comparación	**Software no basado en conocimiento**	**Software basado en conocimiento**
Tipo de tarea	Transacción de información o soporte a la decisión	Igual que experto, debe explicar sus resultados y aprender
Nivel del problema	Operacional, Táctico	Estratégico
Estructuración del problema	Estructurado	Poco estructurado
Requisitos	Objetivos	Subjetivos
Personal implicado	Usuarios, Ingenieros del software	Usuarios, Ingenieros del software e ingenieros del conocimiento, especialistas y expertos
Nivel de precisión e incertidumbre	Determinista	No determinista

Otra situación que plantea la necesidad de integración, o al menos de coordinación se presenta cuando es necesario construir sistemas software para resolver los problemas complejos que aparecen en diversos dominios de aplicación. En muchos casos ese software necesitará utilizar técnicas de InCo para resolver la tarea que tenga encomendada, conjuntamente con funciones que se deben de resolver con un sistema de información tradicional o de

procesamiento de transacciones. A este tipo de sistemas, y siguiendo la definición dada en (Chen, Kendal, Pott, & Smith, 1996), los denominaremos sistemas híbridos o sistemas software híbridos para referenciar específicamente la parte software del sistema. La mayoría de estos sistemas se organizan en subsistemas, algunos de los cuales realizan tareas intensivas en conocimiento relacionadas con el proceso de toma de decisiones, mientras que otros realizan tareas de procesamiento de transacciones y gestión de información en bases de datos. Por tanto, para el desarrollo de estos sistemas híbridos, se deben integrar técnicas y actividades de ambas ingenierías para obtener el mejor resultado para el sistema software resultante (Acuña, Lopez, Juristo, & Moreno, 1999), (Águila I. M., Cañadas, Palma, & Túnez, 2006) y (Cañadas, Orellana, Águila, Palma, & Tunez, 2009).

La falta de cooperación se puede resolver si cada disciplina aporta a la otra aquello en lo que tiene más potencial, más experiencia o más expresividad. Las interacciones puntuales entre ambas han sido habituales, fructíferas y provechosas por el efecto sinérgico logrado (Chang, Handbook of Software Engineering and Knowledge Engineering, Volume 1, 2002), (Chang, Handbook of Software Engineering and Knowledge Engineering, Vol 2 Emerging Technologies, 2002) y (Meziane & Vadera, 2009). Por ejemplo, la InCo puede aprender de la InSo acerca de la producción, reutilización, mantenimiento y gestión del proceso (Águila I. M., Túnez, Cañadas, Bosch, & Marín, 2001), y la InSo puede aprender de la InCo acerca de técnicas de adquisición de información, para mejorar la comunicación con los clientes y obtener especificaciones que se ajusten mejor a sus necesidades (Briand, 2002), o bien utilizar técnicas de Inteligencia Artificial en la mejora de los procesos de la InSo (Sagrado & Águila, 2009) o bien la aplicación de técnicas de búsqueda, también llamada Search Based Software Engineering (SBSE) (Harman & Jones, Search-based software engineering, 2001) y (Harman, The Current State and Future of Search Based Software Engineering, 2007). Sin embargo, en otros campos como el modelado conceptual InSo y InCo han desarrollado sus propias técnicas de forma aislada (Dieste, Juristo, Moreno, Pazos, Chang, & Sierra, 2002), aunque como se verá a lo largo de esta tesis existen muchos puntos comunes que pueden tender a la unificación (Cañadas, Orellana, Águila, Palma, & Tunez, 2009).

El Sistema SAEPI (Sistema de Asesoramiento Experto en Producción Integrada), es un claro ejemplo de sistema software híbrido. SAEPI proporciona

soporte en la toma de decisiones de los técnicos agrícolas y agricultores, para el control terapéutico de plagas y enfermedades que afectan a los cultivos del sudeste de España, y que ha sido desarrollado por nuestro grupo dentro del marco de un proyecto de investigación Proyecto "Un Sistema Inteligente para la Toma de Decisiones en el Entorno Agrícola del Sudeste Español" (CICYT-FEDER, 1998-200 (Gómez-Skarmenta, Vila, & Túnez, 2000) y (Cañadas, Águila, Bosch, & Túnez, 2002). Para el desarrollo de este sistema se ha aplicando CommonKADS (CK) (Schreiber, et al., 1999) y Rational Unified Process (RUP) (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999). La mayoría de los problemas se plantearon en el proceso de implementación e integración de subsistemas que se habían desarrollado de forma aislada con cada metodología (Águila I. M., Cañadas, Palma, & Túnez, 2003)

Otro problema, que también apareció durante el desarrollo de SAEPI, y que puede extenderse al desarrollo de sistemas software híbridos en general, se presentó al realizar demasiado pronto la asignación a una categoría software concreta un conjunto de funcionalidades o tareas. Algunas de éstas asignadas inicialmente para ser resueltas con un SBC, tras su estudio se han convertido en más analíticas de lo esperado y con la aplicación de técnicas de InSo podrían ser resueltas con mejores resultados o casi seguro más eficientemente en términos de recursos de desarrollo, que aplicando técnicas de InCo. Este es el caso de la tarea relativa a la selección del subconjunto de plantas a monitorizar en una parcela. Inicialmente se detectó que era necesario aplicar conocimiento del experto para la selección de las plantas del muestreo, pero, tras un estudio más exhaustivo del problema, el propio sector agronómico decidió hacer una selección aleatoria de las plantas. Por otra parte, objetivos planteados inicialmente como transaccionales se consideró que hubiese sido más oportuno utilizar técnicas de análisis o de implementación más cercanos a la InCo que a la InSo, como es el caso con la tarea de identificación de los patógenos, que SAEPI postergó para nuevas versiones del sistema. Esto ha llevado a la necesidad de definir una propuesta de integración de metodologías de InSo e InCo en donde cooperan actividades y modelos heredados de cada una de ellas, y que se integran bajo la misma descripción el sistema completo a construir. Para llevar a cabo la integración, el primer paso consistió en delimitar el alcance de la integración: ¿definiremos métodos completos, técnicas, notación común?

1.3 Acotación del concepto de metodología

Cuando se desarrolla software, esté basado en conocimiento o no, se debe seguir una determinada metodología, cuya elección depende, entre otros factores, del dominio de aplicación, la propia experiencia y las herramientas de soporte de las que se dispongan. Estas metodologías guían el proceso de producción, así como, la determinación las actividades a llevar a cabo, junto con el orden de realización de las mismas. Pero, ¿cuál es el alcance de una metodología? En la literatura no se encuentra una respuesta unánime a esta pregunta. En el ámbito de la construcción de software se distingue entre dos grandes significados para la palabra metodología (Cockburn, Selecting a project's methodology, 2000); por una parte está el concepto de metodología que se aborda en los libros clásicos de InSo, la metodología con minúscula[1] donde se describen técnicas y notaciones gráficas para algunos de los papeles a jugar en el desarrollo de software. Y por otra tenemos el significado que aparece cuando se aborda el desarrollo de software desde la perspectiva industrial y comercial; en este caso, se habla de METODOLOGÍA con mayúsculas, donde se trata todo lo referente a como un grupo de trabajo repetidamente produce y entrega sistemas que incluyen software a usuarios finales.

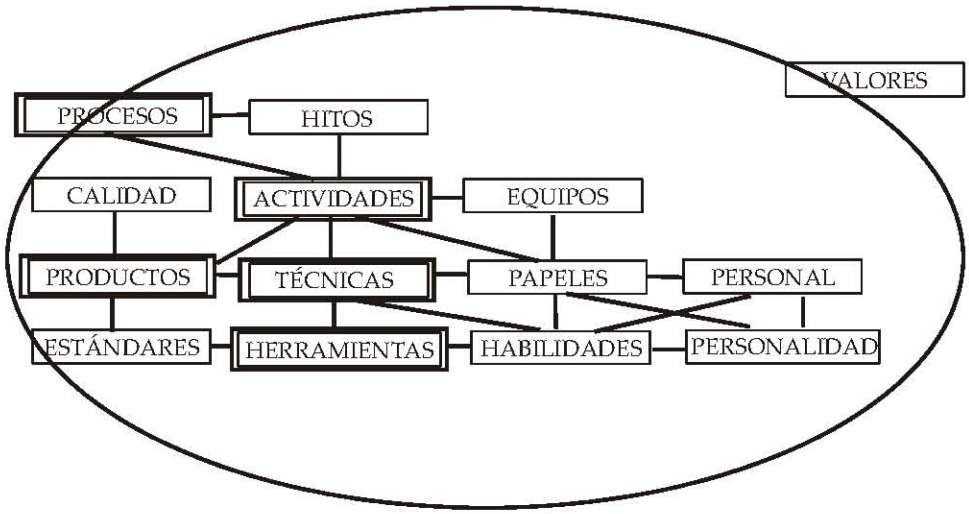


Figura 1.1 Resaltados los elementos de una metodología con minúsculas.

[1] El trabajo referenciado utiliza los terminos little-m y Big-M que han sido traducidos por los de metodología con minúsculas y con mayúsculas respectivamente.

En la figura 1.1 se pueden observar los elementos, que de forma general, describen una metodología con mayúsculas. El *Personal* con sus *Habilidades* y *Personalidades* particulares juega distintos *Papeles* dentro del proyecto trabajando en los distintos *Equipos*. Se utilizan una serie de *Técnicas* para construir *Productos* que siguen unos determinados *Estándares* y que cumplen unos criterios de *Calidad* concretos. La aplicación de las *Técnicas* necesitan *Habilidades y Herramientas*, que facilitan la aplicación de los *Estándares*. Los *Equipos* desarrollan las *Actividades*, que definen el *Proceso* global del proyecto; cuya evolución está marcada por una serie de *Hitos*. Estos elementos tienen que ser compatibles con la escala de valores del equipo, en la que tienen que coexistir la escala de *Valores* del personal con la que emana del proyecto utilizado.

Sobre la figura 1.1 aparecen resaltados los elementos que pertenecen a una metodología con minúsculas. Todos estos elementos y sus relaciones permiten describir y estandarizar el ciclo de vida de una metodología. Este ciclo de vida debe abarcar todos los aspectos involucrados en el desarrollo de software desde la especificación de los requisitos hasta los aspectos ligados al mantenimiento del producto final.

Tabla 1.2 Metodología de Coad y Yourdon.

	Actividades	**Artefactos**
1	Identificar los objetos y las clases del dominio del problema	Lista de clases y objetos especificados
2	Identificar de la estructura del problema del dominio	Diagrama de descomposición de relaciones entre clases
3	Agrupar de clases	Diagrama destacando los grupos de clases
4	Definir atributos de objetos	Un diagrama de clases con atributos
5	Definir servicios de objetos	Diagrama de clases completo

Llegados a este punto conviene establecer una distinción clara entre los elementos proceso y producto ya citados antes, al ser dos términos recurrentes a lo largo de toda esta memoria. Por producto se entiende el conjunto de artefactos elaborados gracias a la utilización de un lenguaje, unas técnicas y unas herramientas, algunos de estos artefactos formarán el software final que se entregue al cliente. Mientras que por proceso o modelo de proceso se entiende el conjunto de actividades encaminadas a la construcción de los artefactos. Un ejemplo se muestra en la tabla 1.2 donde se especifican las actividades (Saeki & Chang, 2002) y los artefactos para una parte del desarrollo en la metodología de

Coad & Yourdon (Coad & Yourdon, Object-Oriented Analysis, 1989) y (Coad & Yourdon, Object-Oriented Design, 1991).

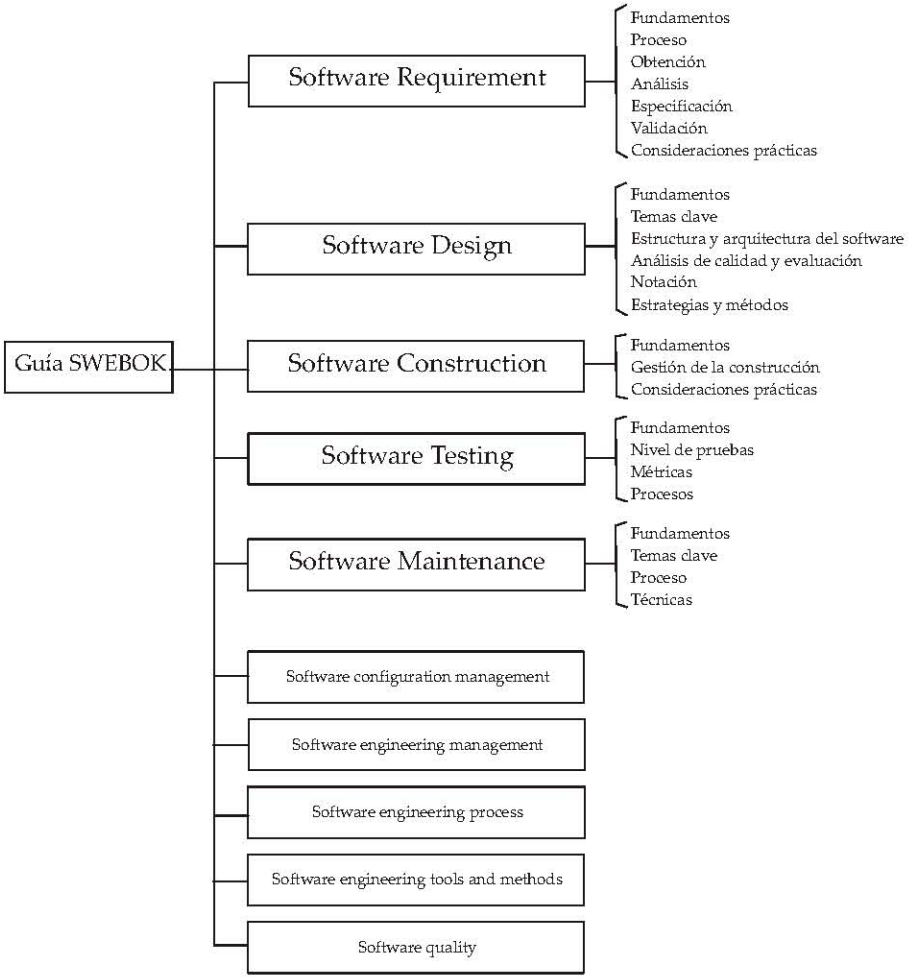


Figura 1.2 Jerarquía de actividades de desarrollo propuestas por SWEBOK.

El proceso establece el marco de trabajo para las actividades que se desarrollan en la construcción de software. Estas actividades tienen una naturaleza muy variada y de forma general se pueden clasificar en dos tipos: actividades de desarrollo y actividades de protección (Pressman, Ingeniería del Software - Un Enfoque Práctico, 2002). Las primeras están estrechamente ligadas al paradigma o aproximación metodológica elegida para el desarrollo, que en la mayoría de los casos se basan bien en el paradigma estructurado o bien en el orientado a objetos, y dependen en gran medida de las características intrínsecas del proyecto. Las segundas son independientes de cualquier actividad de

desarrollo y aparecen durante toda la ejecución el proceso; generalmente se asocian a las tareas de gestión.

Las actividades de desarrollo o proceso se definen en términos de los llamados flujos de trabajo fundamentales o actividades fundamentales[2], que no son más que un conjunto de actividades relacionadas. Cada autor propone diferentes agrupaciones de esas actividades, pero en una situación ideal se debería partir de flujos de trabajo o actividades fundamentales genéricas que se instanciasen en actividades concretas en cada enfoque o paradigma y, de esta forma, obtener un marco común, que como ya se ha dicho, facilitase la integración de las metodologías, independientemente de su categoría. Una referencia básica a estas actividades genéricas son las actividades propuestas en el proyecto Software Engineering Body of Knowledge (SWEBOK) (Abran, Moore, Bourque, Dupuis, & Tripp, 2004) que en su jerarquía básica de temas, que se reproduce en la figura 1.2, los cinco primeros temas se corresponden con estas actividades genéricas relativas a la construcción de software. Este modelo de proceso es el elemento unificador de las metodologías de InSo y el punto de partida de este trabajo de investigación.

1.4 Hipótesis de partida y objetivos

Recordemos que el problema que se plantea en este trabajo de investigación es la necesidad de utilizar una metodología que permita desarrollar productos software de calidad donde se integren técnicas y métodos de la InSo y de la InCo, puesto que son muchos los casos en los que las empresas u organizaciones requieren la implantación de sistemas software donde se integren de forma trasparente componentes basadas y no basadas en conocimiento.

Nuestra intención es definir una metodología con minúsculas aplicable en el desarrollo de proyectos software híbridos, pero teniendo por objetivo la integración y adaptación de los métodos ya existentes, definiendo un marco común de referencia que se pueda instanciar y adaptar a los equipos de desarrollo y la naturaleza del proyecto a desarrollar. Con este planteamiento se minimiza el coste del cambio para los profesionales de la InCo y de la InSo.

---

[2] El término en inglés es workflow, que ha sido traducido por actividad fundamental.

Para ello realizaremos la adaptación y redefinición de modelos de proceso conocidos, centrándonos en la definición de alto nivel de las tareas de desarrollo de software en el ámbito de sistemas híbridos, dejando un lado las actividades de protección o gestión. Este modelo de proceso propuesto, al que denominaremos InSCo, permitirá la integración de metodologías heredadas de la InCo y de la InSo, ofreciendo un punto de referencia para dar soporte lógico al desarrollo de un proyecto software a través de la definición de fases, etapas, acciones, hitos de entrega de versiones, artefactos, métodos, documentación, etc. Para poder poner en práctica este modelo de proceso es necesario definir además un conjunto de técnicas, herramientas, métodos y actividades concretas ligados a cada una de estas actividades fundamentales, así como criterios que permitan identificar, ordenar, evaluar y usar dichas técnicas, herramientas y métodos. De esta forma, además del marco general, se tiene un algoritmo para su instanciación y ejecución.

1.4.1 Alcance de la propuesta

En este apartado se realiza una puntualización importante en el desarrollo de sistemas híbridos sea cual sea la metodología específica que se utilice: La integración de InSo e InCo se realiza en el nivel conceptual.

En la literatura nos encontramos con dos grandes grupos de actividades de modelado durante el proceso de desarrollo de software: el modelado conceptual y el modelado computacional (Blum, 1996). Lo que lleva a dos tipos de modelos: los modelos conceptuales y los computacionales. Esta misma clasificación nos la encontramos los trabajos ligados a Modelo Driven Architecture (MDA) (Object Management Group (OMG), 2003) que mantiene esta separación definiendo el modelo independiente de la plataforma PIM (Platform Independent Model), que describe la funcionalidad del sistema, pero omitiendo detalles sobre cómo y dónde va a ser implementado y el modelo para una plataforma concreta PSM (Platform Specific Model), más cercano al diseño.

Como se muestra en la figura 1.3 el o los modelos conceptuales son el vínculo de unión entre el universo del discurso y el sistema a construir, si el problema modelado se resuelve y se resuelve con software, será un sistema software. El nivel de modelado conceptual es independiente de la implementación y es trabajo de los desarrolladores hacerlo corresponder con un

modelo computacional, de forma que dependiendo del tipo de enfoque de desarrollo el modelo computacional podrá ser orientado a objetos, estructurado o del ámbito de las técnicas basadas en conocimiento.

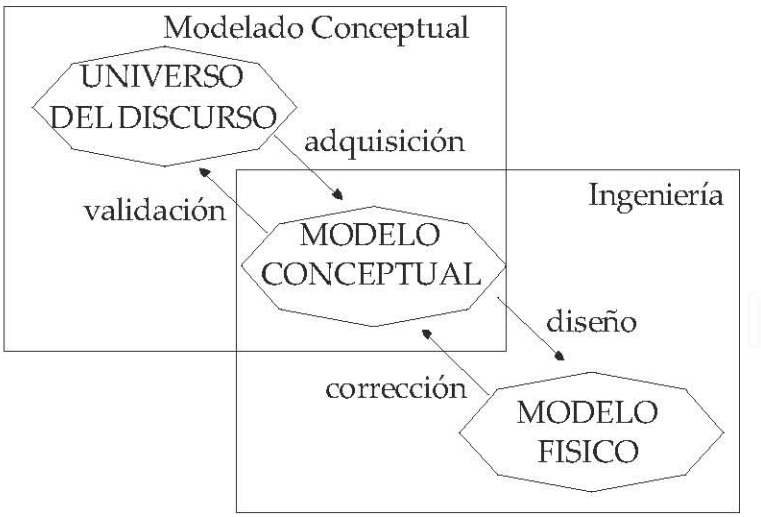


Figura 1.3 Modelado conceptual como vía de conexión con el modelo físico.

En la última parte del desarrollo de software, diseño e implementación, es donde aparecen las diferencias más claras entre los planteamientos de la InCo y de la InSo, ya que las técnicas y herramientas aplicables en cada disciplina son muy diferentes. Por lo tanto, es en el nivel conceptual donde se puede plantear la unificación para las componentes basadas y no basadas en conocimiento del sistema híbrido, se resuelva con el método que se resuelva.

1.4.2 Objetivos específicos

El objetivo principal de esta tesis es la propuesta de un modelo de proceso para el desarrollo de un sistema híbrido, que llamaremos modelo de proceso InSCo. La consecución de este objetivo implica el desarrollo de los siguientes sub-objetivos:

- Estudiar los puntos comunes de las metodologías de la InCo y de la InSo, identificando los elementos similares y características que deberán incluirse en la propuesta.

- Realizar la adaptación y redefinición de los modelos de proceso conocidos, definiendo las actividades fundamentales y artefactos generados y utilizados, conformado el modelo de proceso InSCo.
- Comprobar que InSCo es una cobertura de las metodologías soporte y estándares conocidos.
- Para cada actividad fundamental en el nivel conceptual definir su descomposición en tareas y fijar un posible conjunto de técnicas, herramientas y métodos que se puedan utilizar para su ejecución
- Validar el modelo con su instanciación para el caso concreto del seguimiento de cultivos en producción integrada en agricultura.
- Definir y planificar el desarrollo de un prototipo que facilite la ejecución del modelo InSCo.

Si se alcanza estos objetivos tendremos un marco metodológico que minimizará las vueltas atrás en el desarrollo y evitará que requisitos asignados inicialmente para ser resueltos con un SBC, evolucionen hacia componentes software no basados en conocimiento o viceversa, posponiendo la decisión del tipo de técnicas y métodos a utilizar, hasta que se puedan aplicar criterios concretos que nos hagan optar por una o por otra.

Además, puesto que no existe una única forma de desarrollar proyectos software, el modelo de proceso InSCo a de ser flexible y genérico, permitir la configuración de la ejecución del proceso de forma que se ajuste al proyecto y/o al equipo de desarrollo. La flexibilidad y adaptabilidad se basa en la definición de las actividades fundamentales que colaborarán para evitar la duplicidad de esfuerzos si se desarrollasen de forma aislada y en paralelo las componentes basadas y no basadas en conocimiento.

1.5 Contexto del trabajo investigador

El trabajo de investigación se ha realizado dentro del grupo de investigación de la Universidad de Almería: Ingeniería de Datos, del Conocimiento y del Software (DKSE) TIC-181 [3], que actualmente trabaja en

[3] www.dkse.ual.es

distintos campos de la informática, centrando sus líneas de investigación en Bases de Datos e Ingeniería del Conocimiento e Ingeniería del Software, trasladando los resultados a diversos campos de aplicación como la gestión de información basada en la Web, sistemas de apoyo a la toma de decisiones, en especial en el dominio agrícola. En el ámbito de los Sistemas de Asesoramiento en Agricultura, entre 2004 y 2006 se ha obtenido una financiación para tres convenios de colaboración con la Dirección General de la Producción Agraria de la Consejería de Agricultura y Pesca de la Junta de Andalucía. Actualmente estamos transfiriendo aplicaciones desarrolladas por el grupo al portal Besana[4] .

Este grupo desarrolló, durante los años del 1998 al 2002, un proyecto financiado por los fondos CICYT y Fondos FEDER, en colaboración con las Universidades de Murcia y Granada, titulado "Un sistema inteligente para la ayuda a la toma de decisiones en un entorno agrícola del sudeste español" y donde el grupo DKSE tuvo por objetivo primordial desarrollar el sistema software híbrido SAEPI que proporciona soporte en la toma de decisiones en el dominio del control terapéutico de plagas y enfermedades que afectan a los cultivos del sudeste de España. Las dificultades acaecidas durante la ejecución de este proyecto fue uno de los detonantes de la propuesta InSCo. Esta propuesta comenzó a fraguarse como resultado de la estacia realizada en la Universidad Politécnica de Madrid durante Septiembre del 2003, concretamente en la unidad de Ingeniería del Software del Dpto de Lenguajes y Sistemas informáticos e Ingeniería del Software, de la que es responsable Natalia Juristo Juzgado.

Por otra parte, la historia docente desarrollada a lo largo de casi una década por la doctoranda en el campo de la InSo y las actividades del grupo de investigación ligadas a la Ingeniería de Requisitos, que han llevado a la firma un convenio entre la Universidad de Almería y la empresa es TCP Sistemas e Ingeniería, actualmente Visure Solutions[5], y con la que conjuntamente se han desarrollado varias jornadas de de divulgación, dieron lugar a algunas de las tareas del proyecto bajo el cual se ha desarrollado este trabajo de tesis.

La mayor parte de esta tesis se ha desarrollado en el marco del proyecto "Metodología para el análisis y diseño de sistemas híbridos: Integración de

[4] www.portalbesana.es

[5] www.visuresolutions.com

técnicas de modelado de software y de conocimiento" (TIN2004-05694). Este proyecto tenía un doble objetivo. Por una parte, la definición de una metodología que permitiera el desarrollo de sistemas software híbridos. Por otra, la validación de la metodología propuesta aplicándola en el desarrollo de un producto software para ayuda en la toma de decisiones en el seguimiento de cultivos bajo la norma de calidad de la Producción Integrada, el sistema SAVIA (Sistema de Ayuda y Vigilancia Integrada en Agricultura).

El proyecto fue planteado paralelamente en el nivel conceptual y en el computacional. En el primero de ellos es donde se enmarca claramente este trabajo de tesis y la definición del modelo de proceso InSCo y en el segundo, en desarrollo por parte de otros miembros del grupo de investigación, se ha optado por enriquecer el modelo de diseño de CommonKADS aplicando técnicas no contempladas en esta metodología y originarias de la InSo. De esta forma se facilita la transformación de los modelos conceptuales a modelos de diseño utilizando tecnologías propias de Arquitectura Dirigidas por Modelos (Model Driven Architectures, MDA), ya que proporcionan un enfoque de desarrollo de software basado en modelado y permite la definición de transformaciones entre modelos de forma que puedan ser automatizadas (Cañadas, Palma, & Túnez, 2009). Otra línea de trabajo reciente, que también saca partido a la colaboración entre InCo en InSo, ha dado resultado positivos en el campo de la modelización con redes bayesianas (Sagrado & Águila, 2009) y (Águila, Sagrado, Túnez, & Orellana, Julio 2010 - Aceptado pendiente celebración).

1.6 Casos de estudio y Herramienta InSCo Requisite

Cuando se desarrollan proyectos software de tamaño mediano o grande es necesario manejar y compartir mucha información entre todos los que desarrollan y colaboran en el desarrollo del proyecto. Para obtener el éxito en el proyecto es necesario mantener esta información, utilizando herramientas software que ayuden en la elaboración de los artefactos generados por la metodología utilizada, y sobre todo herramientas para mantener la integridad frente a cambios y la administración de varios proyectos de desarrollo a la vez, al tiempo que deben permitir la interacción y colaboración entre todos los implicados del proyecto, especialmente en las tareas de especificación de las necesidades (Sinha, Sengupta, & Chandra, 2006) y (Lang & Duggan, 2001). Existen herramientas CASE que facilitan todas estas tareas y que están

disponibles para muchas de las metodologías de desarrollo actuales y para las distintas fases o actividades de cada una de ellas, así como para las tareas de gestión del proyecto software.

La propuesta InSCo, aunque para este trabajo de investigación sólo se centrará en la definición de un modelo de proceso y la descripción detallada del nivel conceptual, también necesita ayudarse de herramientas software. Se han especificado y construido prototipos de lo que serían estas herramientas para la implantación del modelo de proceso InSCo[6]. Así la herramienta InSCo Requisite está basada en tecnología Web y permite la administración y control de los proyectos de desarrollo de software de forma distribuida, definiendo responsables y usuarios para cada proyecto. La figura 1.4 muestra la interfaz inicial de esta herramienta.

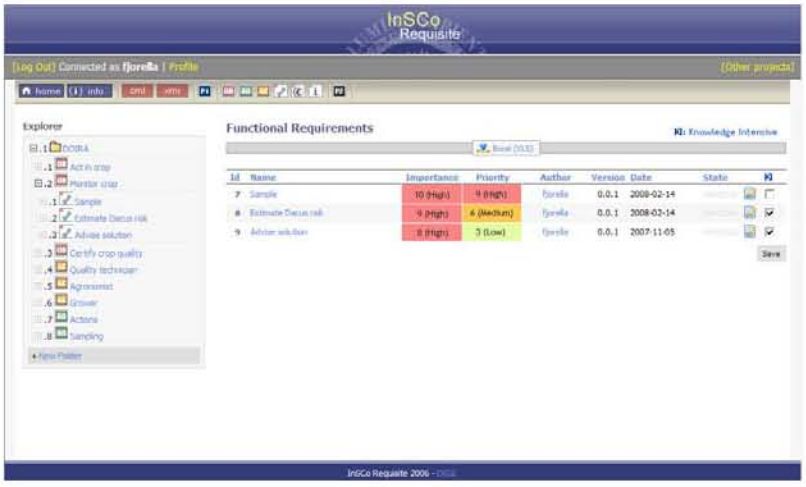


Figura 1.4 Interfaz de InSCo Requisite.

A lo largo de esta memoria se empleará un caso de estudio que se irá desarrollando para la mejor descripción de la propuesta. El proyecto Arrendamientos, es un caso sencillo que ha sido seleccionado por su fácil comprensión y que sólo en circunstancias extremas podría considerarse como un sistema software híbrido. Por el contrario, el caso sistema software híbrido SAVIA (Sistema de Ayuda y Vigilancia Integrada en Agricultura), incluido en el Apéndice B, debido su complejidad si puede considerarse como un sistema

[6] www.dkse.ual.es/insco

híbrido. La descripción inicial del proyecto Arrendamientos se incluye en la tabla ejemplo 1 y la figura 1.5 muestra la utilización de InSCo Requisite para este proyecto.

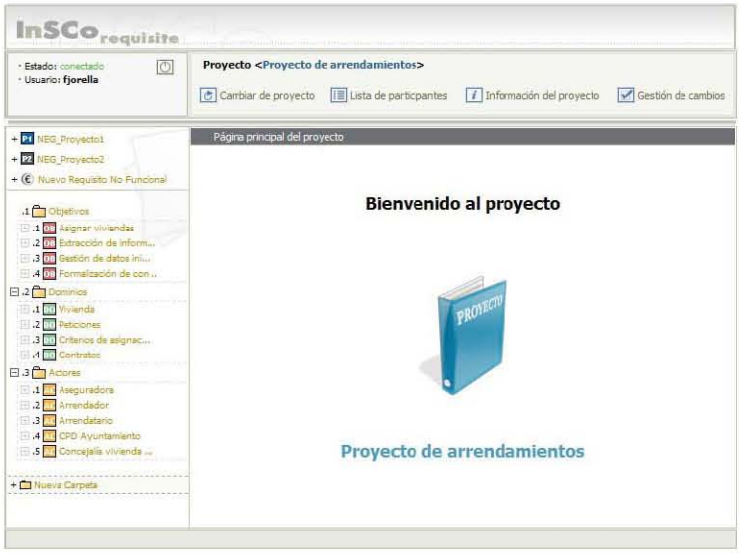


Figura 1.5 Proyecto Arrendamientos en InSCo Requisite.

Ejemplo 1 Descripción del proyecto Arrendamientos.

Debido a la falta de viviendas en alquiler, desde los gobiernos locales se ha promovido la iniciativa de disponer de un servicio de control y asignación de arrendamientos.

La asignación de viviendas se hace desde el ayuntamiento. Las personas que desean alquilar una vivienda se registran como potenciales "arrendatarios". Cada mes se revisa la lista de viviendas y de potenciales arrendatarios para estudiar su asignación. A partir de esta asignación se publica un resumen de los datos de ese proceso de asignación para que los potenciales arrendatarios si lo desean cambien sus prioridades, por ejemplo, seleccionando un área menos popular.

Por una parte se hace necesario disponer de una base de datos con la información relativa a los arrendatarios y a las viviendas, así como quien es el arrendador de cada una de ellas.

Ligado a cada contrato de alquiler se negocia un seguro que permitirá caso de falta de pago o caso de desperfectos en la vivienda cubrir los gastos ocasionados al finalizar el arrendamiento.

Para elegir una vivienda, los potenciales arrendatarios deberán satisfacer una serie de criterios. Existen cuatro tipos de criterios, primero se selecciona la categoría de la vivienda (zona), después el número de habitaciones correspondiente con las necesidades del arrendatario y la vivienda. El tercero sería la correspondencia entre los ingresos del potencial arrendatario y la renta pactada para la vivienda. Finalmente se pueden especificar condiciones particulares para cada vivienda como no fumadores o sin mascotas, etc.

	Edad			
Ingresos	18-27	28-64	+65	Renta mensual
	0-7.999	0-6.999	0-4.999	250
	8.000-11.999	7.000-10.999	5.000-7.999	400
	12.000-15.999	11.000-15-999	8.000-11.999	500
	16.000-19.999	16.000-19.999	12.000-19.999	600
	20.000-24.999			700
	25.000-29.999			800
	+30.000			900

1.7 Conclusiones del capítulo

El problema que se plantea es la construcción de software para sistemas híbridos que son aquellos en los que coexisten componentes basados en el conocimiento y sistemas de información convencionales. Los primeros intentan dar solución a las partes del problema en las que los requisitos están poco estructurados y son subjetivos, y en las que además están presentes la incertidumbre, la incompletitud y la inconsistencia, mientras los segundos presentan una solución válida para las partes del problema cuyos requisitos son objetivos, pudiendo tener un resultado cuantitativo o cualitativo.

Actualmente, las metodologías disponibles no resuelven la integración de forma exitosa, si se emplean métodos de InSo se pierde la expresividad necesaria para modelar conocimiento, y los métodos de la InCo no se pueden emplear si no hay conocimiento, tampoco existe un método integrado que aborde la construcción de estos sistemas híbridos como tales.

El problema no se resuelve con la definición de una nueva metodología para este tipo de sistemas, porque existe un gran inconveniente que no es más que el coste del cambio. Los profesionales de la InCo y la InSo son ya usuarios de alguna metodología, y trasladarse a un nuevo enfoque no es una propuesta viable para casi nadie. Por tanto, la solución debe venir de la mano de extender o mejorar las metodologías conocidas con aquello que les falta, o en los términos que se vienen usando en esta memoria, integrar y hacer converger las metodologías usadas por ambas disciplinas.

En esta propuesta se han buscado puntos de convergencia entre las distintas metodologías de InSo e InCo, que en muchos casos utilizan herramientas y notación comunes. El modelo de proceso InSCo supondrá un nuevo punto de convergencia hacia una metodología común para el desarrollo de los sistemas híbridos y que en este trabajo describiremos hasta el nivel conceptual.

2 Estado de la cuestión

En este capítulo se recopilan y estudian las metodologías más relevantes para el desarrollo de software prestando especial interés a su adecuación para el desarrollo de un sistema software híbrido. Se revisa el modelo de proceso propuesto en cada una de ellas y como podría adaptarse para emplearlo en este tipo de sistemas.

La primera sección de este capítulo sintetiza la evolución que han sufrido las disciplinas InSo e InCo desde sus comienzos hasta nuestros días, seleccionando un conjunto representativo de metodologías que se describen en la siguiente sección. La descripción y análisis completo de estas metodologías sería un trabajo de más de una memoria y no justificado en esta tesis, debido por una parte, al gran número de métodos y a sus distintas variantes y, por otra, tal como se indicó en la sección 1.3. "Acotación del concepto de metodología", a que el concepto de metodología es muy amplio y abarca muchas actividades, de hecho existe una disciplina completa Method Engineering únicamente dedicada al análisis de métodos (Brinkkemper, Lyytinen, & Welke, 1996) y (Saeki & Chang, 2002). Por tanto, se ha optado por seleccionar algunas metodologías agrupadas en cuatro categorías: metodologías estructuradas, orientadas a objetos, de primera generación y segunda generación para el desarrollo de SBC.

En la sección 2.3 "Metodologías para sistemas híbridos" se estudian soluciones específicas para el desarrollo de sistemas híbridos. Las propuestas de los diferentes autores se han agrupado en tres líneas de trabajo, cada una ha sido analizada y criticada, indicando cuales son sus deficiencias y como éstas son

resueltas con la propuesta InSCo. Este modelo será un marco común que facilita la integración y hereda de algunas de las metodologías descritas en este capítulo gran parte de sus características fundamentales, tal como se describe en el apartado 2.3.5 "Características deseables de la propuesta InSCo"

Puesto que el objetivo es la integración y no la definición de un modelo de proceso completamente nuevo, como conclusión del estudio del estado del arte en la sección 2.4 "Metodologías soporte" se han seleccionado tres de las metodologías más ampliamente utilizadas en ambas ingenierías que se llamarán metodologías soporte, a saber Estructurada, Rational Unified Process y CommonKADS. De esta forma, el modelo de proceso propuesto en este trabajo puede ser adoptado tanto por los ingenieros del software como por los ingenieros del conocimiento experimentados estas las metodologías.

2.1 Recorrido histórico de InSo e InCo

La ingeniería del software es una disciplina muy amplia y puede ser vista desde distintas perspectivas, que van desde la formación y docencia o la profesionalización, hasta las metodologías de investigación en InSo (Finkelstein & Kramer, 2000). Para obtener una visión completa de la evolución de esta disciplina se deberían revisar todas las perspectivas, pero esta sección se centra en los aspectos relativos a las metodologías para el desarrollo del software. Por otro lado, la InCo ha tenido una evolución similar, aunque las metodologías surgidas al amparo de esta disciplina se centran más en la adquisición del conocimiento que en el desarrollo del software basado en conocimiento, parte software del SBC. Como se propuso en la sección 1.2 "Necesidad de integración entre InSo e InCo" se tiene que buscar la integración de forma que se superpongan y entrelacen los caminos por los que a partir de hoy deben evolucionar ambas disciplinas.

Aunque es difícil condensar la historia de InSo e InCo, extendiendo el trabajo de Endres (Endres, 1996) y de Shu-Hsien (Shu-Hsien, 2005), se ha optado por definir eras disjuntas, cada una de ellas compuesta por dos períodos que pueden solaparse. Cada período se distingue por los objetivos que persigue, los métodos, las herramientas usadas, las lecciones aprendidas y los problemas identificados. La tabla 2.1 resume esta división en períodos y las principales características y metodologías más utilizadas en cada uno de ellos:

Tabla 2.1 Evolución de la InSo y la InCo.

Era	Período	Descripción	Met.
Dominada por la máquina (1956- 1967)	Orientado por lotes	Forzado por el hardware Lenguajes de alto nivel	
	Interactivo	Codificación en línea. Editores y depuradores CODE and FIX	
Dominada por el proceso (1968 –1982)	Centrado en el proceso	Crisis. Proceso de desarrollo Ingeniería del software	SREM SADT DSED JPS SAD
	Formal	Asegurar corrección. Modelos Problemas en grandes sistemas	
Dominada por la complejidad (1983 –1993)	Estructurado	Llegada del PC. Expansión Necesidad convergencia con modelado de datos	Modern SAD JSD OMT Jacobson Booch
	Orientado a objetos	Reutilización Nuevo concepto de la programación	
Dominada por las comunicaciones (1994-2005)	Distribuido	Expansión comunicaciones. Cliente/servidor Desarrollo de proyectos complejos	CORBA RUP/UML CMMI
	Industria de producción de software	Métodos integrados. Proceso de calidad controlada	
Dominada por la productividad (2006-2010)	Abstracción	Desarrollo en el nivel de análisis Interacción cercana al cliente	MDA XP Scrum
	Manifiesto Ágil		
Dominada por la inteligencia (1956- 1978)	Métodos generales		
	S.B.C	Conocimiento	
Dominada por el proceso (1979 –1989)	Centrado en el proceso	Crisis. Ingeniería del Conocimiento Shell	Buchanan KLIC IDEAL
	Especialización	Aplicaciones específicas	
Dominada por la complejidad (1990-1998)	2º Generación de metodologías	Transferencia de SBC Industria del conocimiento	MIKE Protégé CommonKADS
	Reutilización	Librerías de tareas reutilizables Gestión del conocimiento	
Dominada por las comunicaciones (1999-2007)	Aplicaciones distribuidas. WEB	Ontologías Semantic Web	Mas-CK MASE W3C RDF DAML + OIL
	Minería de datos	Disponibilidad de datos. Aprendizaje automático	
Dominada por la productividad (2008-2010)	Extensión	Trasferencia a otro dominios	
	Integración	Modelo de proceso InSCo	

InSo Era I: Dominada por la Máquina (1956-1967). El término ingeniería del software aún no ha sido acuñado. El desarrollo de código está altamente influenciado por fuerzas externas, el principal objetivo del software es explotar los limitados recursos hardware de la forma más óptima posible. Se definen los primeros compiladores que funcionan en compilaciones no interactivas. Estos entornos van evolucionando hasta la definición de las primeras herramientas CASE de bajo nivel que facilitan la edición, la depuración y la compilación en modo interactivo. Cabe destacar la falta de método en el desarrollo de software, lo que da lugar a un alto riesgo y al origen de una nueva era.

InSo Era II: Dominada por el Proceso (1968-1982). Esta era se origina con la llamada crisis del software que da lugar a la definición de la ingeniería del software (NATO Software Engineering Conferences 1968 y 1969) (Naur & Randell, 1969) y (Randell & Naur, 1970). El objetivo es reducir el riesgo en el desarrollo y mejorar la calidad y la productividad. Surgen las metodologías de desarrollo del software para definir y supervisar el proceso de construcción de software. Una aportación de esta era es el enfoque de modelado formal que permite que el proceso de desarrollo posterior a una especificación formal sea automatizable. Pero desde el punto de vista de la industria, las aproximaciones formales se recortaron debido a la falta de herramientas y de formación, además para sistemas grandes los métodos formales se hacen inmanejables. Como conclusión, en esta era se descubre la necesidad de centrarse en las fases anteriores al diseño y la utilización de modelos más o menos formales para la especificación del software. Se desarrollaron varios de los llamados métodos estructurados como Software Requirement Engineering Methodology (SREM) (Alford, 1977), Structured Analysis and Design Technique (SADT) (Ross & Schoman, 1977), que permitían la elaboración de los documentos de especificación para software de gestión.

InSo Era III: Dominada por la Complejidad (1983-1993). Finaliza el dominio tradicional del hardware sobre el software. Llega el ordenador personal y se abre el campo de aplicaciones de la informática, se aborda el proceso de desarrollo de software de forma integral, desde la especificación o análisis hasta el diseño. Se generaliza el uso de herramientas CASE, que aplicando alguna de las metodologías de la familia estructurada o de modelado de datos (Yourdon, Modern Structured Analysis, 1988), (Ward & Mellor, 1985) y (Hatley & Pirbhai, 1988), facilitan el desarrollo de aplicaciones. Se crean las interfaces gráficas de

usuario y la programación visual, lo que acerca el software a los clientes. No obstante el modelado de datos (bases de datos) y de funciones (métodos estructurados) aún discurren por caminos separados. La convergencia de estas disciplinas aparece en los métodos orientados a objetos, que como las metodologías estructuradas, se aplicaron primero a la codificación y al diseño, y por último, a la especificación (Coad & Yourdon, Object-Oriented Analysis, 1989), (Coad & Yourdon, Object-Oriented Design, 1991) y (Booch, Object-Oriented Analysis and Design with Application, 1993). El empleo este enfoque orientado a objetos posibilita la reutilización del software de manera efectiva y por tanto la mejora en la productividad para la construcción de software.

InSo Era IV: Dominada por las Comunicaciones (1994-2004). La aparición de Internet genera un nuevo concepto de software. La descentralización de las funciones y los datos lleva al rápido desarrollo y expansión de áreas de la informática como la programación y arquitecturas distribuidas y concurrentes, que hasta ahora se desarrollaban en un contexto bien delimitado. Además de las aplicaciones cliente/servidor, y en general el desarrollo de sistemas distribuidos, surge una nueva disciplina dentro de la ingeniería del software llamada ingeniería Web (Murugesan & Deshpande, 2001) y (Institute of Electrical and Electronics Engineers Staff, 1999). Por otra parte, el desarrollo de software se ve más como un proceso industrial en la que debe controlarse la calidad y se produce, por tanto, una separación efectiva el proceso y el producto, con tareas ligadas a la gestión y mejora tanto del producto como del proceso, CMM (Capability Maturity Model) y CMMI (Capability Maturity Model Integrated) (Paulk, Curtis, Chrissis, & Weber, 1993)

InSo Era V: Dominada por la productividad (2005- 2010). La mayoría de los sistemas software creados en esta era son los llamados sistemas corporativos; es decir, están orientados a formar parte de importante de los negocios de las grandes empresas, esto ha llevado a sea necesario ajustar las metodologías en términos de aumentar el nivel de abstracción en los ingenieros del software y acercarlo a la abstracción del problema. Se definen nuevas herramientas que permitirán programar en el nivel de análisis, como es el caso de la propuesta de MDA. El otro gran periodo importante de esta era es la aparición de las metodologías ágiles (Cockburn, Agile Software Development, 2001) y (Cockburn, Agile Software Development: The Cooperative Game, 2006). En los proyectos con desarrollo ágil se busca que todos los recursos se empleen en la creación del

mejor software que satisfaga las necesidades del cliente. Esto significa que el equipo de trabajo se concentra únicamente en tareas y procesos que agregan valor al cliente del producto que se está creando, mejorando o implementando. Las metodologías más conocidas son Extreme Programming (Beck & Andres, 2004) y Scrum (Schwaber & Beedle, 2001)

InCo Era I: Dominada por la inteligencia (1856-1978). Aún no se habla de ingeniería del conocimiento, en esta era se origina la Inteligencia Artificial. Se busca desarrollar sistemas que puedan "pensar" tal como lo hacían los expertos humanos. Se dio un primer período donde se buscaban técnicas generales para la resolución de problemas, como el sistema de planificación STRIPS (Stanford Research Institute Problem Solver System), (Fikes & Nilsson, 1971), o GPS (General Problem Solver) (Newell & Simon, GPS, a program that simulates human thought, 1963). Pero estas técnicas se mostraron insuficientes para resolver ciertos problemas, es necesario aplicar el conocimiento sobre la aplicación más que el conocimiento general. Esto lleva a buscar medios para transferir al ordenador el conocimiento de los expertos en distintos dominios de aplicación, dando lugar el desarrollo de los Sistemas Basados en Conocimiento o Sistemas Expertos, PROSPECTOR (Duda & Reboh, 1984) y (Hart, Duda, & Einaudi, 1978), MYCIN (Shortliffe, 1976).

InCo Era II Dominada por el proceso (1979-1989): Debido a la falta de método, la transferencia de los SBC al campo comercial fue un fracaso en la mayoría de los casos. Aparecían problemas de estimación del tiempo de desarrollo, no se cumplían las expectativas iniciales sobre lo que debería de hacer el producto y, sobre todo, el mantenimiento y verificación de dichos sistemas era una tarea difícil y costosa. La construcción de un SBC se plantea como un proceso de transferencia del conocimiento humano a una base de conocimiento implementada en un ordenador (Studer, Benjamins, & Fensel, Knowledge engineering: Principles and methods, 1998). Así de la misma forma que la crisis del software dio como resultado el establecimiento de la InSo como una disciplina, la situación que se produjo con el desarrollo de los primeros SBC dejó claro la necesidad de establecer planteamientos más metodológicos y con un mejor análisis del proceso (Palma, Paniagua, Martín, & Marín, 2000) y (Buchanan, y otros, 1983). Por otra parte el amplio espectro de aplicación de las técnicas de inteligencia artificial dio lugar a una especialización y diversificación de la

disciplina, minería de datos, visión artificial o el reconocimiento de formas no son más que algunos ejemplos de ello.

InCo Era III. Dominada por la complejidad (1990, 1998) A principios de los noventa aparecen las primeras metodologías que se replantean la InCo desde la perspectiva de la transferencia hacía la perspectiva del modelado, como un intento de solucionar el problema del cuello de botella que suponía la fase de adquisición de conocimiento. Se aplica la idea del nivel de conocimiento propuesta por Nevell como un nivel situado por encima del nivel simbólico y que constituye el nivel de abstracción adecuado para la representación del conocimiento independientemente de su implementación (Newell, The knowledge level, 1982).

Al plantear el proceso de desarrollo de un SBC como un proceso de modelado y la identificación de modelos genéricos de razonamiento, surge la necesidad de plantear metodologías que ayuden en el desarrollo de estos modelos. De estas inquietudes aparecen una serie de trabajos que desembocan en las llamadas metodologías de segunda generación, las más utilizadas son: CommonKADS (Comprensive Methodology for KBS Development) (Schreiber, et al., 1999), MIKE (Model-based and Incremental Knowledge Engineering) (Angele, Fensel, Landes, & Studer, 1998) y PROTÉGÉ-II (Eriksson, Shahar, Tu, Puerta, & Musen, 1995) y suponen el primer intento de ofrecer metodologías completas con una solución al ciclo de vida del SBC. Por otra parte, la necesidad de mejorar la productividad, al igual que ocurrió en su momento en la InSo da lugar a la potenciación de la reutilización de los componentes de conocimiento, de forma similar a la reutilización de clases y objetos en el de desarrollo orientado a objetos.

InCo Era IV. Dominada por las comunicaciones (1999, 2006) El nuevo concepto de software distribuido también se extiende hasta los SBC y también se extiende su campo de aplicación abordando dominios más complejos. La expansión de la Web lleva a la InCo hasta el reto de la recuperar, extraer, publicar y compartir del conocimiento. Aparecen las ontologías y la Web semántica [7] (Gómez, Corcho, & Fernández, 2004) y (Berners, Hendler, & Lassila, 2001), entendidas estas como una red inteligente donde las máquinas comprenderán los

[7] http://www.w3.org/2001/sw/

datos y podrán usarlos para extraer conclusiones. En esta era, dominada por las comunicaciones, la disponibilidad de grandes cantidades de datos acerca de áreas de aplicación concretas dio lugar al aprendizaje automático y la minería de datos (machine learning and data mining) cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. Se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. (Mitchell, 1997) y (Han, Kamber, & Pei, 2006).

InCo Era V Dominada por la productividad (2007- 2010) Un aspecto importante en esta era es la expansión de las técnicas de inteligencia artificial hasta hacerlas comercialmente viables en la generación de productos de consumo como los robots mascota interactivos ("smart toys"), o bien su aplicación a dominios concretos donde hasta el momento no había entrado, como por ejemplo la propia InSo con la Search Based Software engineering (SBSE) (Harman & Jones, Search-based software engineering, 2001) y (Harman, The Current State and Future of Search Based Software Engineering, 2007)

InSCo ERA I. DOMINADA POR LA UNIFICACIÓN (2010, -----) El campo de aplicación de soluciones software cada vez es más amplio y aborda sistemas y dominios cada vez más complejos y comerciales. Es en estos casos donde aparece la necesidad de sistemas híbridos y la solución software necesita coordinar servicios que tratan y gestionan información y la utilización de conocimiento en un solo producto. El desarrollo este tipo de software debe coordinar los procesos aplicados por la InCo y la InSo, para que de esta manera los subproductos generados por ambos enfoques de desarrollo se puedan combinar, ofreciendo a los usuarios finales una única visión del producto software resultante (Cañadas, Orellana, Águila, Palma, & Tunez, 2009) y (Águila I. M., Cañadas, Palma, & Túnez, 2006). Por tanto, tal como se justificó en la sección 1.1 "Ingeniería del software e ingeniería del conocimiento", los caminos de InCo e InSo debe fusionarse para definir propuestas unificadas que probablemente evolucionen hacia una nueva disciplina InSCo.

2.2 Metodologías InSo vs Metodologías InCo

Existen numerosas metodologías ligadas tanto a la InSo como a la InCo. Las que están vigentes en la actualidad no son más que la evolución y/o la unificación de las que se han definido y aplicado en eras precedentes. Por esta razón, y puesto que este trabajo se centra en la definición de un modelo de proceso integrador entre InCo e InSo, es conveniente analizar cómo se ha desarrollado y cuál es el estado actual del mapa metodológico para el que la propuesta InSCo supondrá un paso más. Para la descripción de estas metodologías se ha optado por clasificarlas en grandes grupos, que se corresponden con los apartados de esta sección. Los dos primeros correspondientes a la InSo agrupan las metodologías estructuradas, tanto orientadas a procesos como orientadas a datos y las metodologías orientadas a objetos. Los dos siguientes se corresponden con las metodologías de primera y segunda generación de la InCo.

La tabla 2.2 resume las características de las metodologías más relevantes de la InSo y la InCo, algunas de ellas ya obsoletas, pero de clara influencia en mapa metodológico actual.

Tabla 2.2 Descripción general de las metodologías.

Artefactos	**Notación**	Etapas / Flujos de trabajo
Métodos estructurados		
Análisis y diseño estructurado (SAD) Describe la evolución de los datos a través del sistema modelando el flujo de los datos		
Especificación de los requisitos Modelo de análisis Modelo de diseño	Diagramas de flujo de datos Diccionario de datos Miniespecificaciones Diagramas de estructura	Especificación Diseño Implementación y prueba Mantenimiento
SAD para sistemas en tiempo real Extensión para representar situaciones en tiempo real las más conocidas son la de WARD y MELLOR y la de HARTLEY y PIRBHAI		
Especificación de los requisitos Modelo de análisis Modelo de diseño Modelo de comportamiento	Diagramas de flujo de datos Diagramas de control Especificación de control Diccionario de datos Lenguaje estructurado Diagramas de estructura	Análisis Diseño Implementación y prueba Mantenimiento

Métodos orientados a datos		
Diseño estructura do de datos DSED Se basa en el modelado de la información y su estructura		
Modelo de datos Funciones de la aplicación Resultados de la aplicación	Diagramas de estructura de datos Diagrama de línea de ensamblaje Diagrama de entidades	Definición del contexto de la aplicación Funciones de aplicación Resultados de la aplicación
Método de Jackson (JSP) Define la lógica del programa y los datos empleando árboles de Jackson.		
Estructura de programa Estructura de datos	Diagramas de estructura/entidad Texto estructurado	Estructuración Codificación
Métodos orientados a objetos		
Método de Jackson (JSD) Se ordenan en el tiempo las acciones a realizar sobre las entidades relevantes al problema.		
Modelo inicial Modelo funcional	Diagramas de estructura/entidad Texto estructurado	Paso de acciones y entidades Paso de estructura entidad Paso de modelo inicial Paso funcional Paso de temporización Paso de implementación
Metodología OMT Centrada en diseño de clases y aplicaciones software		
Modelo lógico Modelo físico Modelo estático Modelo dinámico	Diagrama de clases y objetos Diagrama de módulos Diagrama de estados Diagrama de procesos Diagrama de interacción	Conceptualización Análisis Diseño Evolución
UML y RUP Se centra en la realización de descripción mediante un lenguaje de la aplicación.		
Modelo de casos de uso Modelo de análisis Modelo de diseño Modelo de despliegue Modelo de implementación Modelo de prueba	Diagramas de clases, casos de uso, objetos, secuencia y colaboración Diagramas de estados y de componentes Diagramas de actividades y despliegue	Fases: Inicio Elaboración Construcción Transición Flujos de trabajo: Requisitos Análisis Diseño Implementación Prueba

Métodos para de primera generación para el desarrollo de SBC		
Buchanan Primeros intentos de formalización metodológica en la construcción de sistemas expertos		
Bases de conocimiento Métodos de razonamiento	Reglas Marcos	Identificación Conceptualización Formalización Implementación Validación
IDEAL		
Plan Modelo de casos de uso Modelo conceptual estático Modelo de proceso y control Modelo formal Modelo computacional		Identificación de la aplicación Desarrollo de prototipos construcción del sistema Mantenimiento perfectivo y transferencia tecnológica.
Métodos de segunda generación para desarrollo de SBC		
CommonKADS Desarrolla un SBC construyendo un conjunto de modelos de ingeniería que se definen tanto para la organización como para la aplicación.		
Modelo de organización Modelo de tareas Modelo de agentes o actores Modelo de conocimiento Modelo de comunicación Modelo de diseño	Documentos de los modelos. Diagramas de inferencias Diagramas de tareas métodos Diagramas de clases Diagramas de estados Diagramas de casos de uso. Plantillas o formularios.	Revisión Estudio de riesgos Monitorización Revisión
MIKE Cubre todos los pasos del desarrollo. Integra de técnicas de especificación formales y semiformales, prototipado y modelos de ciclo de vida dentro de un marco de ingeniería.		
Modelo de elicitación Modelo KARL Modelo de diseño	KARL Design-KARL	Adquisición Interpretación Formalización Diseño Implementación Evaluación
Protégé II Permite desarrollar herramientas para la adquisición del conocimiento, el desarrollador puede especificar tareas y seleccionar métodos de resolución de la librería.		
Modelo de conocimiento	Herramienta de adquisición de conocimiento Base de conocimiento	---

2.2.1 Metodologías estructuradas

Las primeras técnicas que podían llamarse ya metodologías como SREM (Alford, 1977) y SADT (Ross & Schoman, 1977) extendieron los conceptos de modularización y ocultamiento de información, aplicados en la programación estructurada (Dijkstra, 1976) y (Jackson, Principles Of Program Design, 1975), del diseño a la especificación. Tras estos trabajos iniciales, se propuso el Análisis Estructurado y el Diseño Estructurado (SAD) para sustentar las tareas de análisis de los requisitos y diseño del software, que distintos autores pusieron en práctica desde distintas perspectivas.

Inicialmente su objetivo fue simplemente el de representar los datos y los procesos en formato gráfico, a partir del cual se generaron dos líneas de trabajo: una en la que lo primordial era representar los datos y, de forma complementaria, los procesos, y la otra a la inversa, es decir, los elementos primarios son los procesos y de forma colateral se representan los datos. Este doble enfoque generó dos tipos de metodologías para el desarrollo de software diferenciadas, las orientadas a procesos y las orientadas a datos (Pressman, Ingeniería del Software - Un Enfoque Práctico, 1993) y (Piattini, Calvo, Joaquín, & Férnandez, 1996),

2.2.1.1 Metodologías estructuradas orientadas a datos

Esta familia de metodologías aparece ligada a la actividad de capturar el domino de un problema mediante un conjunto de datos y de generar un modelo para esos datos y sus relaciones. Una de las primeras herramientas para describir la información fueron los diagramas entidad relación propuestos por Chen, que permiten representar tanto los datos como las relaciones entre los datos (Chen P. P., 1991) y (Chen P. P., 1983). Estos diagramas son el origen del modelado conceptual y el diseño de bases de datos, que han evolucionado hacia versiones mejoradas de los diagramas y elementos de modelado más expresivos (Batini, Ceri, & Navathe, 1991) y (Olivé Ramon, 2002).

A partir de estas técnicas de modelado de datos aparecen las primeras metodologías, que centran su trabajo en la descripción de las estructuras de datos y de programas mediante tres constructores: concatenación, selección e iteración. Las más representativas son las metodologías de Warnier-Orr (Orr, 1977) y JSP (Jackson Structured Programming), (Jackson, Principles Of Program Design,

1975) y (Storer & Jackson, 1987). Esta última se centra sólo en el diseño del software, siendo su objetivo la definición los llamados árboles de Jackson o diagramas de estructura-entidad que definen la lógica del programa y los datos mediante los constructores de concatenación, selección o iteración.

La metodología de Warnier-Orr es más completa y comienza desde el análisis. La información y los procesos o tareas se modelan con la misma notación, los diagramas de Warnier, que incluyen los tres constructores básicos antes citados. El desarrollo de software arranca de la definición del contexto de la aplicación mediante el llamado diagrama de entidades, donde cada entidad representa un lugar, tarea o almacén que genera información necesaria para alguna otra entidad. A partir de este estudio del contexto se definen los límites del software y se construye el diagrama de Warnier, que representa la línea de ensamblaje que en términos más actuales representa la arquitectura del software a codificar. Por último también se construyen los diagramas de Warnier que describen las estructuras de los datos.

2.2.1.2 Metodologías estructuradas orientadas a procesos

Este tipo de metodologías se centran en definir cómo los datos de entrada se trasforman hasta obtener los datos de salida, es decir se representan las funciones o procesos. Al igual que ha pasado después con las metodologías orientadas a objetos, éstas han evolucionado pasando por tres grandes etapas: a) una de origen donde se fundamenta la metodología, b) otra de expansión o desarrollo donde los diversos autores centran su atención en un tema y desarrollan la metodología en esa dimensión y c) la última la de unificación, donde se hacen concordar muchas de estas propuestas para afianzar estas metodologías en el mercado de las metodologías efectivas y productivas en el desarrollo industrial.

Los trabajos de DeMarco, Gane y Sarson (Demarco, 1979), (Gane & Trish, 1982) y (Yourdon & Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 1979), convirtieron los meros mecanismos de representación del proceso en las primeras metodologías estructuradas, al completo añadiendo el diccionario de datos y las miniespecificaciones. No obstante, y debido a la propia naturaleza de las técnicas estructuradas se diferencia claramente entre dos fases para el desarrollo de software, el análisis (qué) y el diseño (cómo) y aunque el objetivo ideal es que la

transición entre ambas sea transparente es decir un proceso sin costuras[8], la realidad es muy distinta.

El *Análisis Estructurado* especifica el software construir mediante la identificación de las funciones del sistema y el flujo de datos entre ellas. En las primeras versiones del origen de esta metodología se generan tres artefactos como resultado de las tareas de análisis: Diagramas de flujo de datos (DFD), diccionario de datos y miniespecificaciones. Los DFD, que no son más que un conjunto de grafos dirigidos y jerarquizados cuyos nodos representan las interfaces con el entorno externo (representado por las llamadas entidades externas), las funciones que trasforman datos (llamados proceso o burbujas) o los almacenes de datos que guardan información y cuyos arcos representan flujo de información entre los nodos. Cada DFD y cada elemento del DFD tiene un nombre que debe ser descrito con una entrada en el diccionario de datos. Los procesos se descomponen repetidamente en subprocesos, cuando los subprocesos resultantes son lo suficientemente claros (funciones primitivas), se describen mediante miniespecificaciones, que no son más que una forma de lenguaje natural estructurado basado en la concatenación, secuencia e iteración.

Este enfoque se ha ido desarrollando y expandiéndose generando distintas versiones o extensiones al método básico original, y generalizando el principio del modelado. Una de las aportaciones más conocidas es la que incluye los diagramas de transición de estados (DTE) para describir los sistemas en tiempo real, es decir, aplicaciones software que son estrechamente dependientes del tiempo y procesan más información orientada al control que a los datos. Esta técnica fue propuesta independientemente por dos grupos de trabajo liderados respectivamente por Ward (Ward & Mellor, 1985) y Hartley (Hatley & Pirbhai, 1988). En estas extensiones a la metodología estructurada básica, los desarrolladores describen además del flujo de datos, el flujo de señales de control o flujos de control que pueden activar o desactivar procesos. Los DTE permiten la secuenciación de esos flujos de control. En el método de Ward se describen los flujos de control sobre el DFD, incluyendo nuevos símbolos. En el de Harley los flujos de control se especifican de forma separada empleando el diagrama de

[8] El término en ingles es "seamless process"

flujo de control, donde se muestran los mismos procesos que DFD pero sólo con los flujos de control.

El otro aspecto de la metodología estructurada es el *Diseño Estructurado*, que aunque se busca que sea la continuación directa de los mecanismos de análisis, la distancia real entre análisis y diseño estructurado es mayor de la deseada. Los métodos de diseño y de análisis se definieron de forma separada y, aunque la tendencia es la búsqueda de la convergencia o unificación, aun existe demasiada diferencia entre el software a construir y el software a implementar, es decir, entre el resultado del análisis y el del diseño.

El objetivo de la etapa de diseño estructurado es la definición de la arquitectura del software, estableciendo una correspondencia sistemática entre los procesos del DFD y los módulos del programa, para así obtener un diseño altamente modular. El artefacto clave generado durante el diseño son los diagramas de estructura que representan el conjunto de módulos del programa, las relaciones de llamada-retorno y las interfaces entre los módulos. Para intentar disminuir el salto de la especificación o análisis al diseño, y como intento unificador aparece la metodología estructurada moderna (Yourdon, Modern Structured Analysis, 1988) que emplea distintas técnicas y notación para el modelado. De esta forma, se responde al planteamiento global de que el software puede ser visto como un conjunto de procesos o funciones que operan sobre un conjunto de datos modulados por los estímulos que el sistema software recibe del entorno externo. Así tal como se muestra en la figura 2.1 (a), un sistema software debe ser modelado sobre estas tres dimensiones: datos, procesos y sucesos. A pesar de este talante integrador, la metodología de diseño estructurado moderno comienza la especificación del problema a resolver sólo desde la dimensión funcional.

Esta metodología se basa en una serie de notaciones o mecanismos de modelado bastante conocidos, comentados anteriormente en este capítulo, en los que cada notación representa una dimensión de modelado. El desarrollo se aborda en dos fases: la primera es el llamado diseño de alto nivel, o lo que en otras versiones de la metodología estructurada llama análisis, cuyo objetivo es obtener el modelo esencial del sistema; la segunda fase es el diseño de bajo nivel, donde se deriva el modelo de diseño a partir de los resultados de la etapa anterior.

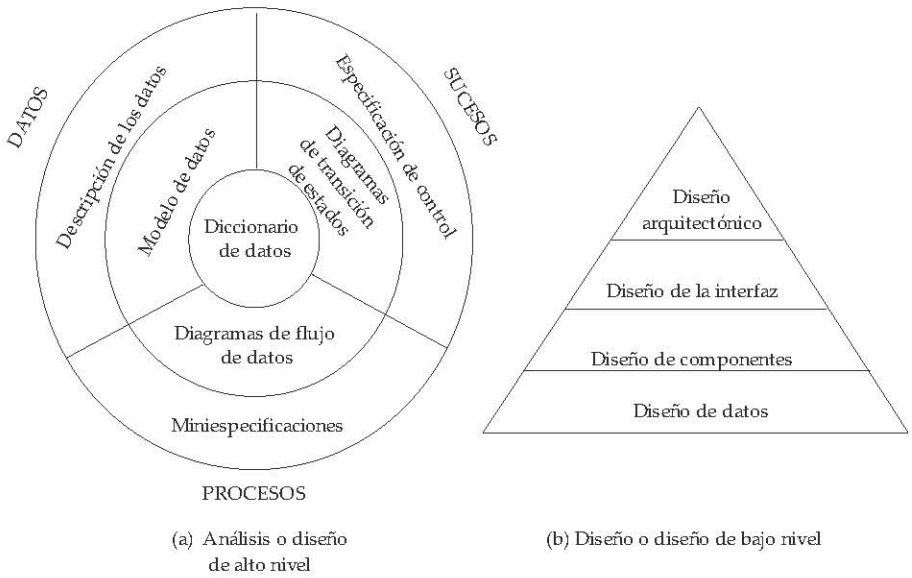


Figura 2.1 Metodología estructurada.

El modelo esencial está compuesto por dos modelos: el modelo ambiental más general donde se recoge el propósito del sistema, compuesto por: el DFD de contexto y la lista de acontecimientos; y el más detallado modelo de comportamiento que recoge los DFD detallados, el diagrama entidad relación y DTE así como las entradas al diccionario de datos y los demás elementos mostrados en la figura 2.1 (a). El enfoque no es top-down, sino middle-up, la construcción de los DFD está guiada por acontecimientos, es decir, un sistema es un productor de salidas en respuesta a acontecimientos o estímulos del ambiente, característica que se hereda posteriormente en los desarrollos guiados por casos de uso.

El diseño de bajo nivel implica las cuatro grandes áreas del diseño estructurado, tal y como se muestra en la figura 2.1 (b): arquitectura, datos, interfaz y componentes. En el diseño arquitectónico o definición de la arquitectura del software se definen las relaciones entre los módulos del software. El diseño de datos conlleva la definición las estructuras de datos que se necesitarán para la implementación del software y, por último, el diseño de la interfaz describe la manera de comunicarse los módulos del software entre sí, con los sistemas que interoperan dentro de él y con las personas que lo utilizan. El diseño a nivel de componentes supone una descripción procedimental detallada

de los componentes o módulos software. El resultado de estas tareas de diseño de bajo nivel es el modelo de diseño que, utilizando las herramientas adecuadas, se traduce a código ejecutable de forma automática.

El talón de Aquiles de las técnicas estructuradas es que hay una desconexión básica entre el modelo de análisis (o modelo esencial) y el modelo de diseño. Si no se salva este abismo, el sistema concebido y el construido acaban divergiendo con el paso del tiempo. Por el contrario, al tratar la construcción de software de forma organizada y sistemática, permite el desarrollo y utilización de herramientas CASE, existen bastantes de estas herramientas que ayudan en las tareas de desarrollo, y en general, los proyectos de riesgo controlado que aplican esta metodología tienen el éxito asegurado.

2.2.2 Metodologías orientadas a objetos

Este grupo de metodologías responden al paradigma de la programación orientada a objetos (POO), que vino a representar en los noventa lo que la programación estructurada fue para los setenta (Budd, 1994). La POO redefine el concepto de software como un conjunto de objetos que interactúan entre sí, dotados de un estado y funcionalidad propia. El objetivo de esta tecnología es obtener software más consistente, robusto y reutilizable.

La metodología o desarrollo de sistemas de Jackson (JDS) se centra en la definición de los llamados diagramas de Jackson, y se puede considerar como el origen de las actuales metodologías basadas en objetos (Jackson, System Development, 1983) y (Sutcliffe A. , 1988). Se comienza con un análisis de las entidades construyendo un modelo de la realidad concerniente al sistema. Se buscan y se ordenan en el tiempo las operaciones o acciones a realizar sobre las entidades relevantes al problema. Después se fija el modelo inicial con los diagramas de especificación de sistemas y se elabora una especificación, que se convierte en software durante la etapa de implementación. El concepto de entidad es el precursor del concepto de clase y es en esta metodología donde aparece por primera vez a más alto nivel, no sólo para las tareas de programación.

Tal como se indicó al describir la evolución de la InSo, a partir de los 90 el modelado es la parte central de todas las actividades que conducen al desarrollo del software, desde la especificación al mantenimiento. El punto de partida para

construir software aplicando la orientación a objetos es el modelado del dominio del problema y, de forma gradual, generar los modelos pertenecientes al dominio de la solución, que definirán el código orientado a objetos. Para este proceso de modelado surgieron un conjunto de métodos cuyo número se incremento en torno a 400% de 1989 a 1994. Después de esta primera avalancha que supuso la expansión de las metodologías orientadas a objetos, comenzaron a aparecer nuevas generaciones de métodos que presentan planteamientos alternativos y en ocasiones enfrentados. Entre ellos destacaron de manera clara unos pocos, en especial el método de Booch (Booch, Object-Oriented Analysis and Design with Application, 1993), el método OOSE (Object Oriented Software Engineering) de Jacobson (Jacobson, Christerson, Jonsson, & Övergaard, 1992) y el método OMT (Objetc Modeling Technique) y OMT-2 de Rumbaugh (Rumbaugh, Blaha, Lorensen, Eddy, & Premerlani, 1990). Otros métodos importantes fueron Fusion (Coleman, Arnold, Bodoff, Dollin, & Gilchrist, 1993), Shlaer-Mellor (Mellor & Shlaer, 1991) y Coad (Coad, North, & Mayfield, Object Models: Strategies, Patterns, and Applications (2nd Edition), 1996). Cada una de ellas era una metodología completa, aunque todas tenía sus puntos fuertes y sus debilidades, podemos decir que, el método de Booch era particularmente expresivo durante las fases de diseño y construcción de proyectos, OOSE proporcionaba un soporte excelente para los casos de uso como forma de dirigir la captura de requisitos, el análisis y diseño de alto nivel, y OMT-2 era principalmente útil para el análisis y para los sistemas de información con gran cantidad de datos.

Cada uno de estos métodos fue evolucionando independientemente hacia los otros, por tanto era mejor hacer continuar esa evolución de forma conjunta en vez de hacerlo por separado. Esto llevó a Rational a plantear proyectos de unificación con los creadores de los principales métodos que diseñaron un lenguaje unificado de modelado (Unified Modeling Language, UML) (Booch, Rumbaugh, & Jacobson, The Unified Modeling Language User Guide, 1998), (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual, 1999). Eliminando la posibilidad de cualquier diferencia gratuita e innecesaria que confundiría a los usuarios, además de esta forma se proporcionaría estabilidad al mercado orientado a objetos.

UML es un lenguaje gráfico estándar para escribir modelos de software que puede utilizarse para visualizar, especificar, construir y documentar los

elementos de un sistema que involucra una gran cantidad de software. UML proporciona una forma estándar de escribir los modelos del software.

Sin embargo, un único modelo no es suficiente y cualquier sistema no trivial se aborda mejor a través de un pequeño conjunto de modelos casi independientes, es decir, que se pueden estudiar por separado. Para comprender la arquitectura de los sistemas se necesitan varias vistas complementarias y entrelazadas: una vista de los casos de uso que muestre los requisitos del sistema, una vista de diseño que capture el vocabulario del espacio del problema y del espacio de la solución, una vista de procesos que modele la distribución de los procesos e hilos del sistema y una vista de despliegue que se centre en cuestiones de la ingeniería del sistema. Cada una de estas vistas puede tener aspectos tanto estructurales o estáticos como de comportamiento o dinámicos, cada uno representado por uno o varios modelos (Kruchten, 1995).

Según la naturaleza del propio sistema a modelar, algunas de las vistas y, por tanto los modelos que lleve asociados, pueden ser más importantes que otras. Esto hace que UML sea apropiado para modelar desde sistemas de información en empresas hasta aplicaciones distribuidas basadas en la Web, incluso para sistemas empotrados de tiempo real muy exigentes.

Los elementos de UML se pueden agrupar en tres categorías. La primera la forman los elementos estructurales (clases, casos de uso, componentes, nodos) y muestran las representaciones estáticas o conceptuales de un modelo. La segunda los de comportamiento (mensajes, máquina de estados, eventos, actividades), representan las partes dinámicas. Por último, los elementos de agrupación (paquetes, notas) se utilizan para describir, hacer observaciones. Estos elementos se agrupan en diagramas y cada uno representa parte de una vista del sistema. Los Diagramas de Casos de Uso, desarrollados por los analistas y expertos del dominio, se construyen en las primeras etapas del desarrollo para especificar el contexto de sistema, capturar los requisitos, validar la arquitectura, dirigir la implementación y generar los casos de prueba. Los Diagramas de Clases se utilizan para denominar y modelar los conceptos del sistema y para especificar las colaboraciones y los esquemas lógicos de las bases de datos. Los Diagramas de Colaboración, se emplean para modelar el flujo de control, ilustrando la coordinación entre la estructura de un objeto y su control. El objetivo de los Diagramas de Secuencia, su objetivo es modelar el flujo de control e ilustrar los escenarios. Los Diagramas de Estados modelan el ciclo de vida de los objetos y

modelan los objetos reactivos. Los Diagramas de Actividad, modelan los flujos de negocio y las operaciones. Los Diagramas de Componentes, desarrollado por arquitectos y programadores, se construyen como parte de la especificación de arquitectura y sirven para organizar el código fuente, construir una versión ejecutable y especificar una base de datos física. Los Diagramas de Implantación o Diagramas de Despliegue, desarrollado por arquitectos, ingenieros de red e ingenieros del sistema, se construye como parte de la especificación de la arquitectura para especificar la distribución de componentes e identificar los cuellos de botella.

Actualmente, el enfoque orientado a objetos forma parte de la tendencia principal para el desarrollo de software, simplemente porque ha demostrado ser válido en la construcción de sistemas en toda clase de dominios de aplicación, abarcando todo el abanico de tamaños y complejidades.

UML es sólo un lenguaje, y por lo tanto sólo una notación aplicable a algunas metodologías para el desarrollo de software. UML es independiente del proceso, aunque para utilizarlo óptimamente se debería utilizar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999). Rational también propone un modelo de proceso. El proceso unificado de Rational (Rational Unified Process) (RUP) que presenta las características apropiadas para que al ser combinado con la notación UML y permite obtener los mejores resultados en la construcción de software.

RUP puede describirse en dos dimensiones. El eje horizontal representa tiempo y muestra el aspecto dinámico del proceso, expresado en términos de ciclos, fases, iteraciones, y metas. El eje vertical representa el aspecto estático del proceso; como está descrito en términos de actividades, artefactos, trabajadores y flujos de trabajo, tal como se muestra en la figura 2.2.

La dimensión estática es la más conocida y pretende separar el factor Cuándo del factor Qué. Las actividades fundamentales definidas en el eje vertical indica qué hay que hacer, los artefactos y técnicas se tienen que utilizar, pero no se fija a priori la secuenciación o el número de revisiones a realizar de cada artefacto generado en la ejecución del proceso. Esto último se fija con el eje horizontal, en el que RUP propone la separación en cuatro fases, pudiendo presentar cada una de ellas más de una iteración, especialmente las dos intermedias. En este contexto se entiende por iteración un ciclo de desarrollo

completo que se da como resultado una entrega de producto ejecutable (interna o externa).

En la fase de Inicio se define el alcance del proyecto, estableciendo los casos de uso del negocio para un nuevo sistema y generando como resultado una visión general de los requisitos del proyecto, un modelo inicial de los casos de uso y del dominio. En la fase de Elaboración se analiza el dominio del problema, y se define la arquitectura software del sistema. En esta fase y en el ámbito de la gestión del proyecto de desarrollo, se define un plan detallado, que indica cómo continuara el proceso de desarrollo hasta el final. Respecto a la gestión se desarrolla un plan comprensivo mostrando cómo se completará el proyecto. Durante la fase de Construcción se desarrolla incrementalmente producto de software completo, el cual estará listo para ser transferido al usuario. Por último, en la fase de Transición se presenta el producto a la comunidad de usuarios.

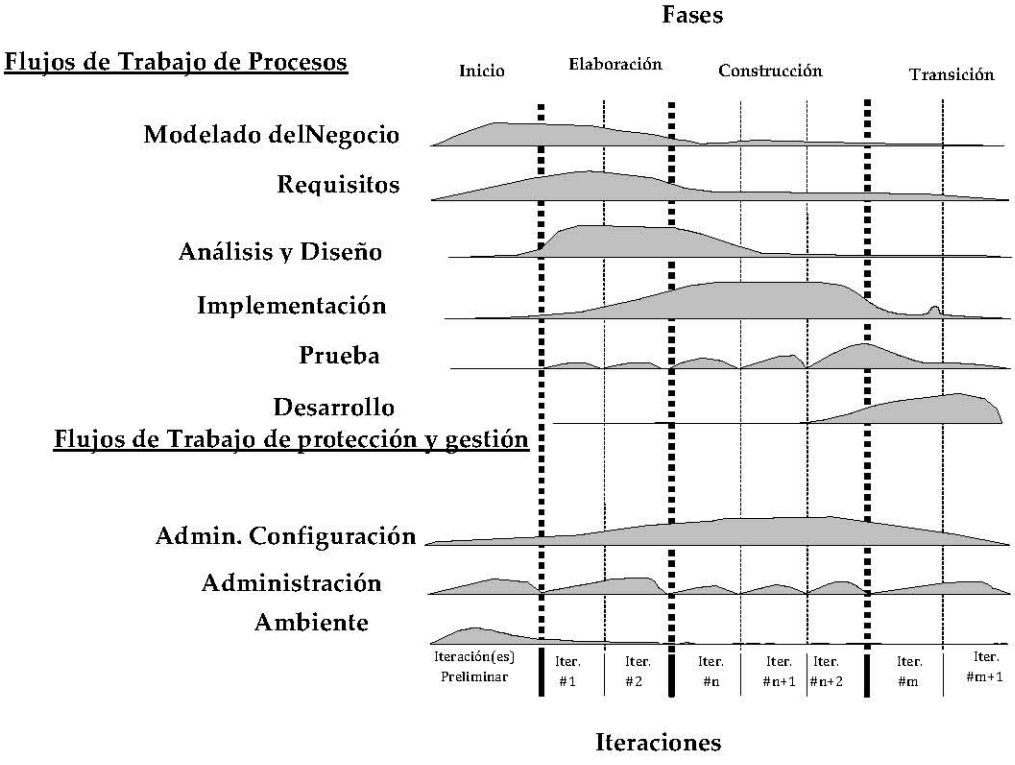


Figura 2.2 Proceso Unificado de Rational

2.2.3 Metodologías de primera generación de SBC

La necesidad de aplicar metodologías para el desarrollo de un SBC aparece con la InCo durante la segunda era, InCo II Dominada por el proceso (1979-1989) descrita en la sección 2.1 "Recorrido histórico de InSo e InCo". En su origen, las primeras metodologías presentan el desarrollo de un SBC como un proceso de adquisición de conocimiento, entendido éste como la transferencia y transformación de la experiencia en la resolución de problemas desde alguna fuente de conocimiento o experto a un programa de ordenador siendo necesaria la intervención de un ingeniero del conocimiento que sirva de intermediario (Buchanan, y otros, 1983) y (Hayes-Roth, 1983).

La aportación más importante del trabajo de Buchanan fue la identificación del cuello de botella que aparece como consecuencia de un problema de comunicación con el experto. Para ayudar al ingeniero del conocimiento a solventar este problema, se propone un ciclo de vida para el proceso de adquisición de conocimiento que abarca desde la concepción del sistema hasta su madurez.

La metodología aplica el ciclo de vida de en cascada mostrado en la figura 2.3 y consta de varias etapas. En la etapa de *identificación*, donde se caracterizan los aspectos más importantes del problema. Etapa de *conceptualización* se hacen explícitos los conceptos y relaciones construyendo un diagrama que se utiliza como base conceptual para la elaboración de un prototipo. Etapa de *formalización* consiste en llevar los conceptos claves, los problemas y las características de los flujos de información a una representación más formal. Etapa de *implementación* implica la traducción del resultado de la formalización a otro marco de representación que tiene que estar asociado con la herramienta utilizada. Por último, en la Etapa de *validación* se evalúa el prototipo obtenido y la representación elegida para implementarlo.

La importancia de esta metodología radica en que es el primer intento que permite abordar el desarrollo de un SBC desde un punto de vista comercial. Durante los años siguientes se promueven numerosas modificaciones y mejoras generando metodologías que plantean el desarrollo de SBC con una aproximación más cercana a las técnicas de InSo, definiendo y organizando las diferentes actividades que conducen a la implementación y uso rutinario de los SBC. Entre estas metodologías que podemos llamar de primera generación

destacamos KLIC (Knowledge Based System Life Cycle) (Guida & Tasso, 1994) e IDEAL (Gómez, Juristo, Montes, & Pazos, 1997) y (Alonso, Antonio, Gonzalez, Fuertes, & Martinez, 1998).

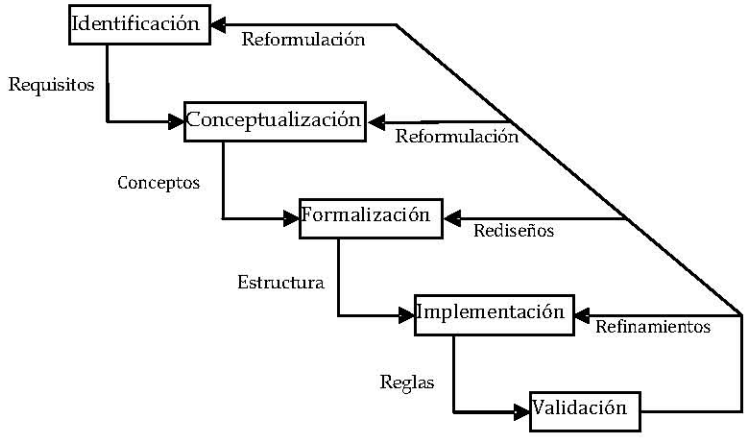


Figura 2.3 Ciclo de vida de Buchanan.

La primera incluye la necesidad de realizar un estudio de viabilidad y análisis de alternativas previos al desarrollo propiamente dicho, además define las fases y los artefactos generados en el ciclo de vida. Básicamente el nivel más alto de abstracción el ciclo de vida está compuesto por: el *Análisis de posibilidades, Análisis de viabilidad, Construcción de un demostrador, Desarrollo del prototipo, Implementación, Instalación y entrega del producto final, Mantenimiento y extensión.*

La metodología IDEAL recibe su nombre de las iniciales de cada una de las fases que conlleva: Identificación de la tarea, Desarrollo de los prototipos, Ejecución de la construcción del sistema integrado, Actuación para conseguir el mantenimiento perfectivo, Adecuada transferencia tecnológica. IDEAL incorpora un ciclo de vida en espiral cónico en tres dimensiones (Alonso F. , Juristo, Mate, & Pazos, 1996) y (Alonso, Fuertes, Martinez, & Montes, 2000). Este modelo solventa el problema del modelo en espiral plano, que no tiene en cuenta el mantenimiento perfectivo, es decir, no recoge la necesidad de la incorporación sistemática de nuevos conocimientos que se producen por el propio uso del SBC.

Esta familia de metodologías fue utilizada con relativo éxito hasta principios de la década de los noventa, fecha en la que se replantea la InCo desde la perspectiva de la transferencia hacía la perspectiva del modelado. La aproximación basada en la transferencia del conocimiento sólo es posible para

pequeños sistemas, y falla a la hora de producir sistemas grandes con aspiraciones comerciales, fiables y con un mantenimiento asequible.

2.2.4 Metodologías de segunda generación de SBC

Las metodologías de segunda generación vienen a permitir realizar el análisis del sistema en el nivel de conocimiento, ofreciendo la posibilidad de especificar el problema a diferentes niveles de granularidad, con lo cual se potencia la idea de la reutilización de componentes de conocimiento. Por otro lado, ofrecen un ciclo de vida completo para el proceso de desarrollo, proporcionado pautas a seguir desde el análisis hasta la implementación final del sistema, al estilo de las propuestas de IDEAL.

El origen de estas nuevas metodologías de desarrollo de SBC se produce el replanteamiento hacia el modelado de la InCo. Su origen está en las ideas propuestas en trabajos anteriores que están todavía muy lejos de ser considerados como metodologías; de entre ellos desatacamos la aportación de Chandrasekaran (Chandrasekaran, Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, 1986) que propone la existencia de tareas genéricas como componentes combinables que pueden reutilizarse para la construcción de diferentes SBC. Posteriormente, el mismo equipo de Chandrasekaran propone las estructuras de tareas para distinguir entre el concepto de tarea que se asocia con el tipo de problema a resolver, y el de Problem Solving Method que es la forma de llevar a cabo una tarea. (Chandrasekaran & Johnson, Generic tasks and task structures: history, critique and new directions, 1993). También hay que destacar el trabajo de McDermott (McDermot, 1988) que introdujo el concepto de rol, que permite aislar el modelado de conocimiento de los aspectos de implementación.

KADS (Knowledge Acquisition and Design Structuring) y Protégé son los primeros trabajos que resolvían el problema de la adquisición del conocimiento desde el enfoque del modelado y que ya se pueden considerar de segunda generación. A partir de estos proyectos se han desarrollado metodologías más completas de entre las que destacamos CommonKADS (Schreiber, et al., 1999), MIKE (Angele, Fensel, Landes, & Studer, 1998) y PROTÉGÉ-II, (Eriksson, Shahar, Tu, Puerta, & Musen, 1995). A continuación se describen brevemente todas ellas prestando especial atención a CommonKADS ya que como se justificará en la

sección 2.4 "Metodologías soporte" es una de las metodologías soporte para este trabajo de tesis.

PROTÉGÉ-II fue el resultado de un proyecto cuyo principal objetivo era resolver las limitaciones de las primeras versiones. Es mas una herramienta de adquisición del conocimiento que de desarrollo. PROTÉGÉ-II proporciona un entorno para InCo en el que el desarrollador puede especificar tareas, y puede seleccionar métodos de resolución de una librería de métodos reusables. Para permitir el modelado, se distingue entre *Tareas, Métodos y Mecanismos*. Las tareas representan un problema en el mundo real, localizado dentro de una organización (*tarea aplicación*), o una especificación abstracta de objetivos independientes del dominio que tienen que ser alcanzados (*clases de tareas, o tarea*). Un método es un modelo independiente del dominio, que indica cómo se resuelve el problema especificado por la tarea. Los métodos pueden delegar problemas a subtareas que, a su vez, pueden ser resueltas por otros métodos, formando un árbol de tareas-métodos-subtareas. Por último, los mecanismos se pueden considerar como métodos primitivos que no pueden ser descompuestos en otras subtareas. Por lo tanto, pueden ser considerados como cajas negras que no admiten descomposición.

La descomposición termina al alcanzar los mecanismos. Esta estrategia de descomposición ayuda a la reutilización de componentes de dos formas: por un lado, los mecanismos, al desarrollar funciones muy definidas, pueden ser aplicables a varios dominios; y por otro, la existencia de métodos descomponibles hace más flexible a la hora de la reutilización, ya que permiten la utilización de métodos y mecanismos alternativos para la resolución de sus subtareas.

MIKE es una metodología para el desarrollo de SBC que aplica un ciclo de vida en espiral, aunando el prototipado con un proceso de desarrollo incremental y el paradigma del modelado. El proceso de desarrollo se plantea con fases cíclicas: *adquisición, interpretación, formalización, diseño, implementación* y *evaluación*.

La fase de *adquisición* empieza con la extracción de conocimiento, con la que se obtienen descripciones informales sobre el conocimiento del dominio y del proceso de resolución. Se genera el modelo de adquisición, donde la información se almacena en lenguaje natural. Durante la *interpretación* se redescriben las estructuras identificadas en la fase anterior en un lenguaje más restringido, es el llamado modelo de estructura, que recoge todos los aspectos funcionales y no

funcionales del dominio. Este modelo, junto con el de adquisición, son la base para establecer la comunicación entre el experto y el ingeniero del conocimiento.

En la fase de *formalización*, se crea un modelo más formal que el de estructura mediante la utilización de un lenguaje formal, KARL (Knowledge Acquisition Representation Language) (Fensel, 1995), que se denomina modelo KARL. La característica principal de este lenguaje es que combina la descripción conceptual de un SBC con una especificación formal y ejecutable. De esta forma, se pueden especificar formalmente, eliminado cualquier vaguedad e imprecisión, las inferencias detectadas en la fase anterior. En la fase de *diseño* se genera un nuevo modelo más detallado, el modelo del diseño, especificado en el lenguaje DesignKARL, que es una extensión de KARL a la que se le añade la capacidad de expresar algoritmos y estructuras de datos. Una vez obtenido el modelo del diseño, se pasa a la fase de *implementación*, en la que esté se materializa en la plataforma hardware y software elegida. La última fase del proceso de desarrollo termina con la *evaluación*, en la que se vuelve a requerir la intervención del experto.

MIKE es una potente metodología que integra formalismos de representación informales y formales con un desarrollo por prototipado. Pero como ya se ha indicado, la utilización de métodos formales conlleva alto consumo de recursos que desde un punto de vista comercial no son asumibles en la mayor parte de los proyectos de desarrollo de software.

CommonKADS es la metodología de desarrollo de SBC más extendida en Europa, evolución de KADS, cubre la gestión del proyecto, el análisis organizacional y los aspectos relativos al desarrollo. Esta metodología recoge ideas que han sido objetivo de InCo e InSo: modelado, gestión de riesgos y reusabilidad (Breuker & Velde, 1994).

El desarrollo de un SBC se plantea como la construcción de distintos modelos sobre el comportamiento en la resolución de problemas en un contexto organizacional concreto. Estos modelos forman el conjunto de modelos de CommonKADS, que como se muestra en la figura 2.4 está formado por seis modelos: *modelo de organización, modelo de tareas, modelo de experiencia, modelo de comunicación, modelo de agentes y modelo de diseño.* (Schreiber G. , Wielinga, Hoog, Akkermans, & Velde, 1994) y (Hoog, Martil, Wielinga, Taylor, Bright, & Velde, 1994)

El *modelo de organización* tiene por objetivo analizar las principales características de la organización para descubrir los problemas y las oportunidades de uso de los SBC, además de establecer su viabilidad y valorar su impacto en la organización. (Hoog R. D., Benus, Metselaar, Vogler, & Menezes, 1994) y (Hoog R. D., Benus, Vogler, & Metselaar, 1996).

El *modelo de tareas* muestra las tareas como partes relevantes del proceso de negocio. Analiza la organización global de las tareas, sus entradas, salidas, precondiciones, criterios de rendimiento, así como la competencia y recursos necesarios. (Duursma, Olsson, & Sundin, 1993)

El *modelo de agentes* muestra los agentes como ejecutores de tareas. Un agente puede ser una persona, un sistema software o cualquier cosa capaz de realizar una tarea. Se describen las características de los agentes, así como las conexiones entre agentes.(Waern & Gala, The CommonKADS agent model, 1993)

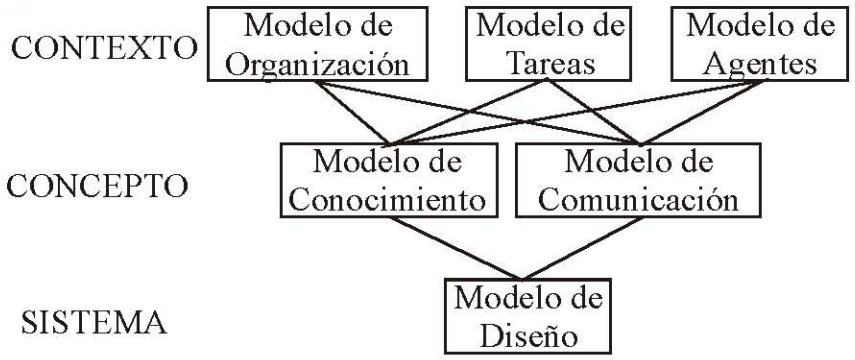


Figura 2.4 Conjunto de modelos CommonKADS.

El *modelo de conocimiento* o *modelo de experiencia*, como fue llamado en versiones anteriores de la metodología (Wielinga, et al., 1994), describe el conocimiento que tiene un determinado agente y que es relevante para la consecución de una determinada tarea, además de describir la estructura de este conocimiento en función de su uso. Este modelo se define en el nivel de conocimiento, sin hacer referencia a aspectos de implementación. Para poder llevar a cabo el modelado de los distintos papeles que puede jugar el conocimiento, éste se descompone en tres categorías disjuntas: el conocimiento de las tareas, del dominio y de las inferencias. El conocimiento de tareas describe de una forma recursiva la descomposición de una tarea de alto nivel en varias subtareas. El conocimiento del dominio especifica los hechos y asunciones que

necesita el proceso de razonamiento para llevar a cabo su cometido en el dominio de la aplicación. Por una parte se va a especificar la forma y estructura del conocimiento que es lo que se denomina ontología del dominio, y por otra, el contenido relevante para la aplicación. El último tipo es el conocimiento sobre inferencias que describe los procesos primitivos de razonamiento que tienen lugar en una aplicación, así como los roles de conocimiento que son usados por las inferencias.

El *modelo de comunicación* es el que se encarga de describir las transacciones que tienen lugar entre los agentes del sistema, así como de especificar los procesos de intercambio de información que materializan el intercambio de conocimiento entre agentes. (Waern, Hook, Gustavsson, & Holm, 1993)

El *modelo de diseño* se ocupa de la transición del nivel de conocimiento (modelado conceptual) al nivel simbólico (implementación). Esta transición se realiza a través de diferentes puntos de vistas: el diseño de la aplicación, el diseño de la arquitectura y el diseño de la plataforma, y se ocupa de la transición del nivel conceptual al nivel de implementación. Cada una de las visiones del diseño recoge un aspecto concreto del proceso de desarrollo. El diseño de la aplicación trabaja en el nivel conceptual de las entidades y tareas del mundo real que se tienen que llevar a la aplicación. El diseño de la arquitectura hace referencia a las estructuras computacionales abstractas en las que se tiene que materializar el diseño de la aplicación. El diseño de la plataforma indica el lenguaje de programación que se va a utilizar así como la plataforma hardware y software que va a ser utilizada. (Velde, et al., 1994), (Kingston, 1998) y (Cañadas, Palma, & Túnez, 2009)

2.3 Metodologías para sistemas híbridos

Los sistemas híbridos son aquellos en los que coexisten componentes basados en el conocimiento y sistemas de información convencionales. Los primeros intentan dar solución a las partes del problema en las que los requisitos están poco estructurados y son subjetivos, y en las que además están presentes la incertidumbre, la incompletitud y la inconsistencia, mientras los segundos presentan una solución válida para las partes del problema cuyos requisitos son objetivos, pudiendo tener un resultado cuantitativo o cualitativo.

Actualmente, las metodologías disponibles no resuelven la integración de componentes de estos dos tipos de forma exitosa, si se emplean métodos de InSo se pierde la expresividad necesaria para modelar conocimiento, y los métodos de la InCo no se pueden emplear si no hay conocimiento, tampoco existe un método integrado que aborde la construcción de estos sistemas híbridos como tales.

No hay consenso en cómo abordar el desarrollo de estos sistemas, distintos autores proponen diferentes líneas de trabajo en la búsqueda de una solución viable dependiendo de su formación o su experiencia. Para determinados aspectos del desarrollo, como los relativos a la gestión del proyecto se han propuesto soluciones que resuelven parcialmente el problema (Alonso, Fuertes, Martinez, & Montes, 2000) y (Águila I. M., Túnez, Cañadas, Bosch, & Marín, 2001) Para las actividades de desarrollo, que tal como se indicó en la sección 1.3 "Acotación de concepto de metodología" son el objetivo de esta tesis, también se han propuesto distintas soluciones metodológicas. Para su descripción se propone una agrupación en tres grandes categorías: a) aquellas basadas en sistemas multiagentes, b) las basadas en la utilización de la ingeniería de requisitos, y c) las basadas en mejoras de los modelos de procesos existentes.

2.3.1 Utilización de métodos para sistemas multiagente

Una línea de solución parte de los trabajos en los sistemas multiagente (MAS). Estos sistemas se pueden definir como una organización computacional consistente en diferentes entidades (agentes) que interactúan entre sí. Para los sistemas híbridos, la solución consiste en plantear una arquitectura de agentes algunos de los cuales serán basados en conocimiento y otros no. En los últimos años se han desarrollado numerosas metodologías apropiadas para el desarrollo de sistemas multiagentes, revisando las comparativas a distintos niveles de estas metodologías (Julián & Botti, 2003), no se puede determinar cuál es la mejor para el desarrollo de sistemas híbridos, pero las propuestas más recientes son las más elaboradas y completas al recoger los progresos de trabajos anteriores, es el caso de MaSE (Multiagent System Engineering) que junto con MAS-CommonKADS serán revisadas con mayor detalle en este apartado como solución multiagente para el desarrollo de sistemas híbridos.

MaSE es una metodología de propósito general para el desarrollo de sistemas multiagente heterogéneos que utiliza un conjunto de modelos gráficos

para describir los objetivos del sistema, el comportamiento de los agentes y la interfaz de comunicación entre ellos. (DeLoach, Wood, & Sparkman, 2001) y (Wood & DeLoach, 2001). MaSE plantea como una extensión o generalización de las técnicas orientadas a objetos y divide el desarrollo del sistema multiagente en dos grandes fases: el análisis y el diseño, figura 2.5 (Wood & DeLoach, 2001).

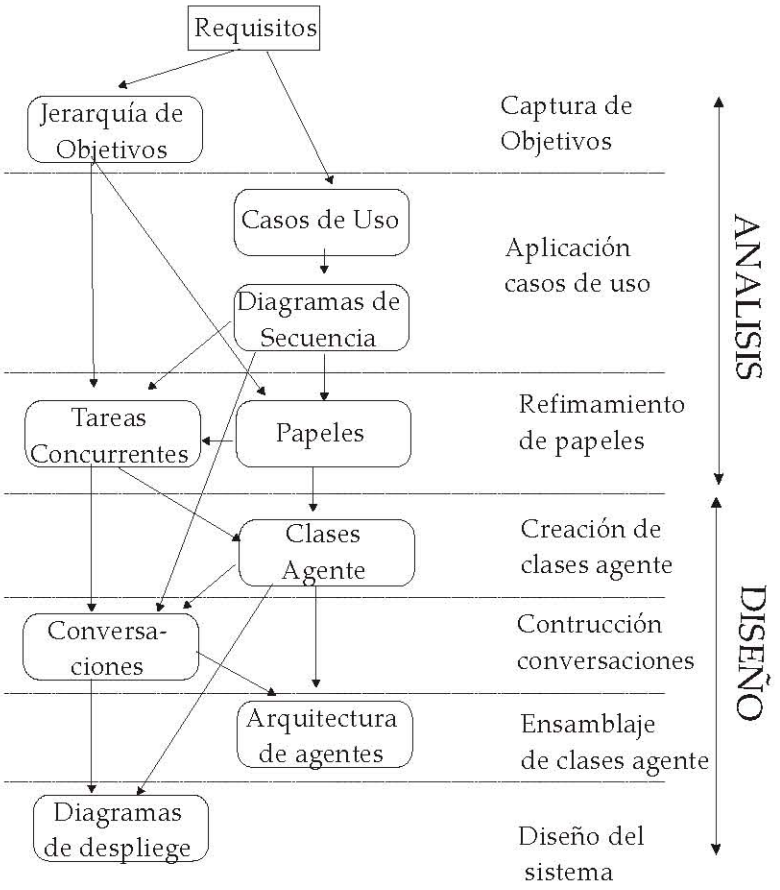


Figura 2.5 Fases de Multiagent System Engineering, MaSE.

El propósito de la fase de análisis de MaSE es producir un conjunto de roles o papeles que describen las tareas que tiene que hacer el sistema para cubrir todos sus requisitos, siendo un rol una entidad que realiza alguna tarea dentro del sistema. El planteamiento de la actividad de análisis de MaSE es bastante simple, y consiste en definir las metas u objetivos del sistema a partir de los requisitos funcionales y después definir los roles necesarios para resolver esos objetivos. MaSE sugiere la utilización de los casos del uso para ayudar a validar los objetivos del sistema y a derivar un conjunto inicial de roles. Esta actividad de

análisis se desarrolla en tres pasos. El primer paso consiste en *definir los objetivos* como casos de uso. Para esto se parte de una especificación inicial del sistema que se transformará en un conjunto estructurado de objetivos. El siguiente paso consiste en la *definición o aplicación de los casos de usos* en el contexto inicial y, posteriormente, generar los diagramas de secuencia que ayudan al analista de sistemas a identificar los roles y las vías de comunicación dentro del sistema. Los casos del uso definen los escenarios básicos que deben poder realizarse en el sistema. El objetivo del último paso del análisis, *refinar los roles*, consiste en transformar los objetivos y los diagramas de secuencia en roles y sus tareas asociadas que serán tratados con más detalle durante la fase de diseño.

La fase de diseño de un sistema con MaSE tiene cuatro pasos. El primer paso consiste en la *creación de las clases agente*, en las cuales el diseñador asigna roles a los tipos específicos de agente. En el segundo paso, se *construir las conversaciones reales entre las clases* agente, mientras que en el tercer paso, *ensamblar clases agente*, se diseñan la arquitectura interna y los procesos de razonamiento de las clases agente. Finalmente, en el último paso, *diseñar el sistema*, el diseñador define el número y la localización real de los agentes en el sistema desarrollado.

La Metodología de Análisis y Diseño de Sistemas Multi-Agente MAS-CommonKADS (Iglesias, 1998) surge como una extensión MAS de CommonKADS, orientada al modelado de Sistemas Multi-Agente. Se le añaden técnicas de las metodologías orientadas a objetos, así como de la ingeniería de protocolos para describir la comunicación entre los de agentes, tales como SDL (Specification and Description Language) y MSC (Message Sequence Charts).

El proceso general define unas fases básicas de *conceptualización, análisis, diseño, codificación, implementación* y *prueba*. La primera fase es la *conceptualización*, donde el analista determina casos del uso a partir de los requisitos iniciales del usuario y después los formaliza con los diagramas de la secuencia. El propósito de esta fase es capturar roles y desarrollar una comprensión inicial de las interacciones que deben ocurrir entre esos roles.

Después de la fase de la conceptualización, se genera una especificación de requisitos del sistema usando seis modelos, similares a los de CommonKADS. El modelo de la organización, que muestra las relaciones estáticas entre los agentes, y grupos de agentes, del sistema. El modelo de tareas que expresa las relaciones e interdependencias entre las actividades para alcanzar un objetivo. El modelo de

agentes específica las características del agente, que incluyen las capacidades del razonamiento, los sensores, los servicios, los grupos y las jerarquías. El modelo de conocimiento que define las fuentes de información que necesitan los agentes para alcanzar sus objetivos. El modelo de la coordinación que describe las conversaciones entre los agentes: sus interacciones, protocolos y capacidades necesarias. Finalmente, el modelo de la comunicación que detalla las interacciones del agente del humano-software y los factores humanos necesarios para desarrollar estos interfaces.

2.3.2 Aplicación de la ingeniería de los requisitos

Otras propuestas que resuelven el desarrollo de sistemas híbridos vienen de la mano de la Ingeniería de los Requisitos (InRe). El primer paso construir software consiste en establecer QUÉ debe hacer el sistema, es decir, definir los requisitos. Las descripciones de los servicios y restricciones son los requisitos para el sistema, y el proceso de obtener, analizar, documentar y verificar estos servicios y restricciones se llama ingeniería de los requisitos.

Uno de los problemas que tiene que afrontar la InRe es el hecho de que el propio término requisito no se usa de forma consistente en la industria del software. En algunos casos un requisito es considerado como una declaración abstracta de un servicio de alto nivel, más cercanos al negocio, que debe ofrecer el sistema, y en otros casos es considerado como una descripción detallada de algún aspecto del sistema referente a los datos o a las funciones que debe implementar. (Duran A. , 2000)

Si atendemos a la definición SWEBOK un requisito es: una propiedad que debe aparecer para resolver un problema del mundo real (Abran, Moore, Bourque, Dupuis, & Tripp, 2004). Sin embargo, a pesar de su aparente simplicidad, es frecuente encontrar el término requisito calificado con adjetivos que pueden resultar confusos en un primer momento: de sistema, hardware, software, de usuario, de cliente, funcional, no funcional, etc.

Por otra parte si revisamos algunos de los métodos de InSo plantean como mínimo un doble papel de los requisitos: a nivel del sistema objetivos ligados al negocio y a nivel de la componente software o los llamados requisitos del software (García, Ortin, Moros, Nicolás, & Toval, 2000), (Sommerville, Software Engineering (6th Edition), 2000), (Sommerville, Software Engineering (7th

Edition), 2004) y (Sommerville, Software Engineering: (Update) (8th Edition), 2006). En la metodología CK se realiza un estudio de los requisitos cuando se modela el contexto, con anterioridad a la tarea de definición del modelo de conocimiento. Sin embargo el análisis de requisitos de la aplicación software se realiza posteriormente, de cara a la definición de los modelos de diseño y a la implementación del producto software. RUP se centra sólo en los requisitos del software empleando en su identificación los casos de uso.

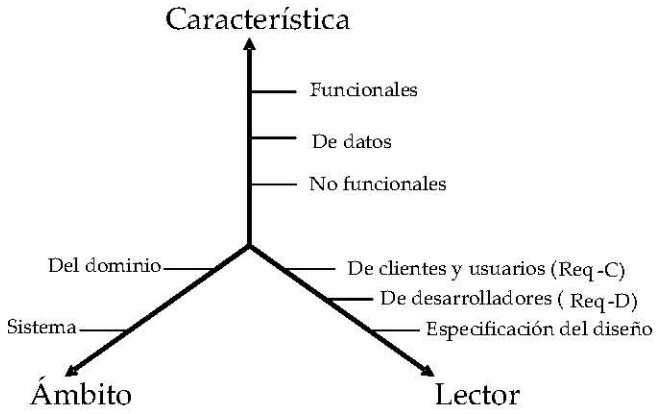


Figura 2.6 Tipos de requisitos según tres ejes ortogonales.

El concepto requisito muestra distintos aspectos ortogonales (Duran A. , 2000) De esta forma se pueden agrupar en torno a tres dimensiones (figura 2.6.):

- **Ámbito**: Indica el ámbito al que debe entenderse el requisito. El ámbito *sistema* es el relativo al conjunto de componentes del sistema completo. El ámbito *hardware o software* es el relativo a los requisitos de cada uno de esos componentes del sistema, y será estudiado con más profundidad y detalle en la construcción de esos componentes.
- **Característica**: Esta dimensión clasifica los requisitos en función de la naturaleza de la característica del sistema que se especifica. La clasificación más habitual suele ser la de requisitos funcionales y no funcionales. Se añaden los requisitos de datos o de almacenamiento de información, separados de los requisitos funcionales, que establecen qué información debe almacenar el sistema por ser relevante para las necesidades y objetivos de clientes y usuarios.
- **Audiencia**: Esta dimensión indica la audiencia a la que está dirigido el requisito, es decir, las personas que deben ser capaces de entenderlo. En general, se pueden distinguir dos tipos de audiencia, los clientes y usuarios, que no tienen porqué tener formación en InSo (requisitos-C), y los desarrolladores de software (requisitos-D). También se ha distinguido entre

una subcategoría que son los diseñadores, para aquellos requisitos relativos al diseño abstracto del sistema o de software. Cuando la audiencia está formada por clientes y usuarios, la forma más habitual de definir los requisitos es mediante lenguaje natural. En el caso de que la audiencia prevista esté formada por desarrolladores de software, los requisitos suelen expresarse mediante un modelo, normalmente utilizando técnicas estructuradas, orientadas a objetos o formales.

SWEBOK propone un modelo iterativo para las tareas ligadas a la InRe que se muestra en la figura 2.7 (a): *adquisición*[9]*, análisis y negociación, especificación y validación*. Otra de las metodologías para InRe es la propuesta por (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000), que consiste en un modelo de procesos iterativo en el que se identifican tres actividades principales (figura 2.7 (b)): *adquisición, especificación y validación*, la especificación se distribuye entre las otras tareas. En la figura 2.7 (c) y (d), también se muestran otra clasificación de las tareas relativas a esta ingeniería propuestas por otros autores que son bastante similares (Sommerville, Software Engineering: (Update) (8th Edition), 2006) y (Loucopoulos & Karakostas, 1995).

La *adquisición de requisitos* no se ha considerado parte del ciclo de vida de desarrollo de software en los inicios de la InSo. Se solía asumir que el cliente proporcionaba los requisitos, de forma que el ciclo de vida comenzaba siempre por la documentación y el análisis de unos requisitos ya dados. Cuando se ha detectado que los problemas en los requisitos son uno de los principales factores de los fracasos de los proyectos software (Glass, 2002), es cuando se le ha comenzado a dar importancia al proceso de obtención de esos requisitos. Sin embargo dentro del desarrollo de SBC, la tarea de adquisición del conocimiento siempre ha sido considerada como de gran importancia y la piedra angular en la construcción de estos sistemas, convirtiéndose en muchos casos en el único objetivo de la InCo.

El estudio de los requisitos es iterativo por naturaleza, ya que es prácticamente imposible obtener todos los requisitos sin tener que volver atrás en

[9] Existen diversas formas de traducir el término "elicitation", algunos autores lo traducen por elicitación, vocablo no existente en castellano o por el cultismo educción. El significado en castellano es sonsacar, obtener con dificultad y por tanto se ha optado por aducir "elicitation" como adquisición u obtención de información.

algún momento del proceso. El ciclo principal *adquisición–análisis–especificación–validación* tiene en cuenta la posibilidad de que, durante el proceso de validación de los requisitos por parte de los clientes y usuarios, aparezcan conflictos o nuevos requisitos que hasta entonces estaban ocultos. En esas circunstancias, es necesario resolver dichos conflictos y consensuar los nuevos requisitos, repitiendo a continuación las actividades de análisis y validación.

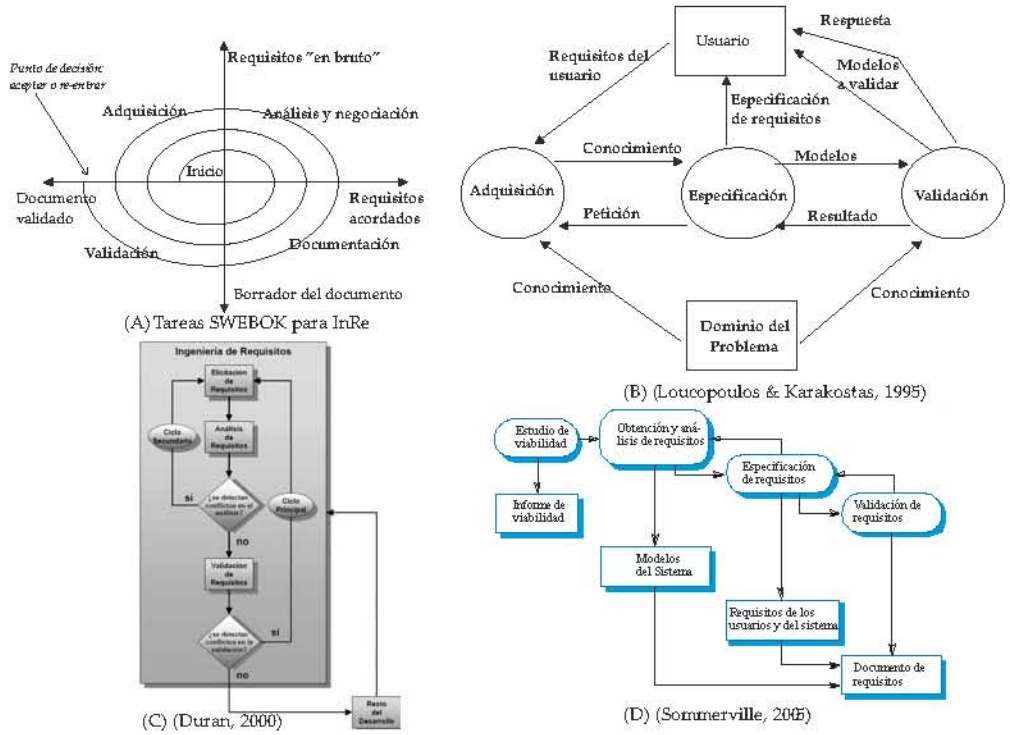


Figura 2.7 Ciclos en la ingeniería de requisitos.

Una constante que se presenta en el desarrollo cualquier tipo de software es la dificultad de adquirir los requisitos de una forma completa que conduzcan a la determinación del diseño del sistema correcto. Las técnicas que se emplean para la *adquisición de los requisitos* se basan, principalmente en plantillas y patrones de requisitos. Los objetivos principales de esta actividad son, en primer lugar, conocer el dominio del problema para poder entenderse con los clientes y usuarios; y en segundo lugar, descubrir las necesidades reales de clientes y usuarios. Por último, la adquisición de los requisitos tiene por objetivo consensuar los requisitos entre los propios clientes y usuarios, y a veces es

necesario negociar entre los distintos participantes hasta obtener una visión común de los requisitos (Boehm, Bose, Horowitz, & Lee, 1994).

Como resultado de la *adquisición* se obtienen los requisitos-C (Braude E. J., 2000), (Braude E. J., 2005) o requisitos definidos desde el punto de vista del cliente o simplemente requisitos, tanto para datos como para funciones del sistema. La *especificación de los requisitos* consiste en generar un documento o su equivalente electrónico, que especifica los requisitos, y que supone el contrato entre cliente y desarrollador y es la línea base para el desarrollo.

Los dos objetivos principales de la actividad de *análisis de requisitos* son por una parte detectar conflictos en los requisitos–C y profundizar en el conocimiento del dominio del problema, normalmente mediante la construcción de un modelo que conlleva un incremento en el grado de conocimiento del problema, y que puede facilitar el proceso de construir un producto útil para clientes y usuarios. En la mayoría de los casos los modelos obtenidos durante la realización del análisis de los requisitos–C suelen constituir las bases para las actividades de diseño. El resultado de esta actividad de análisis son los requisitos–D (desde el punto de vista del desarrollador) o modelos de análisis (Duran & Jiménez, Metodología de análisis de requisitos de sistemas software. Versión 2.1, 2001).

La actividad de *validación de requisitos* debe confirmar que los requisitos–C, una vez analizados y resueltos los posibles conflictos, corresponden realmente a las necesidades de clientes y usuarios. Se trata de una actividad que debe ser realizada principalmente por clientes y usuarios, aunque los ingenieros de requisitos pueden ayudar en el proceso mediante las explicaciones oportunas o mediante el uso de prototipos.

En el desarrollo de un sistema híbrido son de vital importancia las tareas ligadas a la definición de los requisitos ya, que tal como se apunto en la sección 1.2 "Necesidad de integración de InCo e InSo", su mala puesta en práctica puede llevar a que requisitos asignados inicialmente para ser resueltos con un SBC, evolucionen hacia componentes software no basados en conocimiento o viceversa. Por esta razón, se debería permitir posponer la decisión del tipo de técnicas y métodos a utilizar, hasta que se puedan aplicar criterios concretos que nos hagan optar por una o por otra. Es decir, mejorar el proceso de obtención y análisis de requisitos del sistema a implantar empleando las técnicas y metodologías de InRe para el desarrollo de estos sistemas.

2.3.3 Mejoras al modelo de proceso aplicado

La última categoría de soluciones para el desarrollo de sistemas híbridos se recoge en los trabajos que mejoran el modelo de proceso de desarrollo, bien planteando modelos de proceso alternativos (Acuña, Lopez, Juristo, & Moreno, 1999), o bien mediante una definición por capas (Chen, Kendal, Potts, & Smith, 1997) o basándose en la separación entre la definición del sistema a nivel de contenido, ligado al conocimiento, y continente o contenedor, ligado a la implementación software (Gachet & Haettenschwiler, 2003). Por último, cabe destacar, la aproximación propuesta por nuestro grupo en un trabajo previo, donde se determina la necesidad de un modelo de análisis que integra software que usa conocimiento y que no lo usan (Águila I. M., Cañadas, Palma, & Túnez, 2003). Pero en todos los casos se tratan de propuestas descriptivas que necesitan ser desarrolladas en cuanto a nivel de detalle, modelos utilizados, lenguaje de modelado, detalle de actividades, artefactos y documentos generados.

Gachet (Gachet & Haettenschwiler, 2003) considera que un sistema de apoyo a la decisión tiene dos componentes intrínsecas: el soporte para la decisión y el sistema. El primero se corresponde con las tareas de decisión a alto nivel y el segundo con el desarrollo del software propiamente dicho. La propuesta bipartita es una aproximación que reconcilia estas dos caras, separando entre continente y contenido del sistema de ayuda a la decisión, el primero retiene los componentes puramente técnicos y el segundo la base de conocimiento. La interfaz entre ambos componentes se realiza mediante el componente llamado o motor de inferencia que es el cerebro. Por una parte, el motor de inferencia utiliza conocimiento de la base de conocimiento (contenido) para generar nuevos hechos y/o conocimiento y, por otra, se establecen relaciones entre el contenido y el continente, de forma que en determinadas situaciones el motor de inferencia delega sobre el continente.

La metodología HyM (Chen, Kendal, Pott, & Smith, 1996) ve el sistema híbrido de forma jerárquica a tres niveles. El primero lo forma el propio sistema híbrido. El segundo nivel es el nivel de componentes, que pueden ser basadas en conocimiento, tradicionales o híbridas. Cada uno de estos tipos de componentes utiliza los repositorios que se colocan en el nivel inferior de la jerarquía que son de procesamiento, de razonamiento o de uso de bases de datos. Con esta metodología se han propuesto diseños de sistemas para la gestión integral de un hospital con componentes tradicionales como la gestión de los datos de admisión

de pacientes, SBC puros como la planificación de tareas y, por último, componentes híbridos para la el diagnóstico de vértigos (Rashad, Arulledran, Hawthorne, & Kendal, 2001). Para la construcción de un sistema híbrido, esta metodología propone un ciclo de vida iterativo, en el que se definen un conjunto de tareas básicas en las que las iteraciones se fijan en función de la complejidad del proyecto. Estas actividades básicas son el estudio de viabilidad, el análisis y diseño, la implementación, la evaluación y el mantenimiento.

Acuña (Acuña, Lopez, Juristo, & Moreno, 1999) defiende la integración y convergencia entre InCo e InSo, puesto que en ambos casos el objetivo es construir y mantener un producto software que satisfaga las necesidades detectadas por los usuarios. Su propuesta es un modelo cualitativo, informal y descriptivo cuya importancia radica en que adopta, unifica e integra las actividades para construir sistemas software tradicionales y para el desarrollo de SBC. Acuña propone también procesos ligados a la gestión del proyecto, el modelo de ciclo de vida o de soporte, aplicando el modelo estándar IEEE Std 1074-2006 (Institute of Electrical and Electronics Engineers Staff, 2006). Pero sólo revisaremos los llamados procesos de modelado de software que tal como se indicó en la sección 1.4 "Hipótesis de partida y objetivos" son el objetivo de este trabajo de tesis. Se da este nombre al proceso porque el desarrollo de software es básicamente un proceso de modelado que conlleva la construcción de una serie de modelos intermedios a diferentes niveles de abstracción.

Esta propuesta considera cuatro subprocesos dentro del modelo de proceso de modelado de software que son: *Proceso de exploración, de comprensión del problema, de construcción del sistema* y *de utilización*. El primero comprende el estudio, análisis y diagnóstico de la situación, lo que proporciona un reconocimiento y formalización del problema apropiado, una definición y delimitación de la necesidad de una solución computacional, además de determinar la viabilidad del sistema a desarrollar. Por tanto, se han definidos dos actividades principales, que son la elaboración del estudio del dominio y del estudio de viabilidad.

El *proceso de comprensión* del problema, conlleva el análisis conceptual y su objetivo es obtener una definición del problema, además de la compresión del conocimiento relevante y los procesos de resolución del problema, así como obtener una especificación de los requisitos del software. Este proceso se realiza en tres actividades: *el análisis del entorno* (relación del sistema con el entorno), *el*

análisis de conocimiento (conceptos, relaciones, inferencias y tareas a nivel estratégico, táctico y factual) y *el análisis de los requisitos* (que esperan los usuarios del sistema, desde el punto de vista del usuario).

Tras la comprensión del problema, el *proceso de construcción* del sistema tiene por objetivo desarrollar el modelo del sistema correspondiente con el modelo del problema definido en procesos anteriores. Se realizan actividades de diseño, implementación e integración del sistema software, y supone un salto de los modelos cualitativos hasta modelos cuantitativos. El último proceso propuesto por Acuña es el *proceso de utilización,* que incluye las tareas que hay que realizar para instalar, poner en funcionamiento, mantener y retirar un producto software.

Las tres soluciones descritas para el desarrollo de sistemas híbridos: a) basadas en sistemas multiagentes, b) las basadas en la utilización de la ingeniería de requisitos, y c) las basadas en mejoras de los modelos de procesos existentes aun aportando notables beneficios frente a las alternativas más clásicas descritas en las secciones anteriores tienen problemas y limitaciones que se describirán en el siguiente apartado.

2.3.4 Limitaciones detectadas

La nueva propuesta de modelo de proceso InSCo viene a resolver algunos de los problemas se presentarían si para el desarrollo de un sistema híbrido se emplease alguno de los enfoques descritos en los tres apartados anteriores. Desarrollar estos sistemas híbridos como sistemas MAS puede ser un enfoque apropiado cuando en la solución se deban tener en cuenta técnicas multiagente. Sin embargo, no es cierto en la mayoría de los casos, la mayor parte de las metodologías como MaSE o MAS-CommonKADS centran gran parte de su esfuerzo en la definición de las tareas de coordinación y las conversaciones entre los agentes. Concretamente, en el caso se MaSE no se aborda el modelado en el nivel de conocimiento, simplemente se indica que algunos agentes serán basados en conocimiento, pero este no es tratado como un recurso modelable y compartible[10]. En el caso de MAS-CommonKADS la solución es sólo para las

[10] La línea de investigación dedicada al Knowledge Sharing centra sus esfuerzos en la definición de los medios, que permitan compartir el conocimiento y que son el origen de trabajos en Ontologías y Semantic Web.

partes del sistema que se modelan como agentes basados en conocimiento y no se define como integrarlos con aquellos que no lo sean. No obstante, podemos trasladar a nuestra propuesta la idea de la utilización de los casos de uso para tener un proceso sin costuras y, sobre todo, el modelado de los mecanismos de coordinación entre las funciones o papeles basados y no en conocimiento utilizando los planteamientos de las metodologías MAS.

La utilización de las técnicas de InRe es una aportación acertada. En trabajos previos realizados en el grupo de investigación (Gómez-Skarmenta, Vila, & Túnez, 2000) se ha detectado que los clientes y/o usuarios tienen dificultades a la hora de proporcionar los requisitos del sistema a construir. Además, no se les puede pedir que clasifiquen los requisitos, ni que separen el conocimiento de lo que son simplemente datos de un formulario. Por lo tanto, se debe posponer, en la medida de lo posible, la decisión de cuáles son los requisitos basados en conocimiento y cuáles no, para lo que las metodologías de InRe son esenciales. Pero estas técnicas deben integrarse perfectamente dentro de todo el proceso de desarrollo, abarcando el proceso completo hasta llegar a la instalación y puesta a punto del software. Ya que InRe fundamentalmente sólo se ha aplicado a los problemas InSo, un aspecto que no aparece bien definido en los métodos de InRe es cómo tratar la información imprecisa o aproximada que nos encontramos dentro del campo de la InCo.

En la definición del modelo de proceso InSCo se ha optado por la línea de trabajo de mejora del modelo de proceso en el desarrollo de sistemas híbridos. Los trabajos referenciados en párrafos anteriores de esta sección son propuestas de otros modelos de proceso algunas más acertadas que otras. La metodología HyM es demasiado estricta y poco flexible. Define tareas básicas que se descomponen en subtareas y éstas a su vez en otras. Se utilizan listas de comprobación de trabajo, pero a pesar de este detalle no queda claro cómo decidir si el componente será de tipo tradicional, basado en conocimiento o híbrido. En resumen, es un modelo poco flexible y difícil de seguir o instanciar en proyecto reales. En la definición del modelo de proceso InSCo se ha optado por un modelo más flexible, empleando el concepto de actividad fundamental más general y adaptable. La propuesta de Acuña et al. si es flexible, pero tal como indican las autoras define un modelo de proceso cualitativo, informal y descriptivo que, para ser aplicable sobre proyectos reales tiene que describirse de forma mucho más precisa de lo que aparece en este trabajo.

Con esta idea se aborda el trabajo de esta tesis, de forma que una vez definidas las actividades fundamentales para todo el modelo de proceso en el capítulo 3 "Modelo de proceso InSCo" y en los capítulos siguientes se hace una descripción más completa y precisa de las técnicas y métodos a utilizar a nivel de modelado conceptual.

2.3.5 Características deseables de la propuesta InSCo

De análisis del estado del arte en cuanto a cómo desarrollar sistema híbridos y de las metodologías de InCo e InSo en general, se desprenden una serie de condiciones o propiedades que se han inyectado sobre el modelo de proceso propuesto de cara a garantizar el éxito de su implantación y la minimización del coste de cambio. A continuación se enuncian y resumen estas características, que después se verán reflejadas sobre la propuesta.

- El **núcleo** de cualquier metodología actual **es el modelo de proceso** y este debe ser el punto de posible unificación. Cada autor propone diferentes agrupaciones de las actividades a realizar en el desarrollo de software, pero en esta tesis se propone partir de flujos de trabajo o actividades genéricas que se instancien en actividades concretas en cada enfoque o paradigma y, de esta forma, se obtendrá un marco común que facilite la integración de las metodologías, sean de la categoría que sean, guiadas por el modelo de proceso InSCo.
- El modelo de proceso propuesto debe definir un entorno de trabajo que permita una aproximación inicial al software a construir, independientemente de que finalmente se construya con técnicas algorítmicas o heurísticas o que el resultado esté basado o no en el paradigma orientado a objetos, es decir debe definirse como un **modelo genérico**.
- No existe una única forma de desarrollar proyectos software, por tanto necesitamos que el modelo de proceso InSCo sea un **modelo de proceso flexible**, es decir, que permita configurar la ejecución del proceso de forma que se ajuste al proyecto y/o al grupo de trabajo. Este planteamiento lo recoge RUP. Aunque algunos autores (Beck & Andres, 2004) y (Cockburn, Agile Software Development, 2001) consideran una metodología pesada, RUP es más flexible que el enfoque estructurado o propuestas similares.
- Para mantener estas características de flexibilidad o adaptabilidad, el modelo de proceso se debe basar en la definición de **actividades fundamentales,** consideradas éstas como grupos de actividades relacionadas y bajo un mismo

nombre. Este enfoque basado en actividades fundamentales de carácter estático es el que marcan las tendencias de la InSo, tanto estructuradas como orientadas a objetos.

- De esta forma se define qué hay que hacer, los artefactos y técnicas que se deben utilizar, pero no se fija a priori la secuenciación o el número de revisiones a realizar por cada artefacto generado. Esto hace que, de forma general, el modelo a seguir sea **iterativo** e **incremental**, tal y como se recomienda para el modelado de conocimiento (Schreiber, et al., 1999) y (Angele, Fensel, Landes, & Studer, 1998), para la obtención de los requisitos en la propuesta de Durán (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000), o para la obtención de la arquitectura y resto de artefactos en RUP (Booch, Rumbaugh, & Jacobson, Unified Modeling Language User Guide, The (2nd Edition), 2005), (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual (2nd Edition), 2004) y (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999).
- Desde que Newell definiese el **nivel de conocimiento** (Newell, The knowledge level, 1982), el desarrollo de sistemas software cuyo ámbito quede dentro del campo de los SBC necesitan modelar de forma explícita el conocimiento que los usuarios y/o expertos aplican en la realización de sus procesos de razonamiento. Estos modelos tiene entidad en sí mismos, independientemente de si se implemente o no el software que lo utilice y representan el conocimiento utilizado por los miembros de una organización y, al ser explicitados, pueden servir de base para aplicar políticas centradas en la gestión de conocimiento. En muchas ocasiones el trabajo de los ingenieros del conocimiento consiste en la definición de estos modelos de conocimiento que posteriormente pueden ser adaptados a los diferentes entornos o dominios para así construir software inteligente (Túnez, Águila, & Marín, 2001), (Águila I. M., Cañadas, Bosch, Túnez, & Marín, 2003) y (Túnez, Marín, Águila, Bosch, & Torres, 1998).
- Tal como marcan las tendencias actuales, tanto en las metodologías InSo como InCo, el **enfoque de modelado** es el más apropiado el desarrollo de software. Un modelo es una abstracción del sistema semánticamente cerrada que supone una representación explicita de una porción de la realidad percibida por un actor. Un modelo tiene un punto de vista y un nivel de abstracción en el que centrar su definición del sistema. En el desarrollo de software hay dos formas de utilizar los modelos. En primer lugar tenemos aquellos que se utilizan para describir el problema que será resuelto por el software, son los denominados **modelos conceptuales**. En segundo lugar,

tenemos los **modelos computacionales** que se utilizan para describir el propio sistema software que resuelve el problemas (Blum, 1996), al que se debe añadir el modelo de conocimiento. Se debe hacer que los modelos conceptuales sean independientes de la implementación, pero siempre es necesario un punto de **conexión con los aspectos computacionales**, lo más acotado posible, y en los sistemas híbridos con más razón para inyectar el conocimiento en el lugar oportuno.

- Para mejorar la productividad del desarrollo, todas las metodologías potencian la **reutilización** a nivel de modelos completos o de componentes de los modelos (Heineman & Councill, 2001), (Henry & Faller, 1995) y (Taboada, Des, Mira, & Marín, 2001), sin olvidarnos por supuesto de la reutilización de patrones de diseño y de código.
- Antes de construir software propiamente dicho se debe hacer un análisis exhaustivo del sistema donde se ubicará. Este trabajo entra dentro del campo de la Ingeniería de Sistemas Informáticos, más compleja y menos evolucionada que la propia InSo o la InCo. Los sistemas de información están compuestos por una colección de componentes interrelacionados que entran habitualmente en alguna de las siguientes categorías: personas, hardware, software, bases de datos, redes o procedimientos. El software es sólo un elemento más con su propia ingeniería asociada, la InSo. Sería demasiado ambicioso plantear la integración real, y no sólo de forma descriptiva, de la ingeniería de sistemas dentro del modelo de proceso propuesto en este trabajo de tesis, ya que considerando sólo la componente software existen suficientes dificultades para integrar software basado y no basado en conocimiento. Sin embargo, queda claro que para garantizar el éxito de un proyecto es necesario que se tomar en cuenta el **entorno donde se ubicará el sistema** (Post, Wielinga, Hoog, & Schreiber, 1997), de aquí que el modelo de proceso debe dedicar un grupo de actividades a definir el problema y delimitar de forma precisa el contexto de la solución propuesta, es el llamado **modelado del negocio** (García, Ortin, Moros, Nicolás, & Toval, 2000).
- Uno de los principales problemas que dieron origen a la crisis del software reside en las primeras etapas del desarrollo, donde se deciden las características del producto software a desarrollar. De aquí se sea tan importante una delimitación clara e independiente de los **requisitos.** Esta tarea en muchas ocasiones lleva asociada la negociación de los objetivos del sistema con el cliente, para definir un conjunto de requisitos viable para el software a construir. Este aspecto, claramente reflejado en los métodos de la InSo, no se refleja sobre ninguno de las metodologías basadas en conocimiento, pero el modelo de proceso InSCo debe heredarlo y representar

los **distintos tipos de requisitos**, que se diferencian, tal como se ha descrito en el apartado 2.3.2 "Aplicación de la ingeniería de los requisitos", en la característica que describen, el ámbito o momento en el que aparecen y la audiencia hacia la que van dirigidos.

- Se busca un modelo de proceso que permita que los proyectos, ya desarrollados con alguna metodología, pueda encajar en este nuevo conjunto de actividades sin problemas de adaptación de cara al **mantenimiento** y la evolución de estos sistemas. Con este mismo espíritu de convergencia y unificación se deben **minimizar los medios de notación** e intentar adecuarlos a los estándares o cuasi estándares conocidos, por eso vamos a proponer la utilización de UML en la descripción de los modelos. Otra de las técnicas de notación más utilizada en metodologías convencionales para la obtención y especificación de requisitos son las **plantillas o formularios**, que ofrecen un gran potencial para la estructuración y partición del problema.

Estas son las características básicas que se han puesto en práctica sobre el modelo de proceso InSCo y que se reflejaran en las actividades y artefactos de han sido propuestos en este trabajo de investigación.

2.4 Metodologías soporte

Puesto que el objetivo es la definición de un modelo de proceso, al que se ha denominado InSCo, que permita la integración de metodologías propias de la InSo e InCo y no la definición de un modelo de proceso completamente nuevo, InSCo ha buscado puntos de convergencia entre las distintas metodologías, que en ciertos casos utilizan herramientas y notación comunes. El punto de partida son tres de las metodologías más ampliamente utilizadas en ambas ingenierías (Estructurada, Rational Unified Process y CommonKADS). Estas metodologías han sido seleccionadas para que el modelo de proceso propuesto en este trabajo pueda ser adoptado tanto por los ingenieros del software como por los ingenieros del conocimiento experimentados en ellas. A partir de ahora las llamaremos metodologías soporte para la definición del enfoque integrado del nuevo modelo de proceso InSCo.

La figura 2.8 muestra la evolución de las metodologías para la construcción de software en la que se pueden destacar tres grandes hitos que, como puede observarse, presentan un ligero desfase entre su aparición en la InSo y la InCo, marcando los momentos de unificación entre metodologías. El primero de ellos es la necesidad del uso de metodologías de desarrollo que dan lugar

respectivamente a los conceptos de ingeniería del software e ingeniería del conocimiento. El segundo es la migración hacia el enfoque de modelado, donde el objetivo es la elaboración de modelos que sirvan de base a la construcción de software. Por último, nos encontramos con la separación entre producto y proceso, ya presente en las metodologías de InSo pero ausente en la mayor parte de las metodologías actuales de InCo, que adolecen de una definición clara e independiente del modelo de proceso. Estos hitos constituyen puntos de convergencia entre las metodologías existentes y otras nuevas que recogen y adaptan lo mejor de cada una de ellas. En este sentido, el modelo de proceso InSCo supondrá un nuevo punto de convergencia hacia una metodología común para el desarrollo de los sistemas software híbridos.

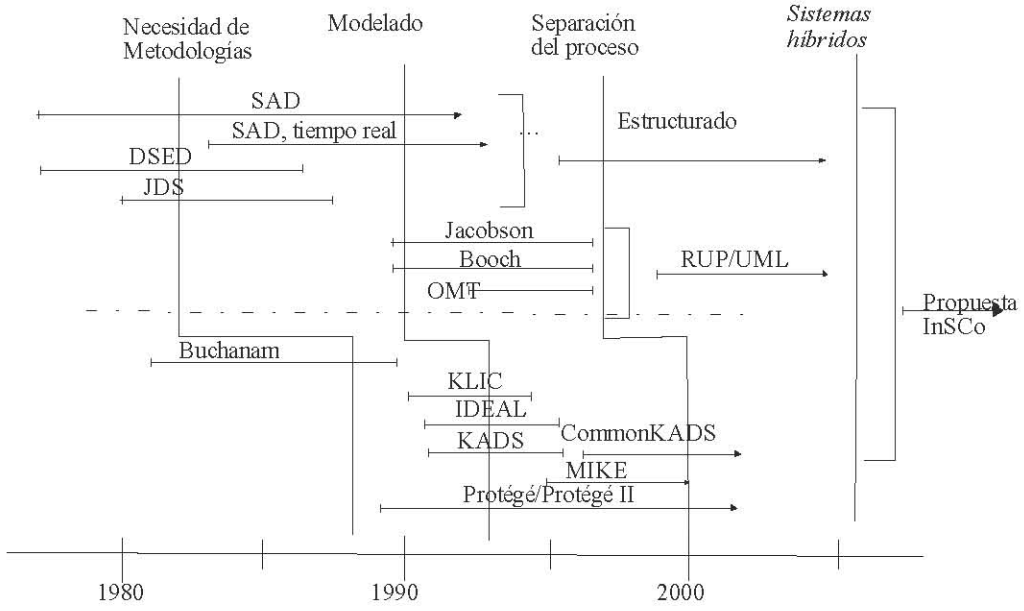


Figura 2.8 Evolución de metodologías.

Las metodologías de los primeros años de la InSo, como la metodología estructurada clásica (SAD) (Demarco, 1979), el diseño estructurado de datos (DSED) (Orr, 1977), la metodología de Jackson (JSD) (Jackson, System Development, 1983) o sus extensiones para tiempo real (Ward & Mellor, 1985) y (Hatley & Pirbhai, 1988), tenían su propia notación. Como ya se ha explicado a lo largo de este capítulo estos métodos han ido evolucionando y han convergido sobre una misma visión del problema de definición y construcción del software,

formando lo que hoy llamamos metodologías estructuradas de desarrollo de software.

Otro gran grupo de metodologías aparecen ligadas a la orientación a objetos. Estos métodos son atractivos para las organizaciones de desarrollo de software debido a que su objetivo es potenciar la reutilización. Al igual que con los métodos estructurados, se ha producido una convergencia. Así del trabajo conjunto de los tres más destacados líderes en tecnología de objetos (Booch, Object-Oriented Analysis and Design with Application, 1993), (Jacobson, Christerson, Jonsson, & Övergaard, 1992) y (Rumbaugh, Blaha, Lorensen, Eddy, & Premerlani, 1990), ha surgido UML como lenguaje de notación de RUP, (Booch, Rumbaugh, & Jacobson, Unified Modeling Language User Guide, The (2nd Edition), 2005), (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual (2nd Edition), 2004) y (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999).

En este punto cabe decir que existe otro tipo de metodologías que están pisando fuerte en el ámbito del desarrollo del software, como son las metodologías ágiles, como Extreme Programming (XP) (Beck & Andres, 2004) o Scrum (Schwaber & Beedle, 2001), pero no se incluyen en la presente propuesta porque se centran en el desarrollo de software de tamaño pequeño o mediano y, por tanto, no son apropiadas para sistemas híbridos.

A la primera generación de metodologías para construir SBC pertenecen KLIC, IDEAL o KADS que evolucionan y convergen entre sí a principios de los noventa, replanteando la InCo desde la perspectiva de la transferencia hacía la perspectiva del modelado, como un intento de solucionar el problema del cuello de botella que suponía la fase de adquisición de conocimiento (Shadbolt, Motta, & Rouge, 1993). Las nuevas metodologías son las llamadas metodologías de segunda generación. Las más importantes son CommonKADS (Schreiber G. , Wielinga, Hoog, Akkermans, & Velde, 1994), (Schreiber, et al., 1999) MIKE (Angele, Fensel, Landes, & Studer, 1998), y PROTÉGÉ-II (Eriksson, Shahar, Tu, Puerta, & Musen, 1995). Estas metodologías son el primer intento de ofrecer a los constructores de SBC una solución al ciclo de vida completo del sistema. Por otra parte, la necesidad de mejorar la productividad, al igual que ocurrió en su momento en la InSo, da lugar a la potenciación de la reutilización de los componentes de conocimiento, de forma similar a la reutilización de clases y objetos en el paradigma de desarrollo de software basado en objetos. Cada

metodología plantea su propia aproximación al modelado. Cada una tiene su propio ciclo de vida y métodos de gestión del proyecto, pero no reflejan aun la clara separación entre producto y proceso. Este tipo de metodologías se encuentran en proceso de evolución hacia una convergencia, tal y como ocurrió con los métodos orientados a objetos.

Para definir el modelo de proceso InSCo propuesto en este trabajo se ha elegido las metodologías: Estructurada, RUP y CommonKADS. La primera y la segunda se seleccionan porque son una cobertura de todos los enfoques de desarrollo de sistemas software que se utilizan actualmente. En el caso de CommonKADS se ha seleccionado porque es una de las más completas (Plant & Gamble, 1997) y (Palma & Marín, 2008), que comienza por el modelado a nivel de contexto enmarcando el SBC a construir dentro de la organización de la que va a formar parte. Otro factor que nos ha llevado a su elección es que se trata de una de las más extendidas en Europa y que propone como notación el lenguaje UML para muchos de sus modelos. Protégé ha sido descartada porque cubre sólo la construcción de herramientas para la adquisición del conocimiento, de hecho existen trabajos en los que se propone la utilización de Protégé como herramienta CASE para la metodología CommonKADS (Schreiber, Crubézy, & Musen, A Case Study in Using Protégé-2000 as a Tool for CommonKADS, 2000). MIKE se centra en el proceso de formalización y en la utilización de lenguajes formales para recoger el conocimiento. Esta última también ha sido descartada puesto que históricamente los llamados métodos formales en la InSo no han sido utilizados y las empresas no consideran costeable aplicarlos en sus procesos software (Sommerville, Software Engineering: (Update) (8th Edition), 2006).

2.5 Conclusiones del capítulo

Se ha realizado una revisión del estado del arte de las metodologías más representativas de la InSo y de la InCo, prestando especial atención a aquellas soluciones propuestas para el desarrollo de sistemas software híbridos, indicando las limitaciones y aportaciones de cada una de ella, que se resolverán con la nueva propuesta de modelo de proceso InSCo.

El modelo de proceso InSCo se opta por la línea de trabajo de mejora del modelo de proceso en el desarrollo de sistemas híbridos. Se plantea un modelo

flexible, empleando el concepto de actividad fundamental más general y adaptable.

Se aúnan muchas de las características de las metodologías previas (InCo e InSo) y que consideramos como características deseables de la propuesta InSCo: el núcleo es el **modelo de proceso**, este modelo es **genérico** y **flexible**, se centra en la definición de **actividades fundamentales**, es **iterativo** e **incremental**, se define en el nivel de conocimiento, se centra en la definición de modelos, **separa** modelos **conceptuales de computacionales**, fomenta la **reutilización**, se propone el **modelado del negocio** y los **requisitos** y **minimiza la notación**.

Puesto que el objetivo es la integración de metodologías y no la definición de un modelo de proceso completamente nuevo, InSCo ha buscado puntos de convergencia entre las distintas metodologías, que en ciertos casos utilizan herramientas y notación comunes. El punto de partida son tres de las metodologías, que llamaremos metodologías soporte: **Estructurada**, **Rational Unified Process** y **CommonKADS**. Estas metodologías han sido seleccionadas para que InSCo pueda ser adoptado tanto por los ingenieros del software como por los ingenieros del conocimiento experimentados en ellas.

3 Modelo de proceso InSCo

En este capítulo se realiza la descripción de las actividades y los artefactos que se manejan y se generan en la propuesta de modelo de proceso InSCo. Para definir un proceso es necesario describir el *Dónde*, el *Qué* y el *Cómo*, representados por las actividades fundamentales y los artefactos. En los siguientes capítulos se tratará la descripción detallada de estas actividades fundamentales.

El modelo de proceso propuesto en este trabajo se acota al nivel conceptual, de forma que se describirán con detalle los métodos y técnicas a utilizar durante la ejecución del modelo de proceso sólo en el nivel conceptual de solución al problema, es decir, los aspectos independientes de la implementación.

Este capítulo se estructura en primer lugar delimitando el significado de los términos que se usan en la descripción del modelo de proceso, para en la sección 3.2 "Metamodelo y Artefactos InSCo" realizar la definición del metamodelo de la propuesta. En la sección 3.3 "Descripción el modelo de proceso InSCo", se describen tanto las actividades fundamentales como los modelos que se generan, dedicando un apartado para la definición de los hitos básicos que se deben definir cuando se aplica el modelo InSCo. Finalmente se verifica la cobertura que la propuesta supone de los demás modelos de proceso y estándares, en la sección 3.4 "Compatibilidad con otros modelos de proceso".

3.1 Concepto de modelo de proceso

Antes de describir la propuesta del modelo de proceso InSCo, que se realizará en las siguientes secciones de este capítulo, conviene revisar la terminología y concretar el significado de los conceptos asociados a cada término para la elaboración de la propuesta desarrollada en este trabajo, todos ellos representados en la figura 3.1. Es conveniente realizar estas puntualizaciones al tratarse de conceptos que han ido matizándose, de forma no totalmente sinónima, durante la evolución de la InSo y la InCo.

El **modelo de proceso software,** o por simplificar **proceso,** es una definición de un conjunto completo de actividades necesarias para convertir los servicios, o necesidades del clientes, en un conjunto consistente de artefactos que conforman un producto software, y para convertir los cambios sobre estos requisitos, en un nuevo conjunto consistente de artefactos, es decir, permitir la evolución del software. Hay que distinguirlo del concepto de **ciclo de vida** que se centra en el producto, describiendo los estados por los que evoluciona el software (Acuña, López, Mate, Ferre, & Chang, 2002).

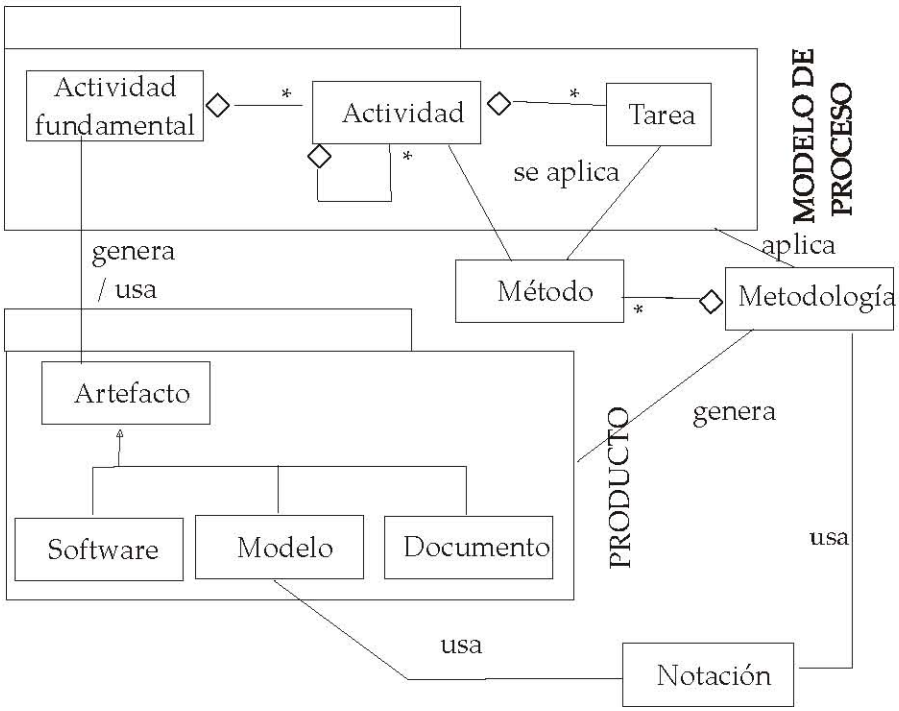


Figura 3.1 Conceptos utilizados en el modelo de proceso y sus relaciones.

Hay que tener clara la diferencia entre proceso y **ejecución del proceso o instancia de proceso**. La puesta en práctica de un proceso dependerá en gran medida de las características de proyecto. El proceso es un marco general que debe adaptarse al proyecto, los desarrolladores y la organización cliente.

El modo habitual de describir un proceso es en mediante flujos de trabajo o actividades fundamentales (workflows), actividades y artefactos. Una **actividad fundamental** es un conjunto de actividades relacionadas y lógicamente organizadas que generan un resultado observable o artefacto. Una **actividad** es una unidad tangible de trabajo que realiza un trabajador y que genera un resultado significativo en el contexto del proyecto. El propósito de una actividad es crear o modificar artefactos a partir otro conjunto de artefactos. Las actividades fundamentales se descomponen en actividades que pueden estar compuestas por otras actividades. Las actividades atómicas se les llaman **tareas**, que son pequeñas, acotadas y reusables. Un **artefacto** es una porción de información que se crea, modifica o utiliza durante el proceso. Un artefacto puede ser: un modelo compuesto, un modelo elemental (una clase, un caso de uso, un subsistema o una tarea de inferencia), un documento, código fuente, un ejecutable, etc. Un **modelo** de un sistema es una descripción o especificación de este sistema y de su entorno para un propósito determinado, es decir, una abstracción del sistema semánticamente cerrada. Los modelos son la representación explicita de una parte de la realidad percibida por un actor. Existe una relación directa entre los modelos y los niveles de abstracción, ya que se pueden tener tantos modelos como niveles de abstracción se definan en las tareas de modelado o puntos de vista del sistema.

En las metodologías de primera generación surgidas, incluso antes de la implantación del enfoque de modelado, es fácil confundir los conceptos, método, metodología y notación. Una **notación** es un conjunto de reglas gráficas o textuales para representar un modelo o artefacto, que terminará evolucionando hasta llegar al concepto de lenguaje. Un **método** es una técnica repetible para la resolución de un problema específico. Una **metodología,** con minúscula (ver sección 1.3 "Acotación del concepto de metodología", es una colección de métodos para la resolución de una clase de problemas, es decir, debe proporcionar una guía para ordenar las actividades de un equipo, dirigir las tareas, técnicas y herramientas a usar por cada desarrollador por separado y del equipo como un todo, además de, especificar los artefactos que deben

desarrollarse y ofrecer criterios para el control y la medición de los productos y actividades del proyecto.

3.2 Metamodelo y Artefactos InSCo

El modelo de proceso InSCo es el punto de partida para resolver el problema enunciado en esta tesis: la ausencia de una metodología integrada que sea aplicable a la construcción de sistemas software híbridos donde sea necesario contemplar técnicas basadas en conocimiento junto con técnicas algorítmicas en la solución definitiva.

El modelo de proceso propuesto permitirá la integración de metodologías del ámbito de la InCo y de la InSo, ofreciendo un punto de referencia para dar soporte lógico al proyecto de desarrollo del software de un sistema híbrido, y que integre bajo la misma descripción el sistema completo a construir. Se enriquece la InCo de un modelo de proceso que no tiene claramente definido, y se enriquece la InSo con la inclusión las técnicas para la obtención de información o gestión de imprecisión e incertidumbre, minimizando el coste del cambio metodológico, puesto que la propuesta InSCo recubre y extiende los métodos ya aplicados en ambas disciplinas. Este modelo de proceso recoge mejores características de las metodologías de ambas disciplinas que se ya se han enumerado en la sección 2.4. "Características deseables de la propuesta InSCo".

La definición de un modelo de proceso se realiza en dos fases, que según se ordenen generan dos tipos de modelos diferentes: modelos de proceso orientados a productos o artefactos y modelos de proceso orientados a actividades (Derniame, Kaba, & Warboys, 1999). Si se definen primero las actividades y después los productos generados se tienen modelos de proceso orientados a las actividades, que son más cercanos a la visión de los desarrolladores. Si por el contrario primero se definen los productos y después la actividades que son necesarias para obtenerlos se obtienen modelos de proceso orientados al producto o artefactos, que están más próximos al punto de vista del cliente. En los modelos de proceso de las metodologías soporte, encontramos ambos enfoques. Las metodologías estructuradas y RUP tienen un modelo de proceso orientado a las actividades, mientras que CK está orientada a los productos.

A raíz de nuestra experiencia en el desarrollo de software aplicando CK, hemos descubierto que cuando se aplica un modelo de proceso centrado en

productos se corre el riesgo de no saber cuál es la tarea siguiente o que se está haciendo en cada momento. Esta deficiencia lleva a que la planificación de los proyectos de desarrollo de software usando CK sea costosa, y que a los riesgos propios del desarrollo de SBC se añadan dificultades de planificación y gestión (Águila I. M., Túnez, Cañadas, Bosch, & Marín, 2001). No obstante, el resultado garantiza productos de más calidad, aunque en ocasiones fuera de plazo. Es decir, CK debería acercarse más al enfoque de actividades.

Por otra parte se ha producido un importante cambio en la InSo, que consiste en la transición desde las técnicas de producción de software orientadas al código hacia las técnicas de orientación a modelos o productos. Estos cambios vienen de la mano de MDA, que es una iniciativa de la OMG (Object Management Group) que defiende la definición de modelos como elementos de primera clase para el diseño e implementación del sistema (Kleppe, Warmer, & Bast, 2003). En MDA, las actividades más importantes pasan a ser las de modelado de los diferentes aspectos de un sistema, y la definición de trasformaciones de unos modelos a otros para que puedan ser automatizados.

Para el modelo de proceso InSCo se emplea un enfoque de modelado de proceso orientado a actividades definiendo un conjunto de actividades fundamentales, que mejora la puesta en práctica del proyecto. Estas actividades fundamentales se descomponen en actividades o tareas y generan y utilizan unos artefactos, de entre los que se destacan los modelos como productos primordiales en el desarrollo del sistema híbrido.

El metamodelo del modelo de proceso InSCo se muestra en la figura 3.2, siendo de igual nivel de importancia los modelos y las actividades. Se han definido seis actividades fundamentales que se describirán en las siguientes secciones de este capítulo. Cada modelo tiene un punto de vista y un nivel de abstracción en el que centrar su definición del sistema y recoge una descripción o especificación de un sistema y su entorno para algún propósito determinado.

Posteriormente, esta visión estática del modelo de proceso se debe recubrir con el enfoque dinámico donde se describan, iteraciones, ciclo de vida, hitos y planificación, es decir, lo que se han llamado actividades de protección o de gestión que también se pueden organizar en otros grupos de actividades, pero que quedan fuera del alcance de este trabajo (Ver sección 1.3 "Acotación del concepto de metodología").

El principal artefacto generado en cada una de las actividades fundamentales InSCo es un modelo o conjunto de modelos. Una cuestión importante que se suele presentar en el campo de la construcción de software es: ¿qué es un modelo? o mejor ¿cómo representamos un modelo? ¿con un conjunto de documentos, uno o varios diagramas, formularios que recojan y estructuren la información textual, un trozo de código en un lenguaje de programación? Un modelo suele estar representando por una combinación de gráficos y texto en algún lenguaje.

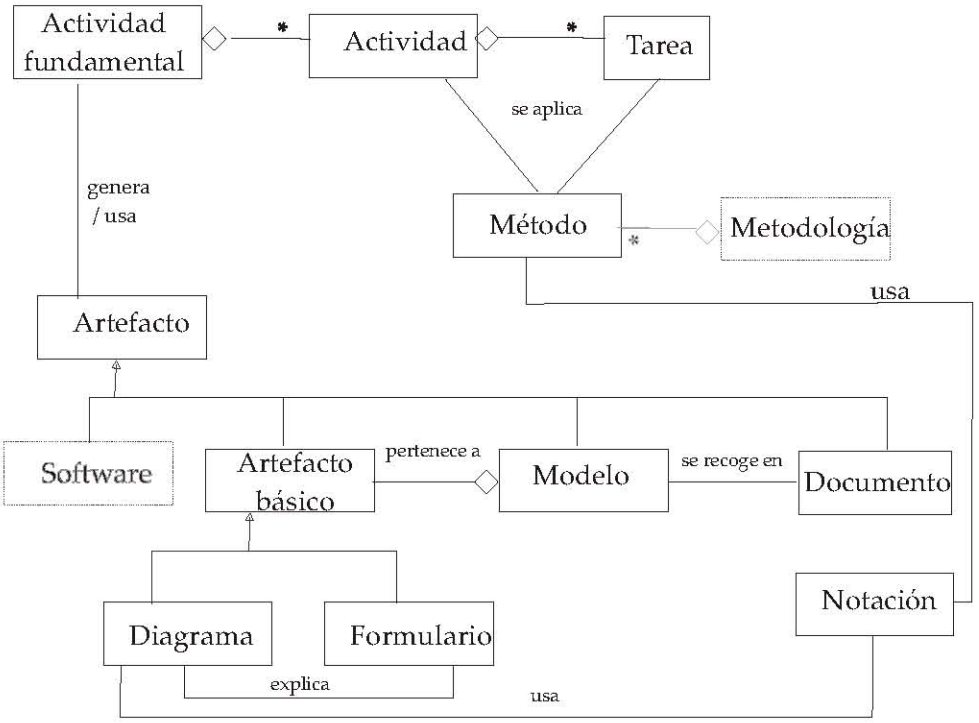


Figura 3.2 Metamodelo InSCo.

Un modelo habitualmente es un elemento contenedor y está compuesto de otro grupo de artefactos que podemos llamar artefactos básicos y, si es preciso, se organizarán jerárquicamente. InSCo propone dos tipos de artefactos básicos, los primeros son los diagramas o notaciones gráficas, que tienen grandes ventajas, sobre todo que permiten de un solo vistazo mostrar sistemas complejos. El inconveniente es que los diagramas más usados siempre se han utilizado ligados a un enfoque concreto o método de desarrollo. El segundo tipo de artefactos son los formularios o también llamados plantillas y constituyen una alternativa para acotar los problemas de falta de claridad en el uso del lenguaje natural

(Sommerville, Software Engineering: (Update) (8th Edition), 2006). Estas dos técnicas las utilizan muchos autores para la modelización de sistemas complejos (Yourdon, Análisis Estructurado Moderno, 1997), (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999), (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000) y (Duran & Jiménez, Metodología de análisis de requisitos de sistemas software. Versión 2.1, 2001) y además se suelen complementar. Ambos tipos de artefactos describen conceptos que dependen del nivel de abstracción del modelo al que pertenezcan. Así se pueden tener diagramas o formularios para describir el sistema completo o bien que describan una componente software a construir.

En InSCo cada modelo es el resultado de la agregación de otros artefactos básicos: diagramas y formularios. Estas relaciones entre diagramas y formularios se muestran en la figura 3.3. Por una parte cada formulario contiene un conjunto de campos de identificación y los demás campos dependerán del modelo concreto al que pertenezca, algunos de estos campos se pueden describir con un lenguaje gráfico, es decir, un diagrama. Por otra parte, cuando se necesita describir el diagrama que posee primitivas gráficas, se incluyen *n* formularios, uno para describir cada componente gráfica que incluya el diagrama, que se conocen como formularios múltiples.

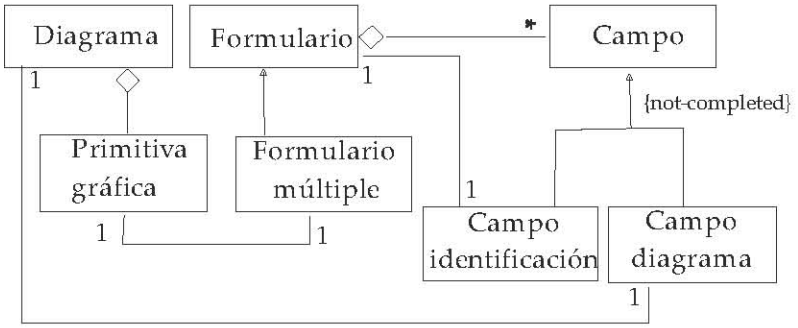


Figura 3.3 Extensión de metamodelo.

El otro tipo de artefactos contenedores que se han incluido en el modelo InSCo son los documentos, que también aparecen en el metamodelo de la figura 3.2. La elaboración de estos documentos es crucial para la gestión de los cambios y control de las versiones. Para la elaboración de los documentos se definieran formatos estándares que indican lo qué, como mínimo, debe recoger cada documento. La importancia de los documentos o especificaciones radica en que

son lo que algunas metodologías llaman productos o artefactos entregables, es decir, aquellas salidas del proceso que junto con el propio software se entregan oficialmente al cliente como parte del desarrollo en fechas previamente acordadas. Los documentos suponen el soporte físico de los modelos y serán descritos más adelante en esta tesis. Aunque en este punto nos podremos plantear la siguiente paradoja (similar al de la gallina y el huevo): *Son los documentos una declaración textual que contiene los modelos o son parte de los propios modelos.* Desde un punto de vista práctico no es necesario resolver la paradoja, se pueden considerar ambos tipos artefactos como equivalentes, es decir, los documentos dan soporte a los modelos para ser entregados a los clientes.

En resumen, el modelo de proceso InSCo que permitirá la integración de la InCo y de la InSo define un conjunto de actividades fundamentales y el artefacto esencial resultante de la aplicación de estas actividades son los modelos. Las actividades fundamentales se descomponen jerárquicamente y los artefactos se clasifican en tres tipos: modelos, documentos y artefactos básicos, siendo estos últimos dependientes del modelo al que pertenecen y se corresponden con los diagramas o formularios.

3.3 Descripción del modelo de proceso InSCo

El modelo de proceso propuesto, modelo InSCo, ofrece un punto de referencia del desarrollo de un proyecto de software híbrido, independientemente de que finalmente se construya con técnicas algorítmicas o heurísticas. El modelo InSCo es genérico y flexible puesto que permite configurar la ejecución del proceso de forma que se ajuste al propio proyecto o bien al equipo de desarrollo. Esta adaptabilidad se ha conseguido con la definición de las actividades fundamentales, que son generales pero a la vez tienen un claro objetivo, que no es más que obtener los modelos que describirán el sistema software híbrido y que deben concretarse a nivel de las tareas.

Así las actividades y modelos de InSCo se han agrupado en los cuatro paquetes mostrados en la figura 3.4. Por una parte, el nivel computacional más cercano a la solución y, por otra, los tres bloques del nivel conceptual que están dentro del mundo del problema. La visión unificada para las componentes basadas y no basadas en conocimiento del proyecto software se centrará en el nivel conceptual. Los modelos y actividades computacionales son dependientes

del enfoque de desarrollo y su descripción detallada queda fuera de este trabajo de tesis. Los tres paquetes conceptuales recogen algunas de las condiciones que debe cumplir el modelo de proceso y que se indicaron en el apartado 2.3.5 "Características deseables de la propuesta InSCo". De esta forma, los primeros pasos se deben centrar en estudiar el entorno donde se ubicará el sistema y el paquete Estudio del negocio dedica un grupo de actividades y modelos a definir el problema y delimitar de forma precisa el contexto de la solución propuesta. Los paquetes Desarrollo orientado al cliente y Desarrollo orientado al sistema han sido definidos para que los modelos conceptuales tengan el mínimo punto de conexión con los aspectos computacionales, restringiendo esta conexión a las componentes del paquete desarrollo orientado al sistema. Así, es en las actividades y modelos de este último paquete donde se inyecta el conocimiento del sistema híbrido, obteniendo una descripción de la estructura o arquitectura del sistema definitivo, preparada para traducida a una aplicación software.

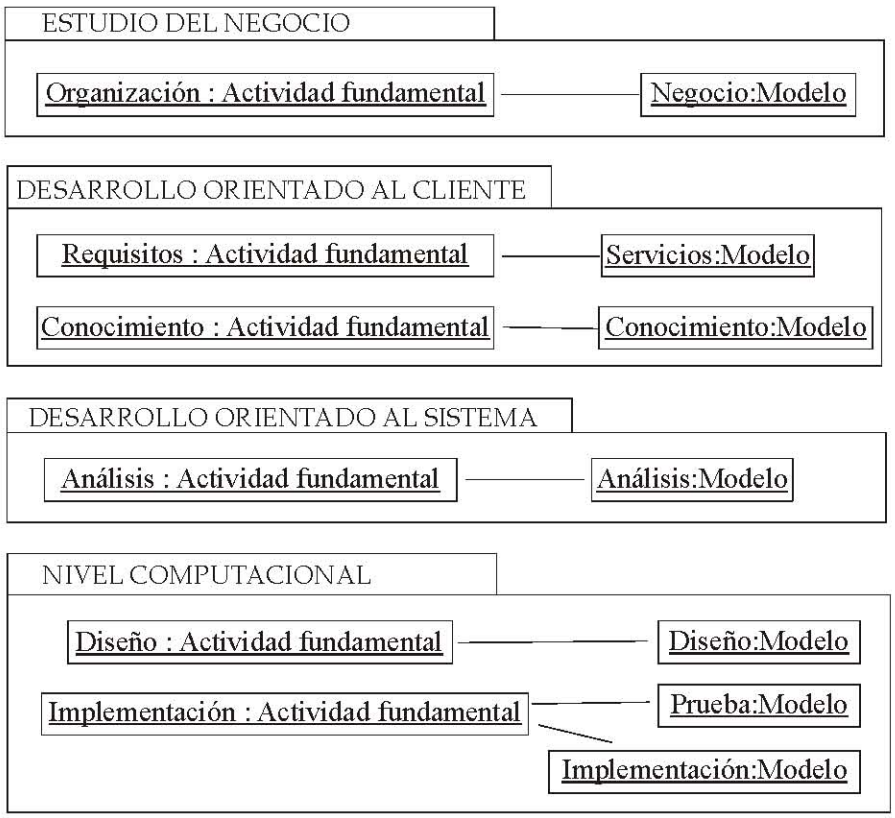


Figura 3.4 Modelo de proceso InSCo.

3.3.1 Actividades fundamentales InSCo

Se han definido seis actividades fundamentales a los que hay que añadir la evolución o mantenimiento: *Organización, Requisitos, Conocimiento, Análisis, Diseño* e *Implementación*. Los primeros grupos de actividades están dentro del mundo del problema o nivel conceptual y las últimas están más cercanas a la solución. Cada actividad fundamental se identifica con la primera letra de su nombre.

Organización - O: Su objetivo es comprender el entorno organizacional donde operará el sistema software. Generalmente los problemas aparecen cuando alguien percibe una dificultad u oportunidades de mejora. Por lo tanto, se debe identificar claramente a qué se pretende dar solución, así como cuáles son los beneficios, costes y los impactos sobre la organización que se van a producir con las aportaciones de este proyecto software.

Requisitos - R: Se estudian los requisitos o servicios que debe ofrecer el sistema desde el punto de vista del usuario. Posponiendo, la decisión de cuáles son los servicios basados en conocimiento y cuáles no. Así se estudian las interacciones entre servicios y se define una primera arquitectura del software a construir. Se especifica la comunicación y coordinación entre los distintos servicios y los conceptos y las tareas de transferencia de información desde el sistema a agentes externos o viceversa.

Conocimiento - C: Su propósito es explicar con detalle los tipos y estructuras del conocimiento utilizado para realizar una tarea de razonamiento. Proporciona una descripción del papel que juegan los diferentes componentes de conocimiento durante la resolución de problemas. Se distingue entre los distintos tipos de conocimiento siguiendo las pautas de CK, de acuerdo con los distintos papeles que puede jugar éste en el proceso de razonamiento: conocimiento de tareas/inferencias y conocimiento del dominio.

Análisis - A: Supone el refinamiento de los servicios inyectando sobre ellos el conocimiento, de forma que obtengamos una descripción que estructure el sistema definitivo. Lleva implícita la asignación de los servicios a la categoría de basados o no en conocimiento, el objetivo es definir una estructura para el sistema que sea a la vez robusta y cambiable, además de estar más cercana al lenguaje del desarrollador.

Diseño - D: Es la actividad fundamental donde se formulan los modelos del plano físico. Esta actividad fundamental se descompone en la definición de varios niveles de diseño o tipos de diseño: diseño de datos y/o clases, diseño de la interfaz de usuario, diseño procedimental, diseño de protocolos y diseño arquitectónico. Las actividades de diseño son fuertemente dependientes de si se trata de software basado en conocimiento o no. En caso de que uno de los subsistemas sea basado en conocimiento una tarea importante será decidir el formalismo para representar el conocimiento.

Implementación - I: Esta actividad tiene como propósito esencial implementar el sistema en términos de componentes, es decir, código fuente, guiones, archivos binarios, ejecutables, etc. Supone la traducción del diseño a uno o varios lenguajes de programación. Dentro de esta gran actividad sería necesario incluir el proceso de prueba, e implantación.

3.3.2 Modelos InSCo

Como resultado de las tareas fundamentales se generan siete modelos. A cada modelo además del nombre se le ha asignado un código de tres letras que se usará para referenciar el modelo y etiquetar sus componentes: *Modelo de NEGocio, de SERvicios, de CONocimiento, de ANÁlisis, de DISeño, de PRUeba* y *de IMPlementación.*

Negocio - NEG: En este modelo se recoge un glosario de términos y la definición de los conceptos relevantes para el dominio, incluyendo los stakeholders[11] o participantes. Se realiza la caracterización de la organización en términos de procesos de negocio, que suponen los objetivos de la organización. Además se identifican los problemas y oportunidades de mejora. NEG se emplea en la toma de decisiones sobre las posibles soluciones y su viabilidad, no siempre ligadas a la construcción de software.

Servicios – SER: Representa las necesidades o requisitos, entendidas como servicios que ofrecerá el sistema software. Se recoge la descripción detallada de

[11] El término en inglés stakeholder va más allá de los usuarios, incluye a todas las personas que puedan estar implicadas en el proceso de definición del sistema como son usuarios, clientes, analistas de mercado, la administración o los propios ingenieros del software, se ha optado por traducirlo como "participante".

estos servicios incluyendo la información que el sistema debe manejar para ofrecerlos, o si usan o no conocimiento, siempre que sea posible hacer esta preclasificación. Estos servicios deben ser priorizados y clasificados inicialmente aunque de forma no disjunta definiendo una primera versión de la arquitectura del sistema.

Conocimiento - CON: Es una descripción del comportamiento que debería mostrar el sistema cuando, supuestamente, emule el comportamiento de un experto humano. Es decir, el conocimiento que tiene un determinado actor y que es relevante para la consecución de una determinada tarea o servicio. Supone una descripción conceptual y declarativa del conocimiento aplicado en los procesos del negocio.

Análisis - ANA: Es una visión completa del sistema en el lenguaje de los programadores obteniendo una descripción de los requisitos y conocimiento que sea fácil de mantener y que nos ayude a dar estructura al sistema de su conjunto. Supone la unificación de requisitos y el conocimiento. Se produce la clasificación definitiva de los requisitos o servicios dentro del ámbito de diseño con técnicas algorítmicas o heurísticas, es decir, la definición completa de la arquitectura del sistema en cuanto a la asignación los requisitos al ámbito de los sistemas sean o no basados en conocimiento.

Diseño, de Implementación y de Prueba – DIS – IMP - PRU: Son los modelos computacionales del software del sistema híbrido, deben recoger los modelos de plano físico que, tal y como se ha indicado anteriormente, son altamente dependientes del tipo de componente software a desarrollar, del paradigma empleado en su desarrollo y de las plataformas hardware elegidas. La relación entre modelos computacionales y conceptual que debería ser *1:n*, para el mismo modelo conceptual se puede optar por distintos modelos computacionales y sus implementaciones, pero en la mayoría de los casos es *1:1* (Juristo & Acuña, 2002) y supone un problema complejo ajustar esta relación. El contenido y descripción exhaustiva de estos modelos está desarrollándose en trabajos paralelos dentro del grupo de investigación.

Cada modelo supone el refinamiento de uno o varios modelos de un nivel de abstracción superior, tal y como se muestra en la figura 3.5. El modelo de servicios SER y el de conocimiento CON son el refinamiento del modelo de negocio, donde se retoman los objetivos del negocio que van a ser resueltos con el software híbrido y se reescriben como servicios basados o no basados en

conocimiento, para finalmente ser totalmente definidos en el modelo de análisis ANA, donde los servicios son asignados definitivamente y de forma disjunta a ser diseñados en el campo de las técnicas de InCo o de InSo según sean o no basados en conocimiento.

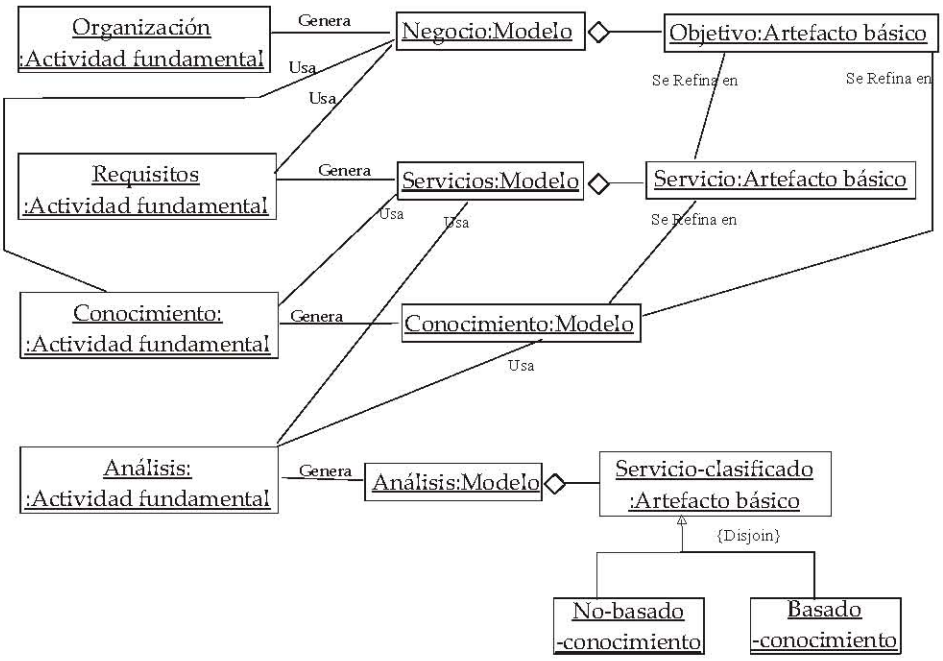


Figura 3.5 Evolución de los modelos InSCo.

El modelado se realiza por una parte, a nivel del sistema y se definen los objetivos ligados al negocio y, por la otra, a nivel del proyecto software, definiendo los servicios. Los servicios descritos en el lenguaje de los clientes se tratan en el paquete de desarrollo orientado al cliente, tanto desde la actividad fundamental Requisitos como desde Conocimiento y lo que es más importante, desde ambas a la vez. Estos mismos servicios ya en lenguaje del desarrollador se asignan de forma disjunta como basados o no en conocimiento en la actividad fundamental Análisis del paquete de desarrollo orientado al sistema. Supone un doble esfuerzo, pero que se verá recompensado evitando vueltas atrás para reasignar los servicios. Se pospone lo más posible la asignación para evitar errores.

Este refinamiento destaca aun más el carácter iterativo e incremental que se considera una característica deseable del modelo de proceso InSCo. En los capítulos siguientes se realizará una descripción detallada de estos modelos y de cómo obtenerlos.

3.3.3 Ciclo de vida. Hitos en el flujo de trabajo principal

El ciclo de vida se centra en el producto, definiendo los estados por los que evoluciona el software, es decir, los estados por los que pasa el producto desde el inicio de su construcción hasta que el software es operativo y, finalmente, hasta que es retirado (Saeki & Chang, 2002) y (Acuña, López, Mate, Ferre, & Chang, 2002).

A lo largo de la historia se han planteado diferentes aproximaciones con diferentes ciclos de vida para los SBC, pero el que se ha considerado mejor es el modelo en espiral de Boehm (Schreiber, et al., 1999) y (Boehm B. W., A spiral model of software development and enhancement, 1988). Pero no es una cuestión sencilla hacer concordar los aspectos gestión con los artefactos generados en una metodología y la evolución de estos artefactos, tal como se refleja en (Águila I. M., Túnez, Cañadas, Bosch, & Marín, 2001) donde se describe una propuesta para el ciclo de vida y la gestión de proyectos de desarrollo SBC bajo la metodología CK. Esta metodología soporte propone la realización paralela del desarrollo y de la gestión. El desarrollo consiste en la elaboración, evolución y validación de los modelos y la gestión introduce el planteamiento de la espiral, definiendo las regiones de tareas: *revisión, análisis de riesgos, planificación* y *monitorización*. Este paralelismo lleva a que no quede claro cuando cruzar ambos procesos y una de las razones que ha llevado a este trabajo de tesis.

Sin embargo, el modelo original de Boehm define entre tres y seis regiones de tareas, pero en todos los casos se propone una o dos regiones que se corresponden con el desarrollo, quedando este inmerso dentro de las tareas de gestión y control (Boehm B. W., Software risk management: principles and practices, 1991). Una de las principales aportaciones de Boehm es la definición de puntos de anclaje[12], que son hitos en el desarrollo que fijan puntos de control y estabilización del proyecto entre iteraciones. Boehm propuso tres puntos de anclaje para un proyecto software típico: *objetivos del ciclo de vida (lifecycle objectives* (LCO), revisión para asegurar que al menos una solución arquitectónica es válida desde el punto de vista organizacional, *arquitectura del ciclo de vida (lifecycle architecture)* (LCA), revisión de los artefactos y *inicio de capacidades*

[12] Traducción de anchor point milestones

operativas (initial operational capabilities) (IOC) (Boehm B. , Anchoring the software process, 1996).

Por otra parte, un ciclo de vida se centra en definir los estados por los que evolucionan el producto, en el caso de InSCo los modelos NEG, SER, CON y ANA, sus componentes y el propio software son el producto, tal como se indicó en la figura 3.2. Los estados por los que evolucione el producto vienen definidos por el grado de completitud, así se tienen para cada componente del producto al menos cinco estados: *Inicial, Identificado, Descrito, Definido, Validado* y *Completado.* Estos estados, ya definidos en la metodología CK, se pueden especializar dependiendo del componente y aparecen descritos en la tabla 3.1.

Tabla 3.1 Estados de las componentes producto.

Nombre del estado	**Descripción**
Inicial	Los trabajos no han comenzado
Identificado	Se han listado las características básicas del producto y sus componentes.
Descrito	Se ha obtenido la primera versión
Validado	El producto ha sido probado, verificado y validado
Completado	El trabajo ha finalizado de acuerdo con los criterios de aceptación

Los puntos de anclaje se fijan en función del estado de los modelos. En el nivel conceptual, los artefactos son: NEG, CON, SER y ANA. El primero está en el mundo de la organización con un alto nivel de abstracción y se trabaja con los Objetivos. Los otros tres se centran únicamente en el software híbrido a desarrollar y sus componentes básicos son los servicios y el conocimiento.

LCO se alcanza una vez que **NEG llega al estado completado.** Es decir cuando tenemos los objetivos, después de un número no determinado de ciclos en la espiral. Con NEG se toma la decisión de cuál es la solución arquitectónica válida para la organización y se acuerda entre todos los participantes en LCO. **LCA** se alcanza cuando **CON, SER y ANA se completan**, quedando IOC fuera del nivel conceptual. Lo importante a destacar es que estos tres modelos se van desarrollando de forma concurrente y con el objetivo claro de identificar, describir y validar los servicios que ofrecerá al usuario el sistema software híbrido, siendo en el modelo de análisis donde son asignados definitivamente a la categoría de basados o no en conocimiento.

En este proceso concurrente es donde radica la complejidad de proceso, se simultanean el estudio de las necesidades implicando al ingeniero del

conocimiento, al ingeniero del software y a los participantes del proyecto. La tarea de análisis es donde se debe llegar a un acuerdo de cuál es la configuración del sistema final en cuanto que es basado en conocimiento y que no; así como cual es el modelo de conocimiento que ha de emplearse de entre aquel que ya ha sido representado en CON.

3.4 Compatibilidad con otros modelos de proceso

Uno de los objetivos del modelo de proceso InSCo es el de ofrecer una cobertura a los principales enfoques metodológicos, así como ofrecer una extensión de los mismos para poder abordar el desarrollo de sistemas híbridos. En esta sección se verificará esta cobertura y supone la justificación de porque se han definido esas actividades fundamentales y no otras.

3.4.1 Cobertura de las metodologías soporte

A nivel de actividades fundamentales es donde planteamos las coberturas de los enfoques metodológicos más representativos. Describiremos las correspondencias que se muestran en la figura 3.6, revisando las actividades InSCo y como éstas recubren los demás modelos de proceso

3.4.1.1 Para la actividad Organización

Todos los enfoques metodológicos reflejan en más o menos medida la necesidad de tratar esta actividad relativa al estudio de la organización donde se ubicará el sistema. RUP contempla el modelado del dominio y de la organización de forma independiente y como tarea previa, ya que la metodología parte de un proyecto software acotado dentro de un sistema ya estudiado y válido (Eriksson & Penker, 1998). En el enfoque estructurado se aplica la ingeniería de sistemas con una fase de análisis y diseño más o menos detalladas, definiendo modelos del sistema que representan el nivel del sistema de información de la organización, incluyendo las componentes no software (Pressman, Ingeniería del Software - Un Enfoque Práctico, 2002) y (Sommerville, Software Engineering: (Update) (8th Edition), 2006).

CommonKADS propone que en el nivel de modelado del contexto se construya el modelo de organización. Este modelo tiene por objetivo analizar las principales características de la organización, para descubrir los problemas y las

oportunidades de uso de los sistemas basados en conocimiento, establecer su viabilidad y valorar su impacto en la organización. Metodologías del campo de la InRe postulan que para la adquisición de requisitos se hace necesario de obtener información sobre el dominio del problema y el sistema actual como primera tarea en el proceso de adquisición de los requisitos. MDA, define un modelo independiente de la computación CIM (Computational Independent Model) en el que se ubica este análisis de la organización.

La actividad fundamental Organización se ha definido puesto que *es necesario estudiar el contexto organizacional de forma independiente a la solución software en particular* para que el producto software resultante tenga la calidad esperada por el cliente, sobre todo cuando se trata de sistemas híbridos.

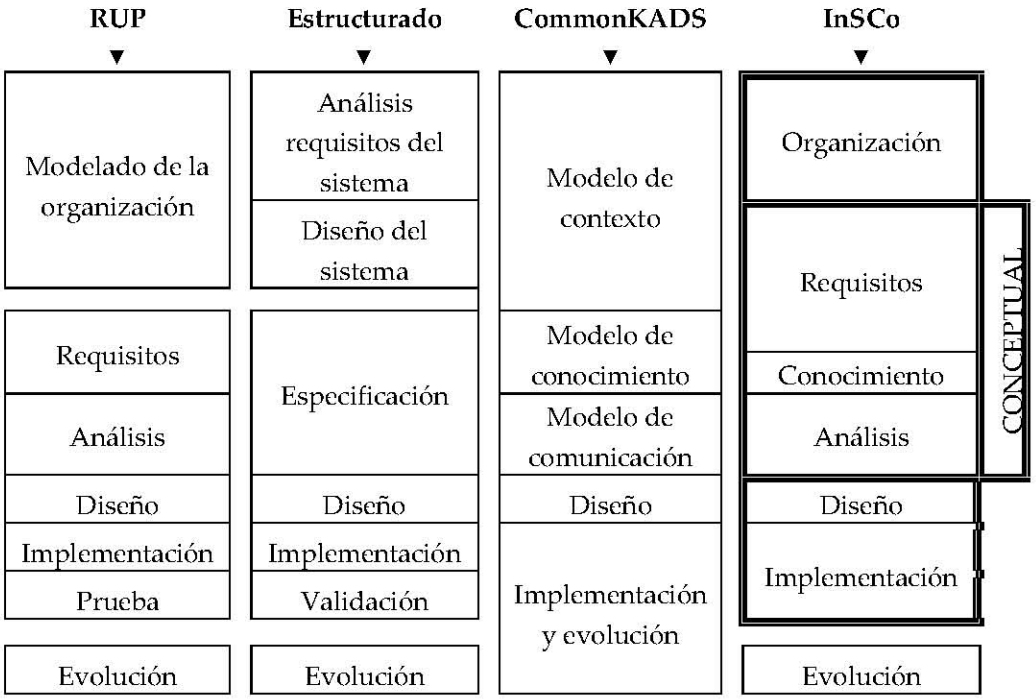


Figura 3.6 Correspondencia de actividades fundamentales sobre metodologías soporte.

3.4.1.2 Respecto a Conocimiento

En los sistemas híbridos el conocimiento es un recurso vital y distinguible del resto. Para esta tarea fundamental se han seguido las pautas marcadas por CK puesto que es la única de las metodologías soporte que recoge el conocimiento.

Se realiza el análisis del sistema en el nivel de conocimiento de Nevell, ofreciendo la posibilidad de especificar el problema a diferentes niveles de granularidad, con lo cual se potencia la idea de la reutilización de componentes genéricos de conocimiento y el uso de ontologías.

El artefacto generado en la actividad fundamental Conocimiento, CON, describe el conocimiento que tiene un determinado agente y que es relevante para la consecución de una determinada tarea, además de describir la estructura del mismo en función de su uso.

Sin embargo, existe una diferencia entre InSCo y CK en la interpretación del papel de este modelo de conocimiento. CK considera que el modelo conceptual del SBC a construir es el modelo de conocimiento, porque su planteamiento es que toda la funcionalidad a implementar es basada en conocimiento y no existen componentes que no usen este modelo de conocimiento/conceptual, mientras que para InSCo el modelo de conocimiento se define en el nivel de conocimiento y es básico aun sin SBC.

Para el modelo de proceso InSCo se define la actividad fundamental Conocimiento, puesto que el conocimiento, como realidad de la organización, debe inyectarse sobre el sistema software que se construya para resolver todos o algunos de los problemas detectados en esta organización. Es decir, *el conocimiento se debe modelar de forma declarativa sin considerar otro elemento, de forma independiente al software y un resultado más del proyecto.* (Águila, Sagrado, Túnez, & Orellana, Julio 2010 - Aceptado pendiente celebración) (5th International Conference of Software and Data Technologies – ICSOFT Julio - 2010).

3.4.1.3 De las actividades Requisitos y Análisis

El nivel conceptual en InSCo lo representan un grupo de actividades fundamentales: *Requisitos, Conocimiento* y *Análisis,* pero aparecen empaquetadas en dos bloques diferentes tal como se mostró en la figura 3.4 "Modelo de proceso InSCo" que son el desarrollo orientado al cliente y el desarrollo orientado al sistema

El término modelo conceptual surge del campo de las bases de datos y se utiliza para presentar los datos y relaciones que debe manejar el sistema de información sin tomar en cuenta la implementación. El ámbito de los modelos conceptuales se ha extendido y su objetivo es representar el dominio del discurso, es decir, el conjunto de datos relativos al problema a resolver y las

operaciones que afecta a esos datos (Juristo & Acuña, 2002). Este término se utiliza también con el mismo significado en la InCo y representa la modelización del conocimiento aplicado en un dominio particular, a nivel estático y dinámico (Speel, Schreiber, Joolingen, Heijst, & Beijer, 2001), en ambos casos de forma independiente de la implementación. La figura 3.7 muestra la partición de los modelos InSCo como conceptuales y no conceptuales.

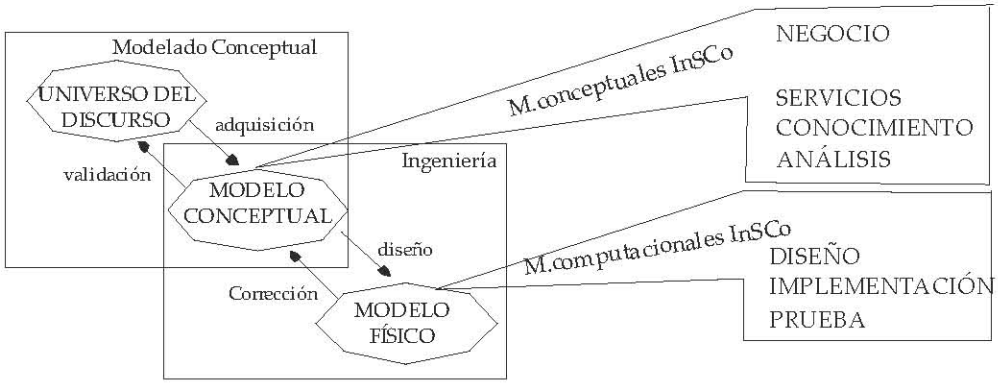


Figura 3.7 Partición de los modelos InSCo.

Para algunos enfoques metodológicos estos modelos conceptuales son de dos tipos, o lo que es lo mismo, el problema modelado se representa desde dos puntos de vista diferentes. Se definen dos actividades de modelado distintas, una más cercana al mundo real y otra más cercana al sistema de cara a intentar mantener el mínimo punto de conexión entre modelo conceptual y computacional.

Esta separación también la encontramos en la metodología soporte RUP donde se define el modelo de requisitos y el modelo de análisis; el primero está escrito en el lenguaje del cliente y supone una visión externa del sistema y el segundo se describe en el lenguaje del desarrollador y supone una visión interna del sistema (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999) que se desarrollan respectivamente en las actividades fundamentales de Requisitos y Análisis.

Otra de las propuestas define una separación entre dos grandes actividades: el modelado conceptual, el análisis orientado a la necesidad o al cliente y el análisis orientado al sistema software (Dieste, Development-Paradigm Independent Conceptual Models, 2001). La primera genera los llamados modelos conceptuales genéricos que en la siguiente actividad se acopla con una aproximación de desarrollo concreta.

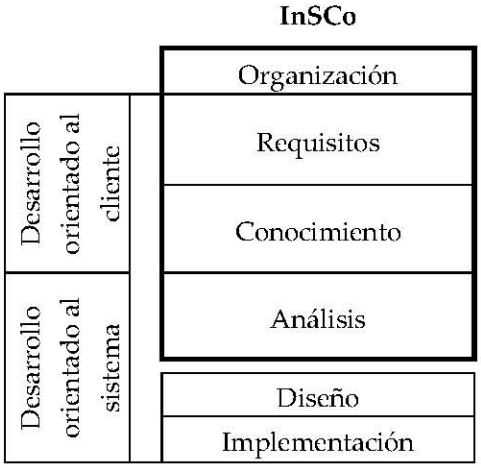


Figura 3.8 Partición de las actividades del nivel conceptual.

Otros trabajos en InRe distinguen entre la adquisición de requisitos y los llamados requisitos-C o del cliente y análisis de requisitos donde se obtienen los requisitos-D o del desarrollador para representar estos dos niveles en el modelado (Duran A. , 2000) y (Braude E. J., 2005).

En la propuesta InSCo se recogen todas estas consideraciones tal como se muestra en la figura 3.8. Por una parte, en la actividad fundamental Requisitos se tratan los servicios desde el punto de vista del cliente, se use o no conocimiento, que junto con la actividad fundamental Conocimiento, suponen la parte conceptual orientada al cliente. Y por otra, en Análisis se genera el modelo conceptual del software a construir representado también desde el punto de vista del desarrollador, o lo que se denomina desarrollo orientado al sistema.

Se definen de forma separada las dos actividades fundamentales Requisitos y Análisis, puesto que el nivel conceptual debe ser independiente de la implementación, pero siempre es necesario un punto de conexión con los aspectos computacionales y en sistemas híbridos con más razón para inyectar el conocimiento en el lugar oportuno.

3.4.1.4 En el nivel computacional: Diseño e implementación

Estas actividades que tratan los modelos son fuertemente dependientes del tipo de componente software a desarrollar y del paradigma empleado en su desarrollo. No obstante, si revisamos la tabla 3.2 donde se muestra la correspondencia entre los modelos computacionales de las metodologías soporte (Hoog, Martil, Wielinga, Taylor, Bright, & Velde, 1994), (Jacobson, Booch, &

Rumbaugh, The Unified Software Development Process, 1999) y(Pressman, Ingeniería del Software - Un Enfoque Práctico, 2002) y la propuesta InSCo, se observa que se ha seguido un planteamiento similar al propuesto por RUP, donde se separa entre las actividades de diseño e implementación. La primera conserva la estructura del sistema impuesta en las actividades anteriores y esquematiza la implementación y la segunda donde traduce el diseño a componentes o porciones de código.

Tabla 3.2 Modelos computacionales.

	InSCo	RUP	Estructurado	CommonKADS
Diseño	M. Diseño DIS	M. diseño M. despliegue	M. diseño arquitectónico, datos, procedimental, interfaz.	M. Diseño
Implementación	M. implementación M. prueba	M. Implementación M. Prueba	Casos de prueba	Implementación

3.4.2 Adaptación del modelo InSCo al estándar IEEE

Las actividades que son necesarias para desarrollar un proyecto software en todas sus perspectivas son de muy diferentes categorías, algunas están ligadas a la gestión del proyecto, otras a la coordinación de recursos, otras al desarrollo propiamente dicho. Se hace necesario hablar un mismo lenguaje, es decir, tener un marco común para el desarrollo de software y este marco común viene definido por los estándares. Éstos no son más que conjunto de criterios aprobados, documentados y disponibles para determinar la adecuación de una acción (estándar de proceso) o de un objeto (estándar de producto). En la InSo tradicionalmente se han desarrollado numerosos estándares en el ámbito de datos, de codificación, estructurales, de documentación, de proceso software, para otras actividades, como la validación o el control de calidad.

Los estándares relacionados con el proceso software se engloban en dos áreas: la definición de procesos estándar, destacando ISO 9000, IEEE 1074 (Institute of Electrical and Electronics Engineers Staff, 2006), ISO 12207 (Institute

of Electrical and Electronics Engineers Staff, 1998) y la definición de un método de evaluación del proceso y métodos de mejora del proceso como SEI's CMMI (Paulk, Curtis, Chrissis, & Weber, 1993). Para que el modelo de proceso InSCo sea efectivo debe cumplir con los estándares que están implantados actualmente, pero siempre teniendo en cuenta que este modelo de proceso no es más que un primer paso para la definición de una metodología más completa y los estándares en la mayoría de los casos son muy generales.

Familia de estándares ISO 9000 se definen para la gestión de la calidad de cualquier proceso de producción. De entre ellos destaca ISO 9001: *Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing* (ISO - International Organization for Standardization, 1994). Este estándar, que describe el sistema de calidad utilizado para mantener el desarrollo de un producto que implique diseño, es aplicable a cualquier proceso de producción: cojinetes, automóviles, TVs, equipamientos deportivos, etc. Está aceptado en más de 130 países y se está convirtiendo en el principal medio con el que los clientes pueden juzgar la competencia de un desarrollador de software, pero no propone un ciclo de vida (ni modelo de proceso), ni entre en muchos detalles y no la tomaremos en consideración.

El estándar IEEE 1074-1998: *Developing Software Life Cycle Processes* define las actividades que constituyen los procesos necesarios para el desarrollo y el mantenimiento de software, los procesos de gestión y soporte a lo largo de todo el ciclo de vida. Este estándar obliga a que sea necesario un ciclo de vida, pero no recomienda alguno en concreto.

Pero el estándar que aporta más luz es el ISO/IEC 12207:1995 (E) *Information technology–Software life cycle processes* (posteriormente adoptado por IEEE/EIA), porque emplea términos bien definidos e indica los procesos, actividades y tareas que se necesitan durante la adquisición de un sistema que contiene software (Piattini, Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software., 2003). Describe la arquitectura de los procesos de ciclo de vida del software pero no especifica los detalles de cómo implementar o realizar las actividades y tareas. El marco descrito por el estándar está diseñado para ser adaptado a cada organización y proyecto. En nuestro caso veremos cómo se adapta el modelo de proceso InSCo a este estándar.

La figura 3.9 recoge los procesos descritos en el estándar. Se definen cuatro grupos de procesos: de soporte, de la organización, de adaptación y principales.

Los **procesos de soporte** se aplican en cualquier momento del ciclo de vida, sirviendo de apoyo al resto y contribuyendo al éxito y calidad del proyecto software. Los **procesos de la organización** o procesos generales tienen por objetivo establecer, implementar y mejorar la organización en la gestión, formación del personal, mejora del proceso, etc. Éstos se realizan a nivel organizativo e independientemente del proyecto específico. Los **procesos de adaptación** permiten adaptar el estándar a cada proyecto y organización y consisten en la eliminación de procesos, actividades y tareas no aplicables. Por último, los **procesos principales** son los procesos útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software durante su ciclo de vida: compradores, suministradores, personal de desarrollo, operadores y personal de mantenimiento del software y en ellos se encaja InSCo.

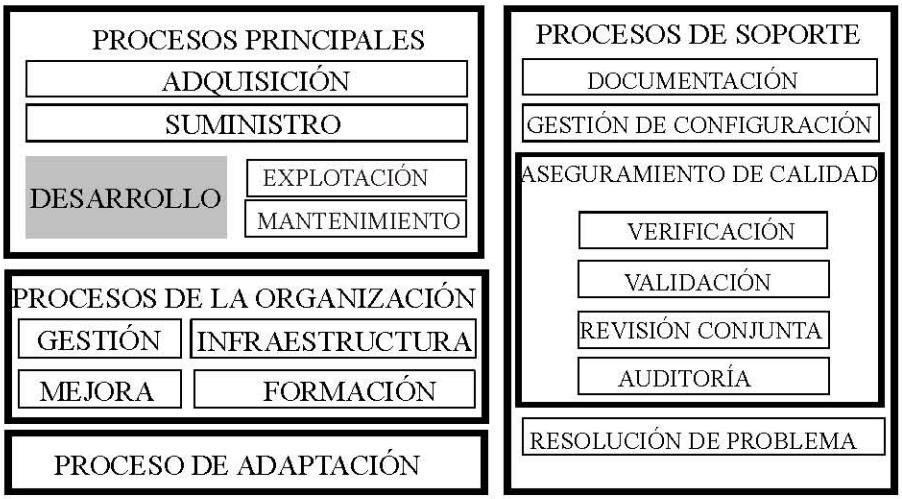


Figura 3.9 Tareas del estándar ISO/IEC 12207:1995.

Estos procesos principales se agrupan en cinco categorías: a) *Adquisición* que son las actividades y tareas que el comprador, el cliente o el usuario realizan para adquirir un sistema o producto (servicio) software, b) *Suministro* que son las actividades y tareas que realiza el suministrador del software, c) *Explotación* del software y del soporte del mismo, d) *Mantenimiento* que se ocupa de las tareas ligadas a los cambios a realizar por corrección, adaptación o mejora y, por último, e) *Desarrollo* que es la que se encarga de la construcción del software propiamente dicho como producto de ingeniería.

Tabla 3.3 Correspondencia del modelo de proceso InSCo con el estándar.

InSCo		**ESTANDAR**	
Organización		Análisis de Requisitos del Sistema	Se identifican y documentan: • funciones y capacidades • requisitos de seguridad y de la interacción hombre-máquina • interfaces del sistema • restricciones aplicables al diseño • requisitos de aceptación
		Diseño de la Arquitectura del Sistema	Identificación de arquitectura de alto nivel del sistema fijando los principales componentes hardware, software y las operaciones manuales. Se asignan los requisitos del sistema a dichos componentes
Conocimiento	CONCEPTUAL	Nada similar no es InSo	
Requisitos Análisis		Análisis de los Requisitos del Software	Se identifican y documentan: • especificaciones funcionales • interfaces externas seguridad y protección • datos a manejar y requisitos de la BD • requisitos de instalación, de aceptación y de mantenimiento
LIMITE DE LA TESIS			
Diseño	COMPUTACIONAL	Diseño de la Arquitectura del Software	Componentes principales del software Versión preliminar de los manuales de usuario Planificación de la integración del software
		Diseño Detallado del Software	Diseño detallado de cada componente software. Diseño detallado de las interfaces. Diseño detallado de la BD.
Implementación		Codificación y Prueba del Software	Se desarrollan los componentes software y las bases de datos Se prueban los componentes (prueba de unidad)
		Integración del Software	Se integran los componentes del software y se prueban según sea necesario
		Prueba del Software	De acuerdo con los requisitos de validación especificados para el software
No se reflejan en esta versión		Integración del Sistema	Se integra hardware, software y operaciones manuales
		Prueba del Sistema	Análoga a la del software, pero para el sistema
		Instalación del Software	En el entorno donde vaya a funcionar
		Soporte del proceso de Aceptación del Software	Finalmente, se debe dar apoyo a la revisión de aceptación y a la prueba del software por el comprador

El modelo de proceso InSCo es un modelo de desarrollo, el estándar ISO/IEC 12207:1995 propone las actividades cuya descripción se muestra en la tabla 3.3, además se muestra la correspondencia de tareas con el modelo de proceso InSCo. Destacamos en el nivel conceptual que el estándar no refleja el conocimiento y nuestra propuesta sí. Por otra parte, como ya indicamos en el apartado anterior, el modelo de proceso que proponemos separa en dos actividades el análisis de los requisitos propuesto en el estándar que se corresponden con el análisis orientado a la necesidad o Requisitos y el análisis orientado al sistema o Análisis.

3.5 Conclusiones del capítulo

- Se ha definido un modelo de proceso que permite una aproximación inicial al software a construir, independientemente de que finalmente se construya con técnicas algorítmicas o heurísticas y pudiendo ser el resultado basado o no en el paradigma orientado a objetos. El **modelo de proceso InSCo** supondrá un nuevo punto de convergencia hacia una metodología común para el desarrollo de los sistemas híbridos que **integra las metodologías** más utilizadas por InCo e InSo y que viene definido por las actividades fundamentales: **O**rganización, **R**equisitos, **C**onocimiento, **A**nálisis, **D**iseño e **I**mplementación. Estas actividades suponen una cobertura de las metodologías soporte y de los estándares IEEE.
- El principal artefacto generado durante la ejecución del modelo de proceso InSCo es un **conjunto de modelos**: Modelo de **Neg**ocio, Modelo de **Ser**vicios, Modelo de **Con**ocimiento, Modelo de **Aná**lisis, Modelo de **Dis**eño, Modelo de **Imp**lementación y Modelo de **Pru**eba.
- Las actividades fundamentales se descomponen en actividades y estas a su vez en tareas. Los modelos son una agregación de otros artefactos, que en el nivel conceptual son básicamente **los formularios y los diagramas.**
- El **conocimiento es un recurso crítico** para la organización y se modela independientemente de que la solución venga de la mano del software. El **modelo de conocimiento** representa el conocimiento de una empresa u organización y pueden emplearse como medio para compartir experiencias o con fines de formación.
- Las actividades del nivel conceptual se definen independientes de la implementación, pero existe un **punto de conexión con la implementación**

que es el modelo de **Anál**isis, de especial importancia porque es el punto donde se inyecta el conocimiento en las funcionalidades del software.

4 Estudio del negocio

En este capítulo describiremos la propuesta InSCo para la actividad fundamental **Organización,** en la que se genera el **Modelo de Negocio (NEG)** y el documento de definición del sistema (**DOC_DS**). Su objetivo principal la compresión y obtención de información sobre del contexto organizacional donde el sistema software híbrido desarrollará su funcionamiento.

La primera sección de este capítulo recoge la estructura y contenido del modelo de negocio y el documento de definición del sistema. Los resultados se ilustran sobre el ejemplo, ya introducido en el capítulo 1 "Motivación ya alcance de la tesis", y cuya especificación se recoge también en los apéndices. Los formularios relativos al ejemplo se describen con otro tipo de letra. Las secciones 4.2 "Flujo de trabajo" y 4.3 "Artefactos generaros por Organización" tratan respectivamente la descomposición en actividades o tareas de Organización y una breve descripción de las técnicas más apropiadas para la ejecución de esta actividad fundamental. Para un mayor detalle en la aplicación y uso de estas técnicas se debería consultar la bibliografía de referencia.

4.1 Independencia del estudio de la ORGANIZACIÓN

Antes de abordar el problema de construir software se debe considerar el estudio del sistema donde se ubicará. Un proyecto se realiza para una organización con sus propias políticas y procedimientos que a su vez están gobernados por políticas más amplias, así como por entornos económicos y sociales. Si el entorno organizacional no se comprende adecuadamente, el software a construir puede no cumplir las necesidades del negocio y ser rechazado por los usuarios y los directivos de la organización.

En el modelo de proceso InSCo se ha optado por seleccionar un enfoque cercano a las propuestas de CommonKADS, pero mejorándolo con el uso de técnicas de modelado más flexibles como escenarios o casos de uso del negocio. Este planteamiento se aproxima a las técnicas de gestión del conocimiento como punto de partida previo a la inyección de un sistema informático basado o no en conocimiento que mejore los problemas detectados durante esta actividad fundamental (Schreiber, et al., 1999).

Esta actividad fundamental adquiere importancia para sistemas grandes y complejos en cuanto a actividades y relaciones; sobre todo, si además, es necesario tomar en consideración y/o implementar el conocimiento aplicado por los miembros de la organización; es decir, gestionar y transferir el conocimiento de una organización.

Cuando se aborda un proyecto de construcción de software, se parte de uno o varios problemas que aparecen en una organización, es decir, alguien percibe una dificultad que en el estado de la situación pueden mejorarse. Estas mejoras pueden venir asociadas a la incorporación en la organización de sistemas software que colaboren en la realización de ciertas tareas o procesos que pone en práctica esta organización (National Aeronautics and Space Administration, 1995) y (Sage & Armstrong, 2000).

La figura 4.1 representa a la actividad fundamental **Organización**, empleando estereotipos para cada clase del metamodelo, como ya se ha mostrado en la figura 3.2 "Modelo de proceso InSCo". Así, la ejecución de esta actividad fundamental se articula a través de la ejecución de cuatro actividades, empleando cuatro tipos de formularios, para describir el propio proyecto, sus objetivos, los conceptos del dominio y los actores. Además, se apoya en la utilización de diagramas para describir los flujos de trabajo y el dominio, generando también como resultado el documento de definición del sistema.

Se debe identificar y recoger la información relativa a las características de la organización que puedan afectar a una posterior toma de decisiones sobre el proyecto de desarrollo de software (Hoog R. D., Benus, Metselaar, Vogler, & Menezes, 1994) y (Hoog R. D., Benus, Vogler, & Metselaar, 1996). Debiéndose identificar claramente a qué se pretende dar solución; así como cuáles son los beneficios, costes y los impactos sobre la organización que se van a producir con las aportaciones de un nuevo sistema software.

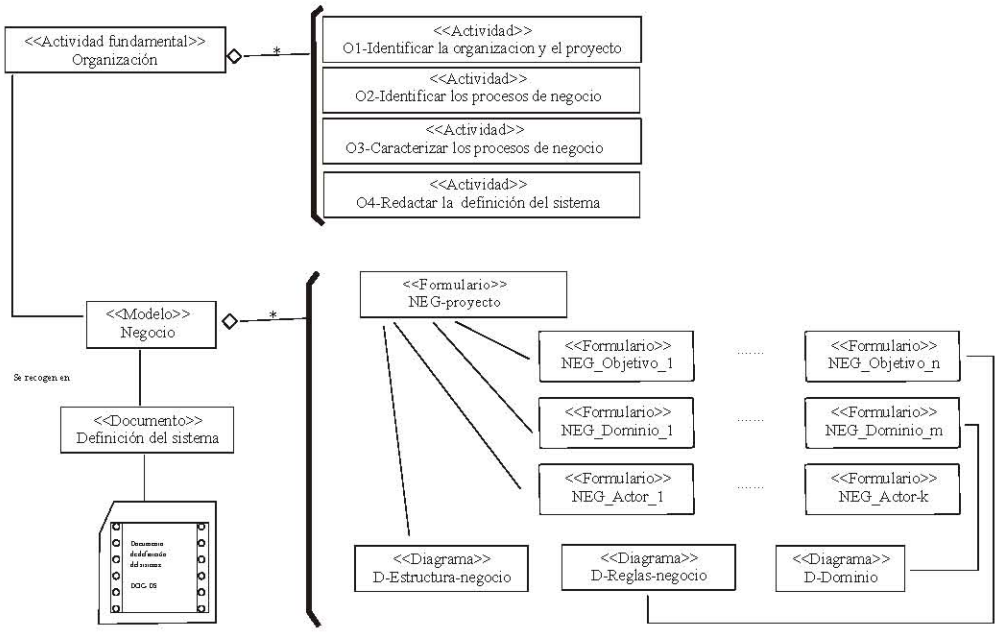


Figura 4.1 Descomposición en actividades de Organización y Modelo de Negocio.

La necesidad del desarrollo de un sistema de información debe buscarse fuera del propio sistema de información, en otras palabras, en el contexto en el que funcionará el sistema (Rolland & Prakash, From conceptual modelling to requirements engineering, 2000). El modelado del negocio es algo más que la simple determinación de los objetivos del sistema, extiende la cuestión: "¿Qué debe hacer el sistema?" con el punto de vista "¿Por qué el sistema es cómo es?".

Esta actividad ayudará a analizar la estructura organizativa de la empresa para poder sugerir mejoras que van más allá del simple análisis de sistemas al modo de la InSo. Se tendrá en cuenta que la Ingeniería del Negocio (Eriksson & Penker, 1998) y (International Council on System Engineering)[13] es una disciplina muy amplia que no se tratará aquí, sólo se considerarán los modelos del negocio como medio para obtener un firme conocimiento del contexto en el que operará el software a construir. Es decir, si la solución a los problemas viene dada por mejoras que no estén ligadas a la construcción de software, como puede ser una mejor gestión del conocimiento, mejora en algunos de los procesos o tareas de formación u otras similares, pararemos el modelado y el proyecto.

[13] INCOSE - www.incose.org/

Para conseguir sus metas, una empresa organiza su actividad por medio de un conjunto de procesos de negocio. Cada uno de ellos se caracteriza por una colección de datos del dominio que son producidos y manipulados mediante un conjunto de operaciones en las que ciertos agentes participan de acuerdo a un flujo de trabajo determinado, siendo el objetivo de la actividad fundamental Organización describirlos. Se estudia cada proceso del negocio, especificando sus datos, actividades organizadas en flujos de trabajo y los roles (o agentes). Además, se debe recoger información sobre el propio desarrollo del proyecto, que permitirá tomar decisiones acerca de la viabilidad del proyecto y del alcance de éste. Es importante generar como resultado de esta actividad el documento de definición del sistema (DOC_DS), que es un artefacto entregable donde se describe el modelo de negocio (NEG) y las decisiones de viabilidad.

Bajo estas consideraciones, InSCo define la actividad fundamental Organización y el modelo de Negocio, mostrados en la figura 4.1. Así su ejecución se articula a través de la ejecución de cuatro actividades: O1, O2, O3, O4 empleando cuatro tipos de formularios centrados en la descripción del propio proyecto (NEG_Proyecto), sus objetivos (NEG_Objetivo), los conceptos del dominio (NEG_Dominio) y los actores (NEG_Actor). Además, es recomendable la utilización de diagramas para describir los flujos de trabajo y el dominio (D_Reglas_del_negocio, D_Dominio), generando también como resultado el documento de definición del sistema (DOC_DS).

4.2 Flujo de trabajo

Una actividad fundamental se describe mediante su descomposición en otras actividades o tareas que emplean una serie de técnicas. Esta forma de descripción se utiliza en metodologías como MÉTRICA 3.0 (Consejo_Superior_de_Administración_Electrónica, 2010) [14] . La actividad fundamental Organización se descompone en las cuatro tareas principales descritas en la tabla 4.1.

La primera de las actividades, **identificar la organización y el proyecto** (O1), tiene por objetivo conocer la situación actual de la organización

[14] http://www.csi.map.es/csi/metrica3/

identificando sobre todo los problemas y áreas donde la aplicación de sistemas de información y, más específicamente, sistemas híbridos pueden apoyar a la estrategia corporativa definiendo posibles soluciones informáticas que den apoyo a los objetivos estratégicos de la organización.

Tabla 4.1 Actividades de Organización.

Organización
O1_Identificar la organización y el proyecto
O2_Identificar los procesos del negocio
O3_Caracterizar los procesos del negocio
O4_Redactar la definición del sistema

Participan activamente en esta tarea los responsables de los procesos de la organización con una visión estratégica y los profesionales de la InSCo (ingenieros del software e ingenieros del conocimiento) enriquecen esta visión con las aportaciones de los sistemas de información, gestión del conocimiento y tecnologías de la información y las comunicaciones.

Esta actividad fundamental lleva asociadas otras acciones relativas a la gestión del proyecto como el estudio de costes y beneficios; así como, la planificación de las actuaciones sobre la organización, pero en esta primera aproximación a un modelo de proceso InSCo no se profundizá en ellas.

Como resultado de esta actividad se genera el formulario NEG_Proyecto y la parte del DOC_DS relativa a la viabilidad y la planificación temporal. La definición del conjunto de procesos del negocio o reglas del negocio es una tarea crucial ya que define los límites del proceso de modelado posterior. La **identificación de estos procesos del negocio** (O2) se realiza a partir de los objetivos principales u objetivos estratégicos de la empresa. Teniendo en cuenta que estos objetivos van a ser muy complejos y de un nivel de abstracción muy alto, pueden ser descompuestos en un conjunto de subobjetivos más concretos que deberán cumplirse para conseguir el objetivo estratégico. Estos subobjetivos pueden a su vez ser descompuestos en otros, de manera que se defina una jerarquía de objetivos. Según resultados de la experimentación con dos o tres niveles de descomposición es suficiente (García, Ortín, Moros, Nicolás, & Toval, 2000).

Un apartado importante de cualquier proceso de negocio es la definición del conjunto conceptos importantes para la organización, también llamado modelado del dominio (Eriksson & Penker, 1998). Representan cosas que existen

o eventos que suceden en el entorno de la organización. Este modelo del dominio es un glosario de términos que ayudan a los usuarios, clientes y desarrolladores a utilizar un vocabulario común y viene a representar una ontología o parte de ella. También es necesario identificar a los actores que participan en cada proceso y con ambos elementos obtenemos la **caracterización de cada uno de los proceso de negocio** identificados (O3), NEG_Dominio y NEG_Actor.

Durante el desarrollo de estas tres tareas se genera el modelo de negocio que recoge los procesos del negocio a partir de la definición de los objetivos estratégicos de la empresa y la definición de los conceptos importantes para la organización. Puede realizarse una preasignación de cada uno de los procesos de negocio como basado o no en conocimiento (la clave es que esta preasignación se puede corregir y ajustar a lo largo del desarrollo)

A partir de la información recopilada y estructurada en las actividades O1 a O3 se **redacta el documento de definición del sistema** (O4) que supone la conclusión de la actividad fundamental Organización.

4.2.1 Técnicas recomendadas

Desde el punto de vista del desarrollo y dejando aparte las tareas de protección o gestión, la actividad fundamental Organización está centrada en la obtención de información. Por tanto, las técnicas aplicables son básicamente las relacionadas con la recogida y análisis de información. Muchas de estas técnicas también se recomendarán en las actividades de la actividad fundamental Servicios que se describirá en el siguiente capítulo.

Las técnicas más habituales se enclavan en tres grandes categorías: **observación, sesiones de trabajo** y **análisis de protocolos**. Dentro de la primera nos encontramos con técnicas donde, en las labores de obtención de información, el papel del usuario es pasivo, como la visita del entorno de trabajo, el estudio de documentación, formularios y archivos existentes o la inmersión en el negocio del cliente (Goguen, 1993) y (Whitten, 1996).

Las sesiones de trabajo agrupan un conjunto de técnicas diverso en las que el usuario debe participar en más o menos medida del proceso de obtención de información. De entre ellas destacamos las entrevistas, Joint Application Development (JAD) o desarrollo conjunto de aplicaciones, brainstorming o tormenta de ideas.

Las entrevistas son la técnica de obtención información más utilizada ya que son una de las formas de comunicación más naturales entre personas. A la hora de aplicar las entrevistas se distinguen tres fases. La fase de *preparación*, donde se seleccionan los individuos a entrevistar, se fijan objetivos y se genera un guión o lista de preguntas que supone una planificación de la entrevista. La fase de *realización* donde se desarrolla el guión de la entrevista. En la fase de *análisis* se reorganiza la información para contrastarla con otras entrevistas o fuentes de información (Piattini, Calvo, Joaquín, & Férnandez, 1996).

La técnica **JAD** fue desarrollada por IBM en 1977 y constituye una alternativa a las entrevistas individuales. Se desarrolla a lo largo de un conjunto de reuniones en grupo durante un periodo de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo. En comparación con las entrevistas individuales ahorra tiempo al evitar que las opiniones de los clientes se contrasten por separado, permite que todo el grupo revise la documentación generada e implica más a los clientes y usuarios en el desarrollo (Wood & Silver, 1995).

La **tormenta de ideas** es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Consiste en reuniones de cuatro a diez participantes. En una primera fase se sugieren todo tipo de ideas sin juzgar su validez para en una segunda fase, realizar un análisis detallado de cada propuesta (Osborn, 1957).

La última categoría de técnicas son las que llamamos de **análisis de protocolos**. Tienen por objetivo responder a la pregunta "¿Qué hay que hacer para?" y dependiendo del momento del modelado en el que se aplique se obtendrá un resultado u otro. Si estamos en el modelo del negocio se aplican estás técnicas para obtener información acerca de *qué_hay_que_hacer_para* que la organización cubra sus objetivos estratégicos, es decir, identificar los procesos del negocio y, por otra parte, *qué_hay_que_hacer_para* poner en práctica estos procesos del negocio, es decir, caracterizar los procesos del negocio. Si estuviésemos en el modelado del conocimiento, estas técnicas nos proporcionarían el conocimiento sobre cómo se descompone una tarea de alto nivel en varias subtareas y el conocimiento sobre inferencias que describen los procesos primitivos de razonamiento que tienen lugar en una aplicación. Si estamos modelando

servicios obtendremos los requisitos desde el punto de vista del cliente, como veremos en el siguiente capítulo.

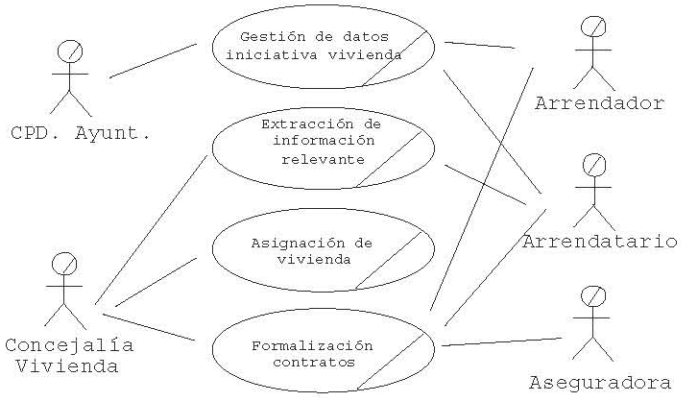
La técnica que tradicionalmente se ha empleado en la definición de protocolos es la descomposición funcional o aproximación basada en objetivos, bien sea utilizando diagramas de flujo de datos o diagramas de actividad de UML. Se ha utilizado en el desarrollo de software tanto para la InSo y como para la InCo, de hecho el modelo de tareas o procesos de CommonKADS se desarrolla usando esta técnica. Pero como CK apuntaba *"el modelo de tareas es difícil de obtener sin observar como la gente hace su trabajo"* (Waern & Gala, The CommonKADS agent model, 1993). Esto nos conduce a otra técnica para determinar *qué_hay_que_hacer_para*, los escenarios (Rolland, Souveyet, & Achour, Guiding goal modeling using scenarios, 1998), que se recogen en los casos de uso.

Los casos de uso fueron propuestos inicialmente en (Jacobson, Christerson, Jonsson, & Övergaard, 1992) y actualmente forma parte de la propuesta de UML (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual (2nd Edition), 2004). Un caso de uso es la descripción de una secuencia de interacciones entre el sistema y uno o más actores en la que se considera al sistema como una caja negra y en la que la que los actores obtienen resultados observables. Los actores son personas u otros sistemas que interactúan con el sistema que se está describiendo.

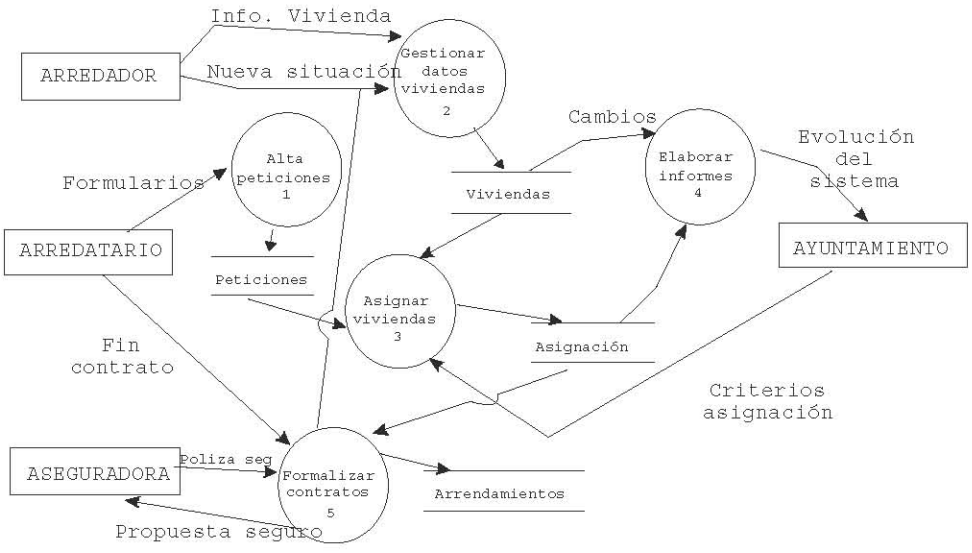
A pesar de ser una técnica ampliamente aceptada, existen múltiples propuestas para su utilización concreta (Cockburn, Writing Effective Use Cases, 2000), entre otras cosas porque se puede aplicar desde el estudio de la organización hasta el estudio de la interacción hombre máquina (Rosenberg & Scott, 1999), (Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, 2008) y (Schneider & Winters, 1998).

Para la descripción concreta de los casos de uso se pueden usar distintos elementos como diagramas de secuencia, formularios o notaciones formales. Los casos de uso tienen una representación gráfica en los denominados diagramas de casos de uso (Booch, Rumbaugh, & Jacobson, Unified Modeling Language User Guide, The (2nd Edition), 2005). La participación de los actores en los casos de uso se indica por una línea entre el actor y el caso de uso. Los diagramas de casos de uso sirven para proporcionar una visión global del conjunto de casos de uso de un sistema así como de los actores y los casos de uso en los que éstos intervienen.

Un ejemplo de utilización de esta técnica basada en escenarios para la determinación de los objetivos estratégicos se representa en el diagrama de casos de uso de la figura 4.2 (a) para el proyecto Arrendamientos. Si se emplea una técnica de obtención de información basada en objetivos y descomposición funcional el resultado para el proyecto Arrendamiento será un diagrama de flujo de datos similar al de la figura 4. 2 (b)



Objetivos estrategicos identificados como casos de uso (a)



Objetivos estrategicos identificados como procesos del DFD (b)

Figura 4.2 D_Reglas_de_Negocio para Arrendamientos.

Pero como ya se ha indicado repetidamente la utilización de una técnica u otra debe ser independiente de las tareas y artefactos generados durante la

puesta en práctica de una actividad fundamental, en este caso los artefactos son el diagrama D_Reglas_del_negocio y los objetivos estratégicos de la organización, descritos en los formularios NEG_Objetivo, cuyo contenido se tratará en la siguiente sección.

4.3 Artefactos generados por ORGANIZACIÓN

En esta actividad fundamental se generan diversos tipos de artefactos. Por una parte, está el modelo del Negocio y, por otra, el documento de definición del sistema. El nivel de abstracción del modelo NEG es el de la organización donde se ubicará el sistema software híbrido a desarrollar. Toda organización para conseguir sus metas estructura su funcionamiento por medio de un conjunto de procesos del negocio. Cada uno de ellos se caracteriza por una colección de datos que son producidos y manipulados mediante un conjunto de acciones que necesitan de una serie de recursos, en las que ciertos agentes (por ejemplo, trabajadores o departamentos con una cierta estructura) participan de acuerdo a un flujo de trabajo determinado.

A continuación describimos NEG, tanto su estructura como el contenido de cada formulario, así como el documento de definición del sistema.

4.3.1 Estructura del modelo de negocio

En este modelo hay cuatro tipos de formularios agrupados en dos categorías: los formularios únicos, sólo uno de cada tipo y los múltiples, que tenemos uno por cada elemento a describir. De la primera categoría es el formulario ligado con el proyecto **NEG_Proyecto,** donde identificamos los problemas y oportunidades de la propuesta. Este formulario es el punto de partida del proyecto y documento de definición del sistema representa el punto final de la realización de modelo del negocio. La figura 4.1 se muestran las relaciones entre los formularios del modelo de negocio.

El formulario **NEG_Actor** indica las personas o los papeles de las personas implicadas en el proyecto, que puede ir desde los beneficiarios del sistema a los usuarios o quienes toman decisiones. Se completa un formulario para cada actor del negocio.

Los formularios donde se recogen los procesos de negocio son **NEG_Objetivo**. Aquí se reflejan los objetivos estratégicos de la organización y se pueden organizar en subobjetivos. El planteamiento es ir refinando los objetivos hasta llegar a definirlos como servicios, descritos en el siguiente capítulo. La primera aparición de estos objetivos se refleja un diagrama en el formulario NEG_Proyecto, D_Reglas_del_negocio y se refina con el conjunto formularios NEG_Objetivo.

El mismo refinamiento se realiza con los conceptos ligados al dominio. Estos conceptos inicialmente definidos en **NEG_Proyecto** se refinan y describen el conjunto de formularios correspondientes **NEG_Dominio,** que viene a representar los objetos, eventos o conceptos del dominio que se emplean para la consecución de los objetivos estratégicos de la empresa.

A partir de NEG se diseñan y planifican los cambios organizativos que resolverán los problemas detectados, fijando el alcance del sistema a construir, siempre que sea viable y realizando la asignación de recursos para la ejecución del proyecto.

Las conclusiones obtenidas se plasman en el otro artefacto de tipo contenedor generado en esta actividad fundamental que es el **documento de definición del sistema DOC_DS**. Este documento es el equivalente a la conclusión del análisis de sistemas InSo. Recopila los resultados de la ejecución de Organización fijando el alcance de las siguientes actividades de desarrollo. Un apartado importante de este documento es el informe de viabilidad, donde se recomienda si es conveniente seguir con el desarrollo de las soluciones propuestas y en qué medida.

El detalle de estos artefactos se muestra en los siguientes apartados empleando como ejemplo el descrito en la sección 1.6 "Casos de estudio y herramienta InSCo Requisite" que se que se identifican con el literal "Ejemplo".

4.3.2 Contenido del modelo de negocio, NEG

NEG está compuesto por una serie de formularios y diagramas. La figura 4.1 recoge los tres diagramas del modelo y los formularios que describen el proyecto, los objetivos, los elementos del dominio y los actores. Una de las grandes ventajas en el uso de formularios es la tolerancia ante la incompletitud, los formularios pueden tener agujeros que se rellenaran en sucesivas iteraciones

hasta dar por bueno en modelo, es decir, alcanzar el punto de anclaje LCO (ver apartado 3.3.3 "Ciclo de vida. Hitos en el flujo de trabajo principal").

Todos los formularios deben incluir una serie de campos de identificación que permiten mantener un catálogo ordenado y que se emplearán en el control de versiones y cambios. Estos campos de identificación son: la versión, los autores y las fuentes, así como una fila para los comentarios. En la tabla 4.2 se muestra una descripción de cada uno de estos campos. Además, si se trata de formularios múltiples debemos utilizar un identificador para cada uno de ellos, tal como se muestra en el Ejemplo 2.

Tabla 4.2 Descripción de los campos comunes.

Versión	Contiene el número y la fecha de la versión actual, es importante para poder gestionar distintas versiones y el control de cambios.
Autores, Fuentes	Estos campos contienen el nombre y la organización de los autores (normalmente desarrolladores) y de las fuentes (clientes o usuarios), de la versión actual, de forma que la rastreabilidad pueda llegar hasta las personas que propusieron la información contenida en el formulario. Además puesto que la elaboración de los formularios se formula como un proceso de refinamiento y profundización, este campo indica donde seguir profundizando.
Comentarios	Cualquier otra información que no encaje en los campos del formulario puede recogerse en este apartado.

Ejemplo 2 Campos comunes de Arrendamientos.

nombre de formulario-id	**nombre descriptivo**
Versión	Número de la versión actual (fecha de la versión actual)
Autores	autor de la versión actual (organización del autor). . .
Fuentes	fuente de la versión actual (organización de la fuente)
.....	.....
Comentarios	comentarios adicionales

4.3.2.1 Formulario NEG_Proyecto

Permite definir el contexto organizacional del proyecto, identificando los problemas y oportunidades y recopilando la información sobre la organización que afectará al proyecto. Además de los campos de identificación, incluye los campos descritos en la tabla 4.3. En el Ejemplo 3 se puede ver la instanciación de este formulario para el ejemplo Arrendamientos.

Los campos estructura, proceso y conceptos del formulario NEG_Proyecto se describen mejor gráficamente, tal como se muestra para el ejemplo Arrendamientos en las figuras 4.2, 4.3 y 4.4

Tabla 4.3 Descripción de los campos del formulario NEG_Proyecto.

Descripción del proyecto	Incluye una descripción breve del objetivo global del proyecto asignando un nombre o un identificador al proyecto completo.
Problemas y oportunidades	Constituyen una lista de los problemas o necesidades organizacionales que se consideran más o menos prioritarias y que deberían ser resueltas por la solución software propuesta.
Contexto organizacional	Referencia a los factores del entorno donde se desenvuelve la organización que pueden influenciar en los efectos del sistema software.
Soluciones	Constituyen las propuestas que han sido o van a ser desarrollados para resolver el problema en cuestión y cubrir las necesidades planteadas.
Estructura	Representa la descripción de la arquitectura de la organización fijando las relaciones entre secciones y personal o elementos externos de la empresa. Lo habitual es describir este campo mediante un diagrama que de forma gráfica represente las relaciones en la empresa, D_Estructura.
Proceso,	Representa los flujos de trabajo principales de la organización descritos en los formularios objetivos. Describe los procesos de negocio, es decir, los objetivos estratégicos y sus relaciones. Normalmente se emplea un diagrama, aunque se podría utilizar una lista, pudiendo ser un diagrama de actividades, un diagrama de flujo de datos o un diagrama de casos de uso del negocio, dependiendo de la aproximación que utilicemos a la hora de determinar cuáles son los objetivos estratégicos de la organización, D_Reglas_del_negocio.
Personas	Constituye la primera aproximación a los agentes. Intenta reflejar tanto el personal como su función dentro de la organización, sin tener en cuenta el conocimiento y potencial que aportan. Se trata de considerar al personal como agentes que desarrollan una función dentro de la organización en tanto tengan alguna relevancia para la solución propuesta. Se puede usar una lista o un diagrama donde se representen las personas o usar el propio D_Estructura.

Conceptos	Reflejan los elementos claves en el dominio de la información que serán descritos con mayor detalle en los formularios del dominio. Al igual que con los objetivos, se puede emplear un diagrama que los represente, como puede ser un diagrama de clases un diagrama entidad relación o bien usar notación Backus-Naur Form (BNF).
Recursos	Describe los recursos utilizados por los procesos que pueden ser o bien sistemas de información y otros recursos computacionales, hardware como software o bien equipamiento y material o tecnología, patentes, etc.
Conocimiento	Intenta representar el conocimiento de alto nivel que puede tener influencia en la definición del problema o de las soluciones. Aquí se describe el tipo de conocimiento que está presente, cómo se mantiene y cuán importante es para la organización. Representa un recurso especial explotado dentro de la organización. Podría incluir la identificación de las ontologías a aplicar en el proyecto.

Ejemplo 3 Formulario NEG_Proyecto para Arrendamientos.

NEG_Proyecto	
Descripción del proyecto	Gestión del sistema de asignación y seguimiento de viviendas arrendadas dependiente de la administración local.
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local: Servicios sociales, Área de apoyo a la juventud Ley de Arrendamientos Urbanos (LAU, 1994) Compañías aseguradoras
Problemas y oportunidades	La asignación y valoración consume tiempo Imposibilidad de tramitar casos urgentes Falta de conocimiento del estado real del parque de viviendas en alquiler Gestión administrativa del arrendamiento Gestión administrativa de los seguros asociados Peritaciones de las viviendas
Contexto organizacional	Misión, objetivos generales de la organización: Facilitar el acceso a viviendas sobre todo a jóvenes siendo ellos mismo los que analicen sus posibilidades Abrir el mercado de viviendas en alquiler dando garantías a los arrendadores
	Factores externos importantes con los que se relaciona la organización: - LAU - Mercado inmobiliario externo (Compra y alquiler) - Variaciones de precios
	Estrategia de la organización: Servicio público (arrendatarios y arrendadores)

	Escala de valores por la que se rige: Bienestar socio-económico del municipal
Soluciones	Base de datos de viviendas Análisis del mercado para ajustar las tablas SBC par la asignación de viviendas Software para el control del estado de los arrendamientos
Estructura	Figura 4.4
Proceso	Figura 4.3
Personas	Arrendatarios Personal evaluador Peritos de las compañías Administrativos del ayuntamiento
Conceptos	Figura 4.5
Recursos	Se dispone de 2 personas para la gestión del trabajo. Un presupuesto inicial de 20.000 Euros Despacho en la segunda planta de Ayuntamiento completamente dotado.
Conocimiento	Criterios de asignación

4.3.2.2 Formulario NEG_Objetivo_n

Este formulario permite caracterizar cada objetivo estratégico de la organización. Incluye los campos que se describen en la tabla 4.4.

Tabla 4.4 Descripción de los campos de NEG_Objetivo_n.

Descripción	Cada objetivo debe identificarse por un código único y un nombre descriptivo. Además, debe ser posible describirlo con un enunciado breve.
Subobjetivos	En este campo pueden indicarse los subobjetivos que dependen del objetivo que se está describiendo. En sistemas complejos puede ser necesario establecer una jerarquía de objetivos. En caso de que esto no sea necesario puede ignorarse este campo.
Realizado por	Referencia a los agentes que realizan o se ven implicados en la tarea, bien sea de forma activa o pasiva.
¿Intensivo en conocimiento?	Indica si este objetivo, en el estado actual de recolección de información, es intensivo en conocimiento, aunque puede cambiar su estado cuando se profundice en el estudio del proyecto.
Importancia	Indica la importancia del cumplimiento del objetivo para los clientes y usuarios. Se puede asignar un valor numérico o alguna expresión enumerada como vital, importante o quedaría bien. En el caso de que no se haya establecido aún la importancia se puede indicar que está por determinar.

Urgencia	Indica la urgencia del cumplimiento del objetivo para los clientes y usuarios en el supuesto caso de un desarrollo incremental. Como en el caso anterior, se puede asignar un valor numérico o una expresión enumerada como inmediatamente, hay presión o puede esperar.

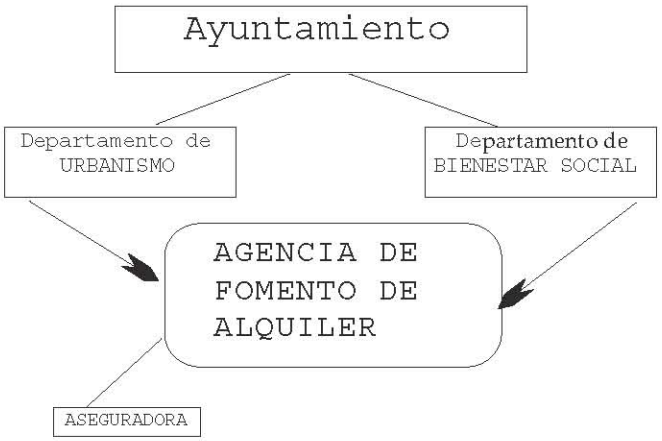


Figura 4.3 D_Estructura de Arrendamientos.

Para Arrendamientos cada elemento (caso de uso o proceso) del diagrama D_Reglas_del_negocio de NEG_Proyecto genera un formulario NEG_Objetivo-i, pues se trata de un proyecto simple que no necesita subobjetivos. Si los tuviese se generaría un nuevo NEG_Objetivo_n para cada subobjetivo.

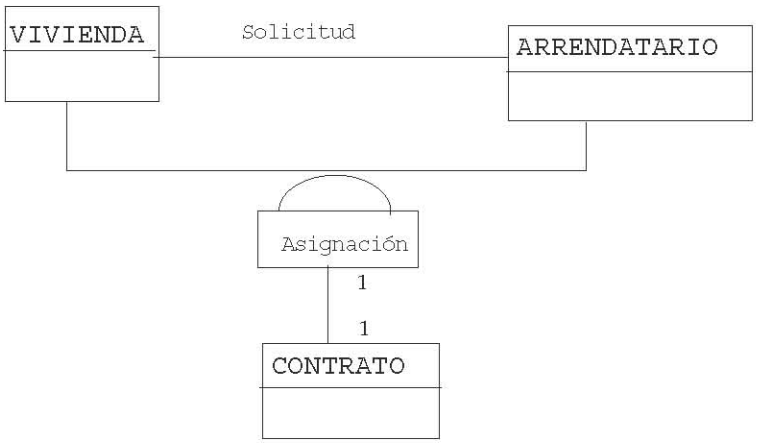


Figura 4.4 D_Domino para Arrendamientos.

Ejemplo 4 NEG_Objetivo_1: Asignación vivienda.

NEG_Objetivo_1	**Asignación vivienda**
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local
Descripción	El sistema debe asegurar la correcta asignación de las viviendas a los potenciales arrendatarios de forma que se verifiquen los criterios fijados por la administración local
Subobjetivos	No relevante
Realizado por	Personal administrativo del ayuntamiento
¿Intensivo en conocimiento?	Si
Importancia	Vital
Urgencia	Alta
Comentarios	

4.3.2.3 Formulario NEG_Dominio_m

El conjunto de todos los formularios NEG_Dominio representa un modelo de los conceptos relevantes en el dominio de la organización. Para cada concepto se incluyen los campos descritos en la tabla 4.5.

Tabla 4.5 Descripción de los campos de NEG_Dominio_m.

Referencias cruzadas	Indica cuales son los objetivos asociados y los agentes responsables de este concepto, bien porque lo posean, lo utilicen o lo actualicen.
Descripción	Es un enunciado breve que describe el concepto. Puede tratarse de un concepto en el sentido clásico o bien de conocimiento.
Estructura	Al nivel de la organización casi todos los conceptos son la composición de otros conceptos o bien de atributos elementales. En este campo se indica a modo orientativo como se estructuran los conceptos, pero no será hasta el modelado de los servicios cuando se detallen datos o atributos más concretos.
¿Formato correcto?	Este y los siguientes campos de este formulario sirven para valorar el concepto, adquieren especial relevancia si se trata de conocimiento. Puede tomar el valor Si/no o cualquier comentario.
¿Lugar correcto?	Puede tomar el valor Si/No o cualquier comentario.
¿Momento correcto?	Puede tomar el valor Si/No o cualquier comentario.
¿Calidad correcta?	Puede tomar el valor Si/No o cualquier comentario.

A continuación se muestran dos formularios, Ejemplos 5 y 6, uno correspondiente a un concepto ligado al conocimiento y otro no.

Ejemplo 5 NEG_Dominio_1: Petición.

NEG_Dominio_1	Petición
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Asignación vivienda
Responsabilidad del agente	Concejalía de vivienda
Descripción	Petición elaborada por los posibles arrendatarios
¿Formato correcto?	Si
¿Lugar correcto?	
¿Momento correcto?	Si, Cada mes, pero podrían acumularse
¿Calidad correcta?	
Comentarios	

Ejemplo 6 NEG_Dominio_2: Contrato.

NEG_Dominio_2	Contrato
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Formalización de contraltos
Responsabilidad del agente	Concejalía de vivienda
Descripción	Documento contractual por el que se rige el arrendamiento conforme a la ley de arrendamientos urbanos vigente
¿Formato correcto?	Si
¿Lugar correcto?	Si
¿Momento correcto?	Si
¿Calidad correcta?	Si
Comentarios	Es necesario guardar copia en papel rubricada por los tomadores del contrato.

4.3.2.4 Formulario NEG_Actor_p

Cada formulario de esta categoría representa una persona o papel jugado por una persona en el proyecto. Incluye los campos descritos en la tabla 4.6. Un ejemplo del formulario de actor para el proyecto Arrendamientos se muestra en Ejemplo 7.

Tabla 4.6 Formulario NEG_Actor.

Referencias cruzadas	Indica cuales son los objetivos asociados, los agentes con los que se relaciona y los conceptos y/o conocimiento que posee o utiliza.
Descripción	Es un enunciado breve que describe al agente.
Cargo	Indica la posición del agente dentro de la organización.
Otras competencias	Incluye otras tareas no relativas a la organización bajo estudio, pero que son competencia de este agente.

Ejemplo 7 NEG_Actor_1: Aseguradora.

NEG_Actor_1	Aseguradora
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Formalización de contratos
Comunicado con	Concejalía de vivienda
Conceptos relacionados	Contrato, Póliza de seguros
Descripción	Este actor representa el rol de los miembros de la compañía de seguros con la que se contratan las pólizas para cada contrato de arrendamiento formalizado
Cargo	Agente externo a la organización Sin relaciones de subordinación con los demás actores
Otras competencias	Gestión de los partes de seguro
Comentarios	

4.3.3 Documento de definición del sistema

Este documento es un punto de referencia para el posterior desarrollo del sistema. Se debe estructurar y recoger la información obtenida, pero va más allá que la simple trascripción del modelo del negocio, puesto que debe incluir información acerca de las tareas a realizar, recursos y la descripción de los componentes del sistema.

Todos los documentos del modelo de proceso InSCo deben seguir un formato genérico como el que se muestra en la tabla 4.7.

DOC_DS debe incluir contenidos o apartados de dos categorías, aquellos relativos al problema y los relativos al proyecto. También debe tenerse en cuenta que debido a la naturaleza incremental del desarrollo de este tipo de sistemas, no se obtendrá una versión definitiva de este documento en la primera iteración y al igual que el resto de artefactos definidos en esta propuesta deben ser tolerante a incompletitud y extensible. La figura 4.5 muestra un posible formato para este

documento pero no se trata de un patrón fijo. Se organiza en siete apartados, los cuatro primeros son los contenidos relativos al sistema y el resto son aquellos contenidos relativos a la gestión del proyecto.

Los contenidos relativos al sistema incluyen la descripción del modelo de negocio y centrado principalmente en los objetivos. Además, es importante destacar las deficiencias o problemas de la organización y cuales es la estrategia de solución seleccionada en cada caso. El último gran tema referente al sistema es la definición de la arquitectura o componentes del sistema. Se debe describir cómo se estructurará el sistema definitivo en cuanto a: personas, tecnología (hardware y software), información y procesos (flujos de trabajo).

Tabla 4.7 Formato de documento.

	Descripción
Portada	Cómo título se incluye el tipo del documento: Documento de definición del sistema, documento de especificación de requisitos o documento de definición de arquitectura, el nombre del proyecto, la versión que se entrega al cliente, la fecha de la publicación de la versión, el equipo de desarrollo y el nombre del cliente, normalmente una empresa.
Lista de cambios	En ella se especifican, para cada versión del documento, los cambios producidos. Para cada cambio realizado se debe incluir el número de orden, la fecha, una descripción y los autores.
Índice.	
Listas de figuras y Tablas	
Introducción	Incluye una descripción breve de los principales datos aportados por el documento y cualquier otra consideración que sitúe al posible lector en el contexto oportuno para comprender el documento.
Contenido o apartados del documento.	

Los contenidos relativos al proyecto son el resultado de las actividades de protección o de gestión de un proyecto software y que, tal como ya indicamos en la sección 1.3. "Acotación del concepto de metodología", quedan fuera de este trabajo, pero cuyo resultado debe reflejarse en este tipo de documento y por esos se indica aquí.

Portada
Tipo del documento
Nombre del proyecto
Versión
Fecha
Equipo de desarrollo
Cliente
Lista de cambios
Listas de figuras y tablas

1.0. Introducción
    Objetivos y entorno de operación
    Resumen de la viabilidad y justificación
    Recursos requeridos de coste y tiempo
2.0. Descripción del sistema. Procesos del negocio
    Objetivos estratégicos
    Estructura de la organización (información y agentes)
    Deficiencias detectadas
3.0. Propuesta de solución
    Para cada uno de los problemas estrategia de solución
4.0. Componentes del sistema. Arquitectura del sistema
5.0. Informe de viabilidad
    Análisis A/C ---- Alternativas
    Límites de coste
    A tecnológico
    Restricciones
6.0. Planificación temporal
    Calendario más o menos detallado....
7.0. Apéndices

Figura 4.5 Formato del documento de definición del sistema.

En el documento de definición del sistema debe recogerse los resultados del estudio de viabilidad realizado, prestando especial relevancia al análisis de coste/beneficio y al estudio de la viabilidad técnica. El primero es normalmente la principal consideración para la mayoría del los sistemas. La justificación económica comprende un amplio rango de aspectos, que van desde el estudio de costes directos e indirectos frente a beneficios tangibles e intangibles hasta las estrategias de ingresos a largo plazo o estudios de mercado e impacto en otros

productos similares. La viabilidad técnica es frecuentemente la más difícil de evaluar, sobre todo en las primeras iteraciones del desarrollo, su objetivo es determinar si en el estado actual de desarrollo tecnológico se pueden abordar las soluciones propuestas con garantías de éxito.

Por último como en el desarrollo de cualquier producto de ingeniería es de vital importancia la agenda o planificación temporal del proyecto, así como la asignación de recursos a las tareas fijadas.

4.4 Conclusiones del capítulo

La actividad fundamental Organización se lleva a cabo mediante las actividades **O1_Identificar la organización y el proyecto**, O2_**Identificar los procesos de negocio**, **O3_Caracterizar los procesos de negocio** y **O4_Redactar la definición del sistema**

Los métodos aplicados en estas actividades son los relacionados con la recogida y análisis de información, como son observación, sesiones de trabajo o el análisis de protocolos.

El artefacto principal resultante de la actividad fundamental Organización es el modelo de Negocio (NEG) que se debe complementar recogiéndolo sobre una notación textual denominada **Documento de definición del sistema** DOC_DS, del que se propone una plantilla.

Como parte de NEG se definen cuatro tipos de formularios: **NEG_Proyecto** que recogen los problemas y oportunidades de la propuesta y supone el punto de partida del proyecto, **NEG_Actor** que indica las personas o los papeles de las personas implicadas en el proyecto, un formulario **NEG_Objetivo** para cada objetivo estratégico de la organización, **NEG_Dominio** que representa los objetos, eventos o conceptos del dominio que se emplean para la consecución de los objetivos.

El modelo NEG se complementa con al menos la definición de dos diagramas, **D_Reglas_del_negocio** y **D_Dominio**, que recogen respectivamente una notación gráfica de los objetivos de la organización y de los conceptos del dominio y sus relaciones.

5 Desarrollo orientado al cliente

La descripción de los modelos y las actividades fundamentales del nivel conceptual se realiza en los capítulos 5 y 6. En el primero se describen las actividades fundamentales de Requisitos y Conocimiento y en el segundo la actividad fundamental Análisis.

La primera sección de este capítulo se plantea las dos perspectivas de modelado de un sistema híbrido: desde el modelo de servicios y desde el modelo de conocimiento, las cuales proporcionan dos puntos de vista del sistema. Las siguientes secciones se estructuran siguiendo estos dos puntos de vista. Así, primero, se describe la descomposición en tareas de cada una de estas actividades en las secciones 5.2 "Flujo de trabajo Requisitos" y 5.3 "Flujo de trabajo Conocimiento, y a continuación los artefactos generados por cada una de ellas en la sección 5.4 "Artefactos generados. Modelos conceptuales". En esta sección también se indica cómo usar el documento estándar de la IEEE Std 830-1998, o su equivalente electrónico, como resultado de esta actividad fundamental. Por último, se describe de forma breve como instanciar el modelo InSCo para otro medio de representación del conocimiento como son las redes Bayesianas, mostrando la capacidad de adaptación de InSCo.

5.1 Dualidad de REQUISITOS y CONOCIMIENTO

Como ya se ha indicado el proceso de modelado conceptual en un sistema híbrido se divide en dos, aquel más cercano al cliente y al mundo real, compuesto por las actividades fundamentales Requisitos y Conocimiento y el más cercano al producto a construir, que se corresponde con la actividad fundamental Análisis.

Esta división mejora la independencia entre la implementación y el modelado (Dieste, Genero, Juristo, Mate, & Moreno, 2003).

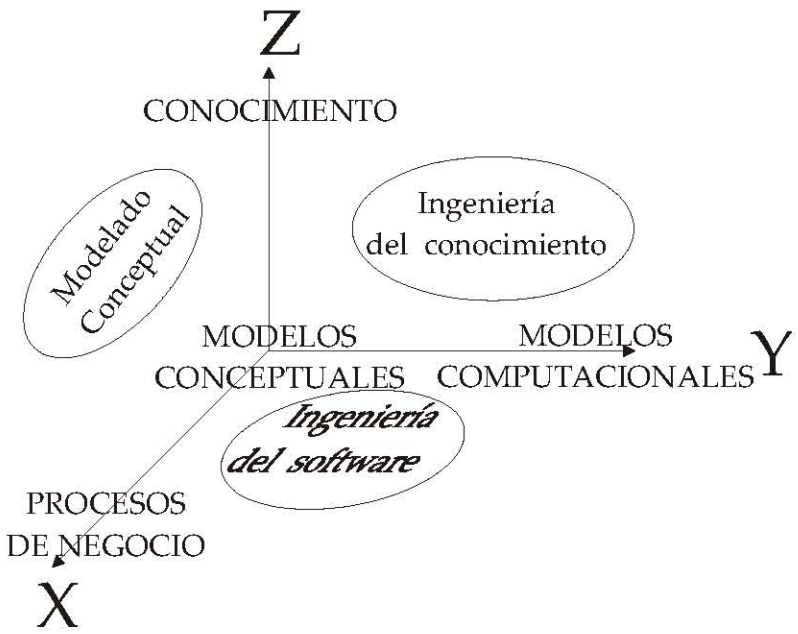


Figura 5.1 Dimensiones del los modelos conceptuales.

Como se indicó en el apartado 1.4.1 "Alcance de la propuesta", el o los modelos conceptuales son el vínculo de unión entre el universo del discurso y el sistema a construir. Es trabajo de los desarrolladores hacerlos corresponder con un modelo computacional. En los sistemas híbridos hay dos tipos de modelos conceptuales, aquellos originados desde las funcionalidades requeridas para el sistema software y el modelo de conocimiento, esta doble visión del nivel conceptual se muestra en la figura 5.1. De una parte, en el eje Z se simboliza que el modelo conceptual del nivel de conocimiento es una representación explicita y declarativa del conocimiento que los expertos aplican en los procesos de razonamiento y que deberá inyectarse sobre el modelo computacional a construir. De otra, el eje X representa las necesidades y restricciones del sistema software a construir obtenidas a partir de los objetivos y procesos del negocio. El proceso de obtener los modelos computacionales desde los modelos conceptuales del nivel de conocimiento es el objetivo de la InCo, el plano ZY. De igual forma, la InSo se representa en el plano XY. Cuando tratamos sistemas software híbridos necesitamos emplear tres dimensiones, de forma que el modelo conceptual se debe definir desde dos puntos de vista, que son respetivamente la actividad

fundamental Requisitos y la actividad fundamental Conocimiento y se representa en el plano XZ. (Águila I. M., Cañadas, Palma, & Túnez, 2006)

Es importante comprender qué Requisitos y Conocimiento no son disjuntas; es decir, un objetivo estratégico descrito en el modelo de la organización puede ser visto desde las dos actividades fundamentales, modelando cada una vistas distintas del mismo objetivo. Por ejemplo, para el proyecto Arrendamientos, para *asignación de vivienda* se aplica de un modelo de decisión basado en conocimiento (visión de conocimiento) y también es necesario generar un informe de las asignaciones realizadas y, en el caso de que los arrendatarios dispongan de dirección de correo electrónico, enviarles la noticia por correo electrónico (visión de requisitos), siendo ambas visiones necesarias para abordar la construcción de una aplicación software que cubra los objetivos del proyecto. La actividad fundamental Requisitos se ocupará de la obtención de los requisitos, centrados en el ámbito del proyecto software, desde el punto de vista del cliente o requisitos-C (Braude E. J., 2005) y para todas sus características, es decir, requisitos de información, funcionales y no funcionales (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000). La de la actividad fundamental de Conocimiento se ocupará del proceso de adquisición del conocimiento y generación del modelo de conocimiento. Este modelo siguiendo las propuestas de CK presenta tres aspectos, conocimiento del dominio, de las tareas y de las inferencias, ver el apartado 2.2.4 "Metodologías de segunda generación de SBC".

A nivel de investigación, la obtención de requisitos es sin duda la actividad a la que menos atención se le ha prestado en la InRe (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000). En cambio, la obtención de conocimientos es probablemente una de las tareas más estudiadas dentro de la InCo, puesto que esta tarea siempre ha sido un cuello de botella en el desarrollo de SBC. De hecho, tal como se indicó en el capítulo 2, el origen de las metodologías para el desarrollo de SBC estuvo centrado en agilizar esta tarea (Palma, Paniagua, Martín, & Marín, 2000). Esto nos hace proponer que las actividades de obtención de ambos aspectos se puedan realizar a la vez y bajo con la denominación de *desarrollo orientado al cliente*.

Otro punto de conexión entre el conocimiento y requisitos aparece ligado al concepto información. Si revisamos la bibliografía (Schreiber, et al., 1999) y (Gómez, Juristo, Montes, & Pazos, 1997) distinguimos entre tres términos

diferentes, datos, noticias y conocimientos y a todos ellos se les da el nombre genérico de información. Los datos son señales no interpretadas que son recibidas y registradas por algún receptor. Las noticias son datos útiles que tiene significado para el receptor. Por último el conocimiento es el núcleo de información poseída por las personas para usarla en actuaciones o generar nueva información. La tabla 5.1 muestra las diferencias entre estos conceptos con un ejemplo.

El límite entre la noticia o requisito de información (S.O.S.) y el conocimiento del dominio (Alerta de emergencia) es difícil de establecer. Con las operaciones ocurre igual, el límite entre requisito funcional y las tareas de inferencia es difuso. En el modelo de proceso InSCo se estudian a la vez, como diferentes vistas de los mismos objetivos que se acoplarán durante la actividad fundamental Análisis. Requisitos y Conocimiento modelan la misma realidad sobre lo *qué* hace el sistema, pero desde dos enfoques diferentes.

Tabla 5.1 Diferencia entre dato-noticia-conocimiento.

	Característica	**Ejemplo**
Dato	Valor sin interpretación	. . . - - - . . .
Noticia	Significado ligado al dato	S.O.S.
Conocimiento	Propósito ligado a la noticia Potencialidad para generar una acción	Alerta de emergencia -→ Comenzar operación de rescate

En el desarrollo de sistemas software híbridos hay dos actividades fundamentales en el mundo del cliente: **Requisitos** y **Conocimiento**. De estas actividades se generan el modelo conceptual del problema a resolver, desde el punto de vista del cliente, es decir, la representación del software de forma independiente de la implementación. En las figuras 5.2, 5.3 se recoge la representación de estas actividades, de forma que la definición de requisitos se realiza a través de la ejecución de cuatro actividades R1 a R4, que dan lugar al **modelo de servicios** y el **documento de especificación de los requisitos**. Las actividades C1 a C3 generan el **modelo de conocimiento**, representado este según las propuestas de CK. Cabe destacar, que si se emplease otro método de representación y modelado del conocimiento, éste encajaría perfectamente dentro de InSCo, sólo haría falta redefinir apropiadamente la actividad fundamental Conocimiento, tal como se verá en el caso de redes Bayesianas en la última sección de este capítulo.

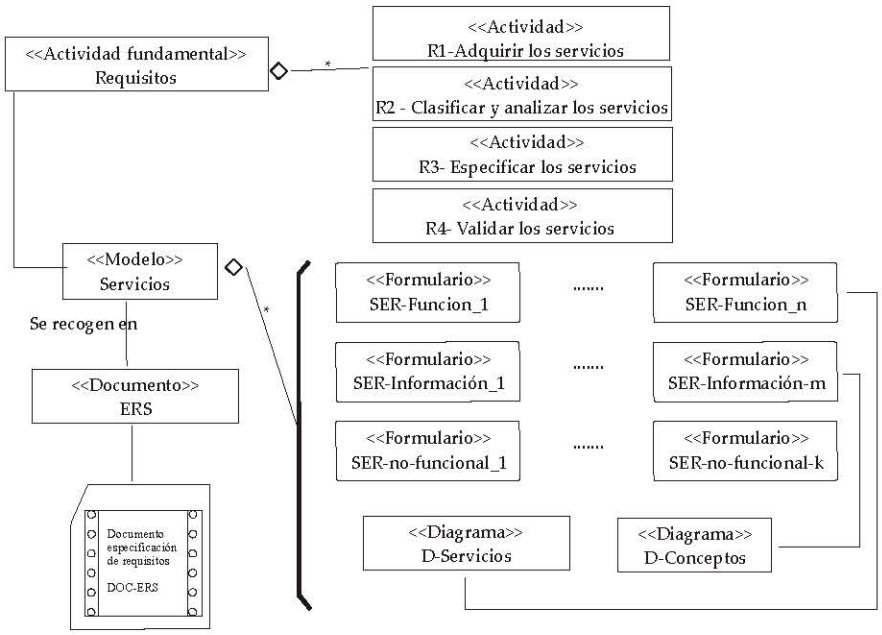


Figura 5.2 Actividad fundamental Requisitos.

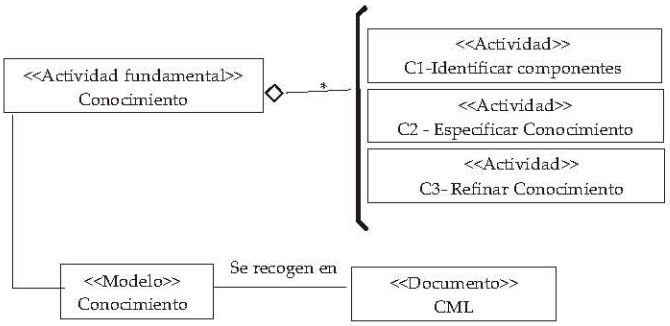


Figura 5.3 Actividad fundamental Conocimiento.

5.2 Flujo de trabajo de Requisitos

La actividad fundamental Requisitos, aplicando el esquema básico propuesto en (Abran, Moore, Bourque, Dupuis, & Tripp, 2004), se descompone en cuatro actividades principales que se muestran en la tabla 5.1, aunque hay una dependencia estrecha entre la forma de obtener el modelo de servicios y la propia organización en donde desarrolla. Esto ha sido puesto de manifiesto por diversos autores en el campo de la InRe (Kotonya & Sommerville, 1998) y (Sommerville & Sawyer, Requirements Engineering: A Good Practice Guide, 1997) y puede llevar pequeñas variaciones en su descomposición en tareas dependiendo de la organización concreta.

Tabla 5.2 Actividades de Requisitos.

Requisitos
R1_Adquirir los servicios
R2_Clasificar y Analizar los servicios
R3_Especificar los servicios
R4_Validar los servicios

La actividad fundamental Requisitos se corresponde con las tareas de ingeniería de requisitos en el desarrollo de software no basado en conocimiento. Los requisitos expresan necesidades o restricciones que deben plasmarse en el producto software y que contribuyen a la solución de un problema del mundo real (Kotonya & Sommerville, 1998), es decir, son servicios que debe ofrecer el producto software a demanda del cliente.

La **adquisición de servicios_R1** se refiere tanto a los servicios que definen explícitamente como a los que se deben extraer de entre todos los participantes [15]. Es fundamentalmente una actividad humana y es donde se identifican los participantes y se establecen las relaciones entre el equipo de desarrollo y los clientes. También se le puede llamar captura, descubrimiento u obtención de servicios.

Uno de los principios fundamentales de un buen desarrollo de software es que haya una buena comunicación entre los desarrolladores y usuarios del software. Antes de comenzar el desarrollo, se deberán definir los canales para esta comunicación. Las fuentes de posibles servicios son: los objetivos, los conceptos y relaciones del dominio, los propios participantes, y el entorno operacional y organizacional. Con lo que el mejor punto de partida de esta actividad es el modelo de negocio NEG.

Durante la actividad **R2, Clasificar y Analizar los servicios**, se produce una revisión de los servicios adquiridos, detectando y resolviendo los conflictos entre ellos. También se delimita exactamente el software a construir y su modo de interactuar con el entorno. En trabajos clásicos de ingeniería de requisitos (Sommerville, Software Engineering: (Update) (8th Edition), 2006), (Abran,

[15] Recuerdese que el término en inglés es stakeholder, se utiliza para referirse a «*quienes pueden afectar o son afectados por las actividades de una empresa,*» en este caso proyecto de desarrollo de software.

Moore, Bourque, Dupuis, & Tripp, 2004) y (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000) la actividad de análisis de requisitos también lleva implícita la definición de los requisitos del sistema a partir de los requisitos del usuario definidos en R1, pero en el caso de sistemas software híbridos esta labor se pospone para la actividad fundamental Análisis, donde se completará el modelo conceptual.

Los tipos de servicios contemplados en R2 son al menos servicios funcionales, no funcionales y de información tal como se detallarán en la sección 5.4, aunque se pueden establecer otras clasificaciones. Para InSCo es importante realizar una clasificación de cuáles de los servicios pueden también ser asignados como servicios que usan conocimiento; es decir, para cubrir las necesidades de los participantes descritas en ese servicio va ha ser necesario aplicar parte de conocimiento que se está describiendo en el modelo de conocimiento.

La **especificación de los servicios_R3** consiste en la elaboración de un documento, o su equivalente electrónico, que suponga le base contractual del proyecto o pliego de condiciones para el desarrollo de software, habitualmente se redacta en lenguaje natural, pero debe ir complementado con descripciones formales o semiformales de los servicios, que permitan la **validación de estos servicios_R4**. Cada vez es más habitual que en lugar de disponer únicamente del documento, se utilicen herramientas software que almacenen electrónicamente el modelo de servicios o requisitos, así la generación del documento de especificación será un proceso automatizado a partir de una plantilla previamente definida. La gestión de los servicios, y también de los objetivos en InSCo, se puede realizar a través de la herramienta software InSCo Requisite que se describirá brevemente en el Apéndice C y es uno de los resultados tecnológicos más importantes del proyecto TIN2004-05694. Aunque puede realizarse con cualquier herramienta comercial como DOORS, Caliber, IRqA configurada para seguir el modelo de proceso InSCo.

5.2.1 Técnicas recomendadas

Al igual que en la actividad fundamental Organización se puede aplicar cualquier técnica de búsqueda de información para la adquisición de los servicios, también es habitual utilizar lenguajes de modelado, como pueden ser UML, para completar el proceso de especificación que permiten generar partes

de un modelo conceptual de problema que terminará de definirse en la actividad fundamental de Análisis.

Las técnicas más habituales son: observación, sesiones de trabajo y análisis de protocolos. Las sesiones de trabajo agrupan técnicas en las que el usuario debe participar en más o menos medida del proceso de obtención de información; se destacan las entrevistas, desarrollo conjunto de aplicaciones o tormenta de ideas. La técnicas de análisis de protocolos tienen por objetivo responder a la pregunta *¿Qué hay que hacer para?* como estamos modelando servicios obtendremos los requisitos desde el punto de vista del cliente.

La técnica que tradicionalmente se ha empleado en la definición de protocolos es la descomposición funcional o aproximación basada en objetivos. Pero como se indicó en el capítulo 4, esta técnica es menos apropiada en la comunicación con los usuarios que los casos de uso y es por la que se ha optado para InSCo y la actividad fundamental Requisitos.

Los casos de uso y los escenarios fueron propuestos inicialmente en (Jacobson, Christerson, Jonsson, & Övergaard, 1992) y actualmente forman parte de la propuesta de UML (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual (2nd Edition), 2004). Los casos de uso describen en forma de acciones y reacciones el comportamiento del sistema y sus relaciones con el entorno. Los usuarios externos al sistema pueden desempeñar diferentes funciones, distinguiéndose entre actores primarios, para los que el objetivo de caso de uso es esencial y actores secundarios, que interactúan con el caso de uso pero cuyo objetivo no es secuencial. Los diagramas de casos de uso son una notación gráfica que permite representar los casos de uso como elipses y los actores como personajes.

Los actores y los casos de uso se vinculan mediante relaciones de comunicación que pueden representan servicios que el sistema debe suministrar a los actores, información del sistema que debe introducir, consultar o modificar un actor, cambios que intervienen en el entorno, de los que informa una actor o bien cambios que ocurren dentro del sistema y que son informados al actor. También se pueden establecer relaciones entre casos de uso, como la de inclusión (include) y la de extensión (extend). La relación de inclusión sirve para enriquecer un caso de uso con otro, es una representación de una subfunción y sirve para compartir funcionalidad común entre varios casos de uso. La relación de extensión enriquece un caso de uso mediante un caso de uso subfunción, pero

de forma opcional. Por último, cabe decir que también es posible especializar casos de uso, heredando subcaso el comportamiento y las relaciones de comunicación, extensión e inclusión del supercaso de uso.

Los casos de uso ofrecen una técnica de representación adecuada para dialogar con el usuario ya que su formalismo es cercano a lenguaje natural. Por lo tanto, es necesario describir los escenarios instancias de los casos de uso de forma textual mediante pasos que describen cada operación tal como proponen diversos autores (Duran & Jiménez, Metodología de elicitación de requisitos de sistemas software. Versión 2.1, 2000), (Booch, Rumbaugh, & Jacobson, Unified Modeling Language User Guide, The (2nd Edition), 2005) y (Cockburn, Writing Effective Use Cases, 2000).

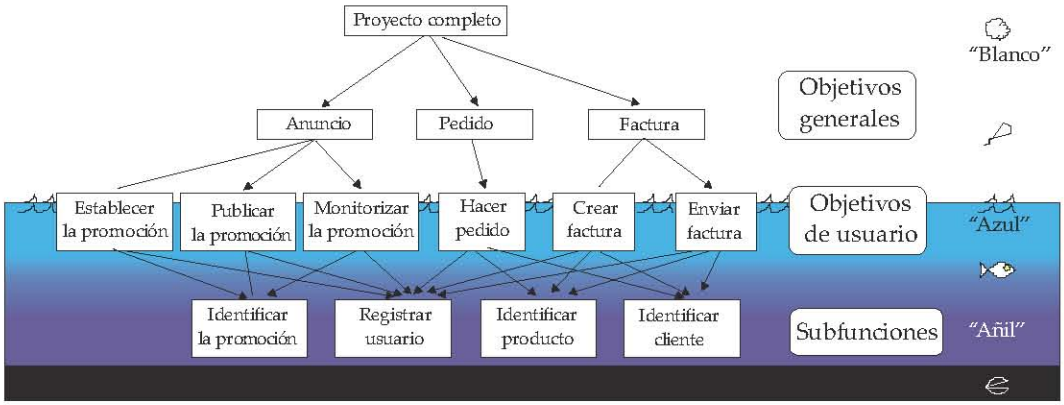


Figura 5.4 Representación de niveles de los casos de uso.

Un aspecto muy importante de esta técnica es el nivel de descripción de los casos de uso. La figura 5.4, reproducida y traducida de (Cockburn, Writing Effective Use Cases, 2000), muestra con una metáfora de varios niveles de abstracción. El nivel del mar, en azul, es el nivel de descripción en el modelo de servicios, mientras que el nivel blanco, por encima del mar, es el nivel de descripción de NEG donde se definen los objetivos. El nivel bajo el agua, representado también por el color azul marino o añil, será el refinamiento previo al diseño con el que se tienen que describir los casos de uso para el modelo de análisis (ANA). Estos niveles de refinamiento, ya descritos en el apartado 3.3.3 "Ciclo de vida. Hitos del flujo de trabajo principal", se muestran en la tabla 5.3 utilizando también los iconos propuestos por Cockburn. Así en el nivel de negocio, y con un alto nivel de abstracción, se describen las funciones en NEG, que se refinan desde el punto de vista del usuario en los requisitos, pero estos

requisitos terminarán de definir con más detalle en el Análisis, decidiendo entonces si son basados en conocimiento o no.

Tabla 5.3 Refinamiento de las funcionalidades.

	InSCo				InSo		
							
 Negocio	O-NEG	∅	∅		Sistema	∅	∅
 Sistema	∅	R-SER	D-DIS		∅	Requisitos	Diseño
 Subsistema	∅	A-ANA	D-DIS		∅	Requisitos	Diseño

Otra técnica que ayuda en el descubrimiento de los requisitos es el storyboarding o guiones gráficos. Esta técnica heredada de la industria cinematográfica ha probado su utilidad en otros aspectos de la construcción de software como la ingeniería Web o la construcción de la interfaz hombre-máquina (HCI). Se centra en el usuario, pero su principal beneficiado es el desarrollador.

Un guión gráfico es una descripción lógica y conceptual de la funcionalidad del sistema para un escenario específico, incluyendo las interrelaciones entre usuarios del sistema y el propio sistema (Sutcliffe A. , 2002). Esta técnica facilita la comunicación con los participantes y existen independientemente del sistema software, esto hace que sean más estables que los prototipos convencionales puesto que no fallan ni dan falsas impresiones sobre el sistema. Los guiones gráficos identifican los jugadores o actores, explican qué les pasa y describen cómo les pasa.

Los casos de uso y los guiones gráficos son herramientas similares, mientras que los primeros tienen su origen en el modelado orientado a objetos y están mejor definidos en cuanto a forma, los guiones gráficos provienen de la experiencia. Buscan la mejora de la comunicación con los usuarios y acercan el diseño del software al usuario más que los propios casos de uso (Gottesdiener, 2002), facilitando la obtención de los requisitos. Existen ya herramientas que plantean el uso guiones gráficos para la definición de las funciones del sistema (Thronesbery, Molin, & Schreckenghost, 2007).

5.3 Flujo de trabajo de Conocimiento

La vida económica y social cada vez más está guiada por el conocimiento. Se ha convertido en un tópico decir que vivimos en la sociedad de la información, pero sólo se ha comenzado a explorar y comprender sus consecuencias reales y cotidianas. Una de esas consecuencias es el crecimiento de la importancia del conocimiento: *"el acero fue la quintaesencia de la industrialización y el producto talismán de la era de la información es el microchip. Pero mientras que el acero obtiene su valor de su naturaleza física, cuanto más acero más valor; el microchip obtiene su valor de su diseño, cuanto mejor es el diseño más valor, el número no es importante. El valor radica en el conocimiento que se inyecta en el diseño del microchip"*. Este ejemplo, que aparece en el manual de CommonKADS (Schreiber, et al., 1999), ilustra como el conocimiento se ha convertido en el recurso clave de cualquier organización y su apropiada gestión y tratamiento es una actividad crucial en las empresas modernas. Al igual que en la revolución industrial las ingenierías mecánicas y eléctricas vinieron a ofrecer teorías, métodos y técnicas aplicables en los procesos de producción, hoy las disciplinas de la InCo y la gestión del conocimiento vienen a ofrecer estas mismas herramientas aplicadas al conocimiento.

Pero tal y como venimos repitiendo, el objetivo de este trabajo es definir un modelo de proceso para la construcción de sistemas híbridos, donde el conocimiento es un recurso vital y distinguible del resto. Desde esta perspectiva, el objetivo es que este conocimiento, como realidad de la organización, sea inyectado sobre el sistema software que se construya para resolver todos o algunos de los problemas detectados en esta organización.

Si la organización tuviese ya una política establecida para la gestión, modelado y representación del conocimiento de la organización, el objetivo de esta actividad fundamental sería no modelar sino sólo seleccionar la porción del modelo ya existente, a considerar en el proyecto de desarrollo de software concreto. Si no existe este modelo de conocimiento la actividad fundamental Conocimiento aglutina las labores relativas a adquirir, especificar y representar el conocimiento; en una palabra, generar el modelo.

El propósito del **modelo de conocimiento** es explicar en detalle los tipos y estructuras del conocimiento utilizado para realizar una tarea. Este modelo, siguiendo las propuestas de CK, tiene tres partes recogiendo cada una de ellas

una categoría de conocimiento. La primera categoría es el conocimiento del dominio. Recoge los tipos de información específicos del dominio de los que es necesario hablar en una aplicación. Es comparable al modelo de datos o de objetos de una aplicación software tradicional (es la vista desde el conocimiento de los requisitos de información). La segunda categoría es el conocimiento de las inferencias, que describe los pasos de inferencia básicos que debemos realizar cuando se usa el conocimiento del dominio. Las inferencias son los bloques constitutivos básicos de los procesos de razonamiento. La tercera categoría es el conocimiento de las tareas, que describe los objetivos perseguidos por una aplicación y como estos objetivos se descomponen en subtareas hasta llegar a las inferencias.

La descomposición de la experiencia en el conocimiento específico del dominio por un lado y de las tareas y las inferencias por otro, favorece la reutilización del conocimiento en dos niveles. Al haber definido un conjunto de inferencias elementales independientes de la aplicación se permite que estas sean utilizadas en otros procesos de resolución de problemas. El otro nivel de reutilización lo establece el conocimiento del dominio, que al ser definido independientemente de las tareas, puede ser reutilizado por otro conjunto de tareas definidas sobre el mismo dominio (Taboada, Des, Mira, & Marín, 2001).

Igual que podemos hablar de sistema de información sin informática, podemos tener un modelo de conocimiento en un sistema sin que tenga que ser implementado sobre un sistema software y la actividad fundamental Conocimiento se encargaría de generarlo.

Distinguimos tres tareas en el proceso de construcción del modelo de conocimiento, tal como se muestra en la figura 5.5, *identificación, especificación* y *refinamiento* del conocimiento. No obstante, el proceso de modelado del conocimiento es quizás el aspecto mejor desarrollado de la metodología CK y en esta memoria sólo se hará una descripción breve que puede completarse con (Schreiber, et al., 1999), (Valente, Breuker, & Velde, 1998) y (Schreiber G. , Wielinga, Akkermans, Velde, & Anjewierden, 1994).

En la tarea **C1_Identificación** de componentes se identifican las fuentes de información útiles para modelado del conocimiento. Realmente es una fase de preparación para la especificación de modelo de conocimiento actual. Se construye un glosario de términos del dominio. Se revisan componentes de

modelado ya construidos como modelos de tareas genéricas o esquemas y ontologías del conocimiento del dominio.

En la segunda tarea, **C2_Especificación**, el ingeniero del conocimiento comienza con la construcción de la especificación del modelo de conocimiento. Se utiliza un lenguaje semiformal como CML o formal como ML2 completado con la utilización de diagramas. Las tres categorías del conocimiento en CK tienen asociados sendos diagramas y cada elemento del diagrama se representa con una primitiva CML. En sí, la especificación CML y los diagramas son equivalentes, estas relaciones se muestran en la figura 5.6. Se han construido herramientas que permiten el modelado del conocimiento con este lenguaje (Cañadas, Palma, & Túnez, 2009)

Conocimiento		
	Subtarea	Resultado
C1_Identificación	Explorar las fuentes de información	Lista de fuentes de información y glosario de términos
	Enumerar los componentes potenciales	Lista de posible componentes a reutilizar a nivel componentes tareas y componentes del dominio
↓		
C2_Especificación	Seleccionar la plantilla de tarea	Plantilla o combinación de ellas que servirá de patrón en la elaboración del modelo de conocimiento
	Construir primera versión de conceptos del dominio	Principales tipos de información, esquema del dominio
	Completar la especificación del modelo de conocimiento	Modelo de conocimiento completo a falta de rellenar
↓		
C3_Refinamiento	Rellenar de contenido los modelos de dominio	Conocimiento necesario para la tarea modelada
	Validar el modelo de conocimiento	Simulación, en papel o prototipo del conocimiento.

Figura 5.5 Actividades de Conocimiento.

Existen dos aproximaciones para la construcción del modelo de conocimiento. La primera consiste en comenzar con el conocimiento de las inferencias hasta modelar el del dominio y el de las tareas (middle-up) y la segunda realiza en paralelo el modelado del conocimiento del dominio y de las tareas y las conecta con el conocimiento de las inferencias (top-down).

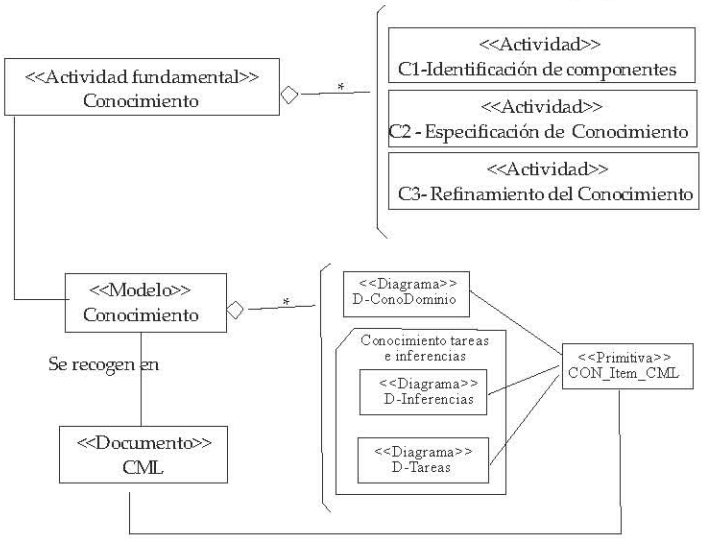


Figura 5.6 Artefactos de Conocimiento.

La última tarea, **C3_Refinamiento**, pretende la validación del modelo de conocimiento y la captura de la mayor cantidad de conocimiento o instancias de conocimiento para insertarlas en la base de conocimiento ligada a ese modelo. La validación consiste en determinar si el modelo construido puede generar el comportamiento requerido en la resolución del problema, en el caso Arrendamientos, hacer la asignación de las viviendas de acuerdo con los criterios. La técnica utilizada durante la validación suele ser la simulación en papel o a través de un prototipo del modelo de conocimiento.

Las tareas C1 a C3 están entrelazadas, no se desarrollan de forma secuencial y es normal que se necesiten varias iteraciones. Por ejemplo, para rellenar el modelo de dominio podemos necesitar recurrir a nuevas fuentes de información que han de ser identificadas.

5.4 Artefactos generados. Modelos conceptuales

Los artefactos generados por ambas actividades fundamentales deben expresarse de forma que todos los participantes sean capaces de entenderlos,

especialmente los clientes y usuarios. En este trabajo seguiremos la propuestas en las que se recomienda usar distintos lenguajes según la audiencia (Abran, Moore, Bourque, Dupuis, & Tripp, 2004). En concreto, lenguaje natural para la audiencia de clientes y usuarios y modelos conceptuales para la audiencia de los desarrolladores (ingenieros del software y del conocimiento).

5.4.1 Estructura y contenido del modelo de servicios, SER.

El modelo de servicios (SER) es el refinamiento de los Objetivos y conceptos de Dominio que se definieron en el modelo de negocio. Es decir, se estudian y describen con detalle aquellas partes del proyecto que se han fijado como viables, abordando el estudio a nivel conceptual del proyecto. La figura 5.7 muestra este refinamiento de forma que el proceso descrito en **NEG_Proyecto** se refina en los objetivos con **NEG_Objetivo** y cada uno de ellos se describe a través de los requisitos funcionales con un formulario **SER_Función** Las mismas consideraciones se aplican a los conceptos o items del dominio.

El modelo de servicios se organiza en base a tres tipos de formularios: **SER_Función**, **SER_no_funcional** y **SER_Información**, tal como se mostró en la figura 5.2. También se utilizan diagramas que apoyan la descripción del modelo y que permiten mostrar de forma visual las relaciones entre todos los tipos de servicios. Estos diagramas deben describir la información y las operaciones y/o funciones. Se pueden utilizar diversos diagramas o bien sólo uno, pero que muestre al menos estos dos puntos de vista (información/operaciones).

Todos los formularios deben incluir los campos de identificación, de referencias cruzadas a otros formularios o material y campos de comentarios, igual que los formularios de NEG. Además, cada uno de ellos incluye campos específicos para SER que se detallan en las siguientes secciones y se ilustran con ejemplos basados en el proyecto Arrendamientos.

Un artefacto fundamental, también mostrado en la figura 5.2, que se debe aparecer cuando se completa el estudio de los requisitos de un sistema software, es la especificación de los requisitos del software (ERS). En InSo, el término especificación se refiere a la producción de un documento, o su equivalente electrónico, que pueda ser sistemáticamente revisado, evaluado y aprobado. El documento ERS es una descripción completa del comportamiento del sistema que se va a desarrollar, además es el contrato entre clientes y desarrolladores.

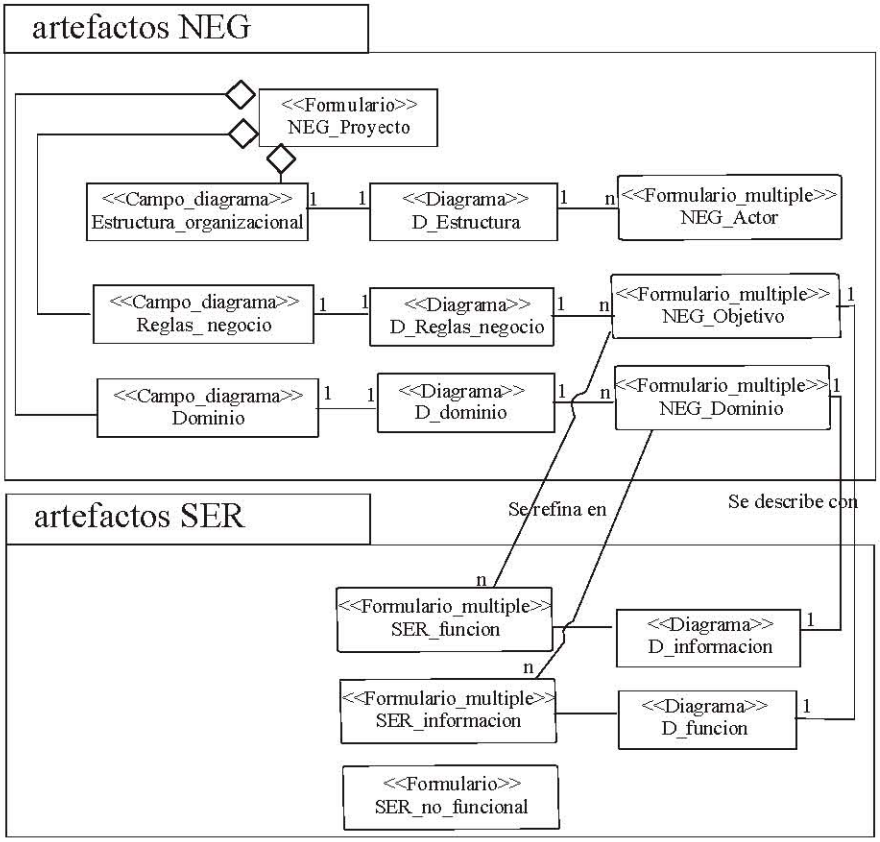


Figura 5.7 Refinamiento en el modelo de servicios.

5.4.1.1 SER_Función_i

Este formulario permite caracterizar cada requisito o servicio funcional. Como lenguaje para la descripción se puede utilizar lenguaje natural con sintaxis restringida (lenguaje estructurado) similar a las miniespecificaciones de la metodología estructurada, un diagrama de secuencia UML u otra técnica que permita representar una secuencia de acciones. Pero puesto que estamos en el ámbito del cliente, el único lenguaje común entre cliente y desarrolladores es el lenguaje natural y se propone la utilización de este lenguaje estructurado. Tal como se desarrollará en la sección de técnicas recomendadas, al igual que con los objetivos, se puede emplear la descomposición funcional para la identificación y caracterización de los procesos o tareas en los se expande cada objetivo o bien los casos de uso y escenarios. Los campos a incluir en cada formulario se muestran en la tabla 5.4.

Tabla 5.4 Descripción de los campos de SER_Función.

Versión, Autores, Fuentes	Campos de identificación.
Referencias cruzadas	A objetivos y otros requisitos.
Descripción	Especificación en lenguaje natural del propósito de este requisito funcional.
Precondición	Se expresan en lenguaje natural las condiciones necesarias para que se pueda realizar el caso de uso.
Secuencia normal	Campo contiene la secuencia normal de interacciones del caso de uso. En cada paso, un actor o el sistema realiza una o más acciones, o se realiza (se incluye) otro caso de uso. Un paso puede tener una condición de realización, en cuyo caso si se realizara otro caso de uso se tendría una relación de extensión.
Postcondición	Se expresan en lenguaje natural las condiciones que se deben cumplir después de la terminación normal del caso de uso.
Excepciones	Especifica el comportamiento del sistema en el caso de que se produzca alguna situación excepcional durante la realización de un paso determinado.
Rendimiento, Cota de tiempo	Puede especificarse el tiempo máximo para cada paso en el que el sistema realice una acción.
¿Intensivo en conocimiento?	Si en el estado actual de recolección de información es intensivo en conocimiento, aunque puede cambiar su estatus cuando se evolucione en el estudio del proyecto.
Frecuencia	Se indica la frecuencia esperada de realización del caso de uso.
Importancia	Como para el formulario NEG_Objetivo.
Urgencia	
Estado	Indica el estado desde el punto de vista de su desarrollo. Ver apartado 3.3.3 "Ciclo de vida. Hitos en el flujo de trabajo principal"
Estabilidad	Indica la estabilidad, es decir una estimación de la probabilidad de que pueda sufrir cambios en el futuro. Esta estabilidad puede indicarse mediante un valor numérico o mediante una expresión enumerada como alta, media o baja, es información importante de cara a la evolución del software.
Referencia	Material adicional necesario para comprender el requisito.
Comentario	Comentarios adicionales al requisito.

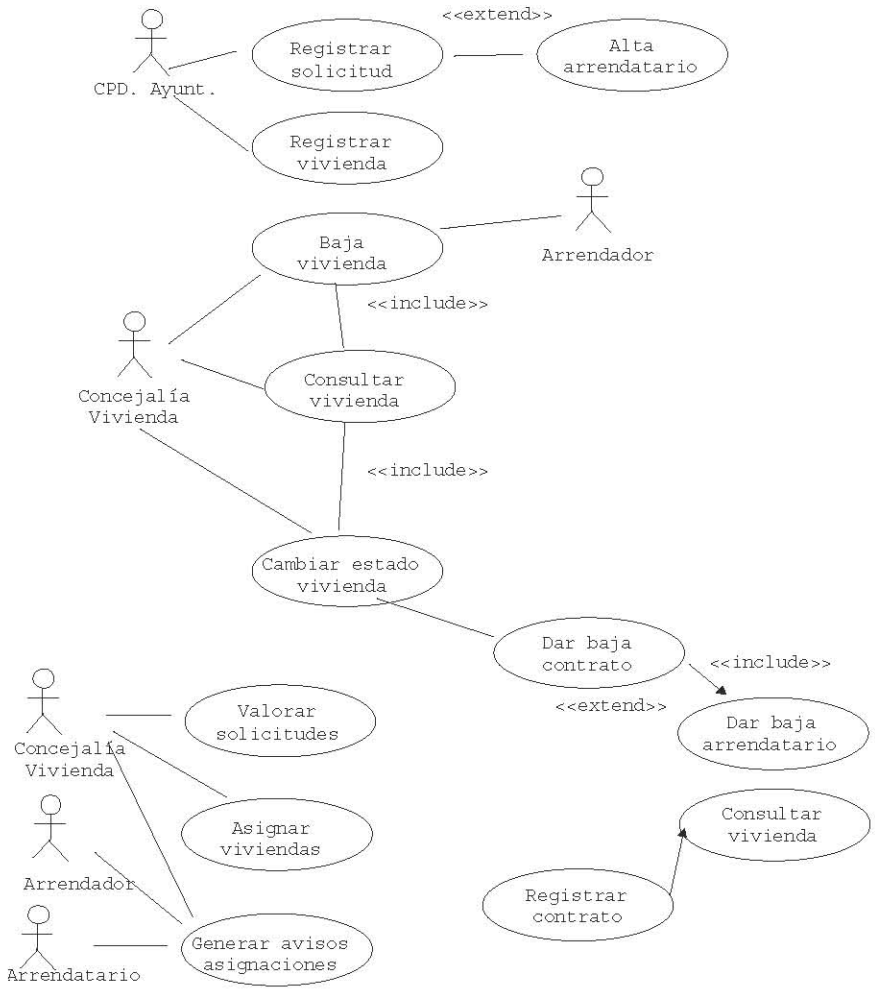


Figura 5.8 Diagrama de casos de uso para el modelo de servicios.

Para Arrendamientos, y eligiendo como mecanismos de definición de los servicios funcionales los casos de uso, tendríamos como referencia un diagrama de casos de uso que representa todos los servicios funcionales de cada objetivo estratégico considerado y, para cada uno de ellos, una descripción con un formulario SER_Función. Pueden aparece otros servicios funcionales que no tenga relación directa con una interacción, es decir, un caso de uso, el típico ejemplo de este tipo de servicios es "realización de copias de seguridad". En este caso, se genera un formulario SER_Función que no aparecerá reflejado en los diagramas de casos de uso. Este diagrama correspondiente a los objetivos gestión

de información de iniciativa vivienda y asignación de viviendas se muestra en la figura 5.8.

La siguiente tabla muestra la descripción mediante un formulario SER_Función del servicio registrar solicitud.

Ejemplo 8 SER_Funcion_1: Registrar solicitud.

SER_Funcion_1	Registrar solicitud	
Versión	Versión 1, 05-Marzo-2010	
Autores	Isabel María del Águila Cano - Universidad de Almería	
Fuentes	Centro de procesamiento de datos del Ayuntamiento	
Objetivos asociados	Gestión de datos de iniciativa vivienda	
Requisitos asociados	RI-1 Solicitud RI-2 Arrendatario	
Descripción	El sistema deberá comportarse tal como se describe cuando llegue una nueva solicitud al ayuntamiento	
Precondición	El plazo de solicitudes debe estar abierto	
	1	El arrendatario inicial el registro de la solicitud
	2	El sistema pide que el NIF del arrendatario
	3	Si no existe una solicitud anterior con ese DNI dar alta de arrendatario, ejecutando el caso de uso Alta Arrendatario
	4	Si existe obtener datos de la solicitud anterior
	5	El sistema pide que se introduzcan los datos de la solicitud, o se modifiquen
Poscondición	El sistema ha almacenado la información de la solicitud	
Excepciones		Si es la segunda solicitud de este mes comprobar la fecha de la última válida
Rendimiento		
¿Intensivo en conocimiento?	No	
Frecuencia	Una vez al mes	
Importancia	Alta	
Urgencia	Vital	
Estado	Validada	
Estabilidad	Estable	
Referencia		
Comentario	Posibilidad de extender el registro vía Web	

5.4.1.2 SER_Información_j

Un aspecto importante en los sistemas de información es precisamente la información que gestionan. En la sección 2.3 "Metodologías para sistemas híbridos" se mostró como diversos autores clasifican los requisitos (Duran A. , 2000) y (Braude E. J., 2005). Uno de los tipos de requisitos utilizados son los relativos al almacenamiento de información. El formulario para estos requisitos, cuyos campos se incluyen en la tabla 5.4, se generan respondiendo a la pregunta *"¿qué información, relevante para los objetivos del negocio, debe ser almacenada por el sistema?"*. Supone un refinamiento de los conceptos descritos en el modelo de Negocio.

Si el paradigma que se seleccione para la implementación de los datos es orientado a objetos, estos servicios de información son la primera versión de las clases persistentes del sistema, si se emplea el paradigma estructurado es la primera versión de análisis de las tablas de la base de datos.

Tabla 5.5 Campos de los requisitos de información.

Versión, Autores, Fuentes	Campos de identificación.
Objetivos asociados	Referencias cruzadas.
Requisitos asociados	Referencias cruzadas.
Descripción	Identificación del concepto sobre el que se almacenarán los datos específicos descritos en el siguiente campo.
Datos específicos	Lista de los datos específicos asociados al concepto relevante, de los que pueden indicarse todos aquellos aspectos que se considere oportunos (descripción, restricciones, ejemplos, etc.)
Intervalo temporal	Indica durante cuánto tiempo es relevante la información para el sistema. Puede tomar los valores pasado y presente, si la información es siempre relevante, o sólo presente si la información tiene un periodo de validez concreto.
Importancia, Urgencia, Estabilidad, Referencia, Comentarios	Igual que en servicios funcionales.

Para el proyecto Arrendamientos la figura 5.9 muestra el diagrama de conceptos en UML que representa los requisitos de información y, para el caso de la Vivienda, en Ejemplo 9 se muestra el formulario correspondiente a ese requisito de información.

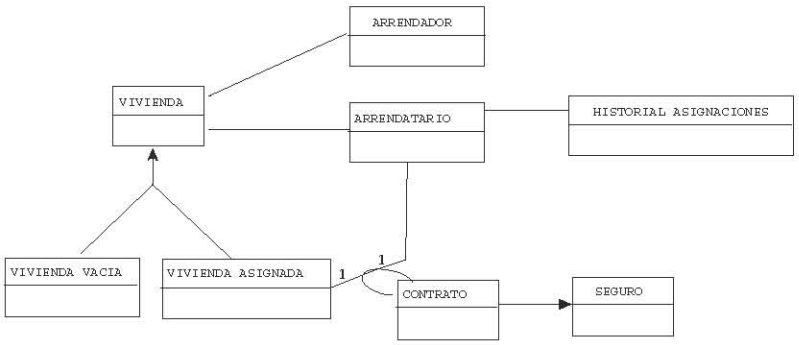


Figura 5.9 Requisitos de almacenamiento de información.

Ejemplo 9 SER_Información_1: Vivienda.

SER_Información_1	Vivienda
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Centro de procesamiento de datos del Ayuntamiento
Objetivos asociados	Gestión de datos iniciativa vivienda
Requisitos asociados	Alta vivienda
Descripción	El sistema deberá almacenar la información correspondiente a una vivienda a alquilar.
Datos específicos	Identificador: entero Dirección: cadena Renta: entero Ciudad: cadena N° habitantes: entero N° habitaciones:entero Referencia catastral: cadena Agua: lógico Luz: lógico Teléfono: lógico ADSL: lógico Electrodomésticos: lógico Gastos Comunes: entero Otros: cadena Fumadores: lógico Mascotas: lógico
Intervalo temporal	Persistente
Importancia	Alta
Urgencia	Vital
Estado	Validado
Estabilidad	Alta
Referencia	-
Comentarios	Posibilidad de incluir plano

5.4.1.3 SER_no_funcional

Son aquellos servicios que no se refieren directamente a las funciones o datos específicos del sistema, sino a las propiedades emergentes como la fiabilidad, la respuesta en el tiempo o la capacidad de almacenamiento. En muchas ocasiones son más críticos que los demás requisitos, pero el problema es que son difíciles de formular y de verificar, porque alcanzan a todo el sistema y su grado de satisfacción no se puede comprobar sobre componentes discretos (Abran, Moore, Bourque, Dupuis, & Tripp, 2004). La figura 5.10 muestra la clasificación que tradicionalmente se da a los requisitos no funcionales (Sommerville, Software Engineering: (Update) (8th Edition), 2006). Se redactan para reflejar las metas generales de los usuarios como son la facilidad de uso, la capacidad de recuperación frente a fallos o la respuesta rápida al usuario. Actualmente, se está desarrollando este aspecto de la InRe ya que hay opiniones que lo consideran como un cajón de sastre excesivamente heterogéneo donde se han clasificado aquellos requisitos que resultan incómodos (Duran A. , 2000). Siguiendo las propuestas de este último autor, algunos de los aspectos que tradicionalmente se consideraban como no funcionales, por ejemplo el rendimiento, se han integrado en los formularios para servicios funcionales.

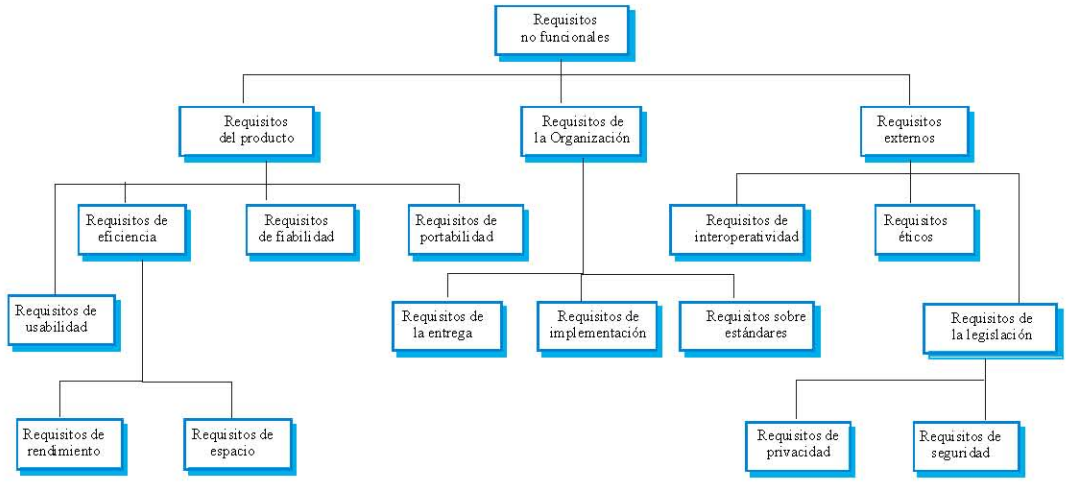


Figura 5.10 Requisitos no funcionales.

Tabla 5.6 Campos de los servicios no funcionales.

Versión, Autores, Fuentes	Campos de identificación.
Objetivos asociados	Referencias cruzadas.
Requisitos asociados	
Descripción	Capacidad que deberá presentar el sistema.
Importancia, Urgencia, Estado, Estabilidad, Referencia, Comentarios	Cómo en formularios anteriores.

Ejemplo 10 SER_no_funcional_1: Confidencialidad.

SER_no_funcional_1	Confidencialidad
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Centro de procesamiento de datos del Ayuntamiento
Objetivos asociados	Gestión datos iniciativa vivienda
Requisitos asociados	Alta vivienda, consulta vivienda
Descripción	Confidencialidad en la dirección exacta de la vivienda
Importancia	Media
Urgencia	Baja
Estado	
Estabilidad	Descrito
Referencia	
Comentarios	El objetivo es evitar ocupaciones o asaltos

5.4.2 Estructura y contenido del modelo de Conocimiento, CON

El propósito del modelo de conocimiento es explicar en detalle los tipos y estructuras del conocimiento utilizados para realizar una tarea. Proporciona una descripción a nivel conceptual del papel que juegan los diferentes componentes de conocimiento en la resolución del problema. El modelo de conocimiento tiene tres partes cada una de ellas recoge una categoría de conocimiento.

La primera categoría es el **Conocimiento del Dominio**. Recoge los tipos de información específicos del dominio de los que es necesario hablar en una aplicación. Es comparable al modelo de datos o de objetos de una aplicación software tradicional (es la vista desde el conocimiento de los conceptos del

dominio). La segunda categoría es el **Conocimiento de las Inferencias**, que describe los pasos de inferencia básicos que debemos realizar cuando se usa el conocimiento del dominio. Las inferencias son los bloques constitutivos básicos de los procesos de razonamiento. La tercera categoría es el **Conocimiento de las Tareas** que describe los objetivos perseguidos por una aplicación y como estos objetivos se descomponen en subtareas hasta llegar a las inferencias. Tanto el conocimiento de las inferencias (pasos básicos) como los de las tareas (servicios funcionales) suponen la vista desde el conocimiento de aquellas tareas y objetivos perseguidos por la aplicación.

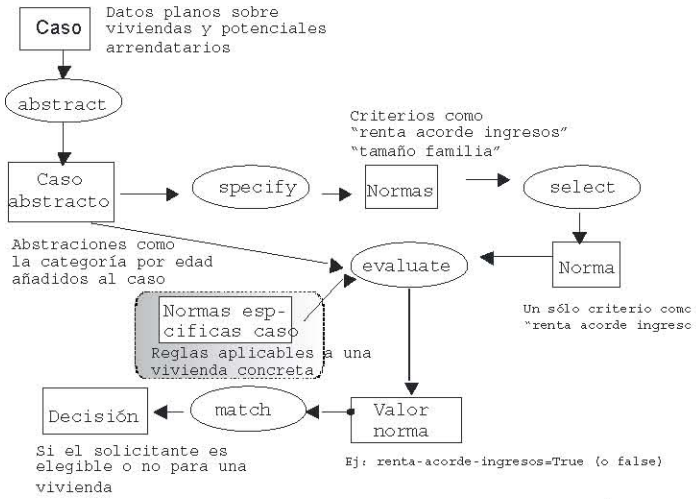


Figura 5.11 Conocimiento de inferencias para la Asignación de vivienda en Arrendamientos.

En cuanto a su notación, cabe decir que los autores proponen distintos niveles de descripción: a) *Descripción gráfica*, para el conocimiento del dominio se utiliza en la mayor medida posible el UML aunque para algunos aspectos como las reglas se extiende este lenguaje, para el conocimiento de tareas e inferencias se emplean unos diagramas específicos que son los diagramas de inferencias y los diagramas de tareas/métodos; b) *Descripción conceptual*, que se corresponde con descripciones semiformales que incluyen descripciones en lenguaje natural en un lenguaje similar al empleado en las miniespecificaciones de los procesos primitivos de los métodos estructurados y cuyo nombre es CML (Conceptual Modeling Language); y, c) *Descripción formal*, donde CK propone un lenguaje lógico formal para el modelado del conocimiento ML2 (Akkermans, van Harmelen, Schreiber, & Wielinga, 1993). Desde la perspectiva de esta tesis se

empleará CML ya que es más operativo. Fue propuesto en la fase final del proyecto KADS-I y desde entonces ha evolucionado hasta la versión actual (Anjewierden & Schreiber, 1994), pero en el estado actual de este tipo de representaciones debería mejorarse con una nueva versión lo más cercana posible al estándar UML.

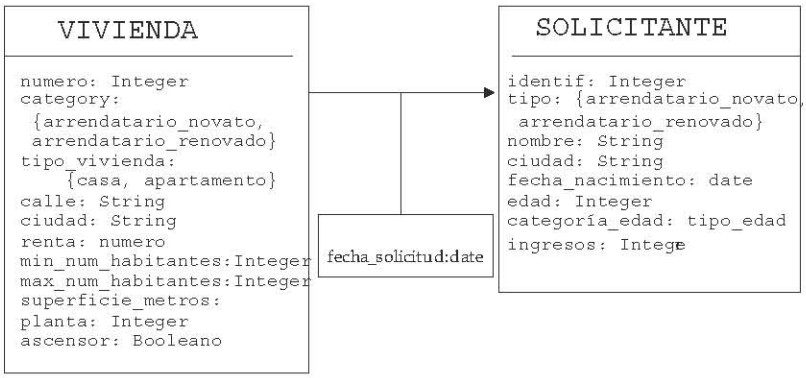


Figura 5.12 Porción del conocimiento del dominio para el proyecto Arrendamientos.

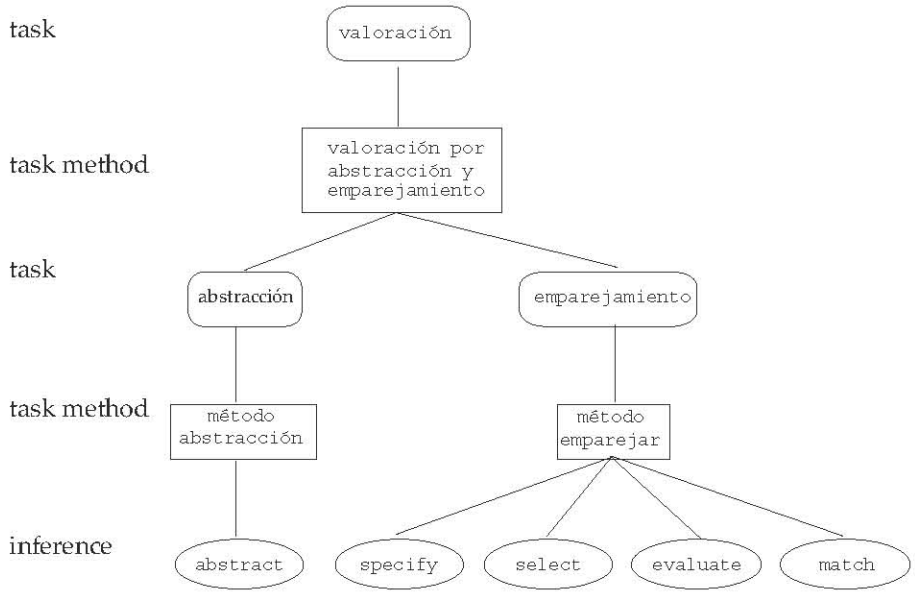


Figura 5.13 Conocimiento de tareas métodos para Asignación de vivienda.

En resumen, tenemos el conocimiento del dominio, el conocimiento de las tareas e inferencias que suponen una representación independiente de la

implementación de conocimiento aplicado en la resolución de ciertas tareas. Para el proyecto Arrendamientos el conocimiento de inferencias, dominio y tareas se muestran respectivamente en las figuras 5.11, 5.12, 5.13, siendo Ejemplo 11 una porción de CML para Arrendamientos.

Ejemplo 11 Porción de código CML de Arrendamientos.

```
KNOWLEDGE-MODEL Arrendamientos;
/* Modelo de conocimiento para la tarea de asignación de peticiones en el dominio
de la asignación de vivienda   */

DOMAIN-KNOWLEDGE vivienda-dominio;
DOMAIN-KNOWLEDGE-SCHEMA esquema-valoracion;
         /* tipos de información vivienda/arrendatario */
CONCEPT vivienda- disponible;
    DESCRIPTION:
        "vivienda que puede ser asignada";
    VIEWPOINTS:
        DIMENSION: categoria;
            vivienda-nueva, vivienda-registrada;
            DISJOINT: YES;
            COMPLETE: NO;
    HAS-PARTS:
        vivienda;
            ROLE: descripcion;
            CARDINALITY: 1-1;
    PROPERTIES:
        vivienda-num: NATURAL;
    AXIOMS:
          categoria = vivienda-nueva OR    categoria = vivienda-registrada
        <->     descripcion.tipo-subvencion = subvencionable;
END CONCEPT disponible-vivienda;
…
CONCEPT potencial-arrendatario;
    DESCRIPTION:
        "Una persona o grupo de personas registradas como posibles peticionarios
de un vivienda";
    SUPER-TYPE-OF: nuevo, residente-registrado;
        DISJOINT: YES;
        COMPLETE: YES;
    PROPERTIES:
        nombre: STRING;
        dirección-actual: STRING;
        ciudad: STRING;
        fecha-nacimiento: STRING;
        categoria-por-edad: valor-categoria-por-edad;
        año-registro: NATURAL;
        num-miembros: NATURAL;
        miembros: valor-miembros;
END CONCEPT potencial-arrendatario;
…
VALUE-TYPE valor-categoria-por-edad;
     TYPE: ORDINAL;
     VALUE-LIST: {'18-27', '28-64', '+65'};
END VALUE-TYPE valor-categoria-por-edad;
…
RULE-SCHEMA requisito-vivienda;
    ANTECEDENT:
        vivienda-solicitante;
            CARDINALITY: 1+;
    CONSEQUENT:
        vivienda-criterio;
            CARDINALITY: 1;
    CONNECTION-SYMBOL:
        indica;
```

```
END RULE-SCHEMA requisito-vivienda;
…
RULE-SCHEMA vivienda-regla-decision;
    ANTECEDENT:
       vivienda-criterio;
    CONSEQUENT:
       vivienda-decision;
    CONNECTION-SYMBOL:
       implica;
END RULE-SCHEMA vivienda-regla-decision;
END DOMAIN-KNOWLEDGE-SCHEMA esquema-valoracion;

DOMAIN-MODEL valoracion-conocimiento;
    USES:
       abstraccion-vivienda FROM esquema-valoracion,
       requisito-requisito-vivienda FROM esquema-valoracion,
       vivienda-regla-decision FROM esquema-valoracion;
    EXPRESSIONS:
…
       arrendatario.ingresos-anuales>= 25000 AND
       arrendatario.ingresos-anuales<  29999AND
       vivienda.descripcion.renta < 800
            INDICA
        renta-acorde-ingresos.valor-de-verdad= true;

       arrendatario.ingresos-anuales> 30000 AND
       vivienda.descripcion.renta < 900
            INDICA
        renta-acorde-ingresos.valor-de-verdad= true;
…
       /* reglas de decisión */

       correct-vivienda-categoria.valor-de-verdad= true AND
       correct-tamaño-familia.valor-de-verdad= true AND
       renta-acorde-ingresos.valor-de-verdad= true AND
       vivienda-specific-constraints.valor-de-verdad= true
           IMPLICA
       decision.valor = eligible;

       correct-vivienda-categoria.valor-de-verdad= false
           IMPLICA
       decision.valor = not-eligible;
…
END DOMAIN-MODEL valoracion-knowledge;
END DOMAIN-KNOWLEDGE

INFERENCE-KNOWLEDGE valoracion-inferencias;
…
INFERENCE evaluate;
    ROLES:
      INPUT:
       criterios,
       descripcion-caso,
       requisitos-especificos-caso;
      OUTPUT:
       valor-criterios;
      STATIC:
       requisitos;
    SPECIFICATION: "
       Establecer el valor de verdad de los criterios de entrada para la dada
descripcion caso. El conocimiento de dominio subyacente está formado por dos
los   requisitos   de   la   base   de   conocimientos,   así   como   otros
requisitos de casos específicos, que forman parte de la entrada. ";

END INFERENCE evaluate;
…
END INFERENCE-KNOWLEDGE
```

```
/* Tareas */

TASK-KNOWLEDGE valoracion-tareas;

TASK valorar-caso;
    DOMAIN-NAME: valorar-vivienda-solicitante;
    GOAL: "
       Valora si una solicitud para una vivienda de un determinado arrendatario
satisface los criterios
    ROLES:
      INPUT:
        descripcion-caso: "Datos sobre el arrendatario y la vivienda";
        criterios-especificos-caso: "criterios del caso";
      OUTPUT:
        decision: "eligible o no eligible para una vivienda";
END TASK valorar-caso;

TASK-METHOD valoración-por-abstraccion-y-emparejamiento;
    REALIZES:
       valorar-caso;
    DECOMPOSITION:
       TASKS: abstraer-caso, emparejar-caso;
    ROLES:
      INTERMEDIATE:
       caso-abstracto: "Caso original y sus abstraciones";
    CONTROL-STRUCTURE:
       caso-abstracto(descripcion-caso -> caso-abstracto);
       match-case(caso-abstracto+ requisitos-especificos-caso
             -> decision);
END TASK-METHOD valoración-por-abstraccion-y-emparejamiento;
…
END TASK-KNOWLEDGE valoracion-tareas;

END KNOWLEDGE-MODEL
```

5.4.3 Documento de especificación de requisitos

En el campo de la InSo no existe un documento estándar para la especificación de los requisitos del sistema software a construir. Los proyectos largos y complejos habitualmente utilizan tres documentos que en diferentes organizaciones tienen distintos nombres y estructuras.

La IEEE Computer Society nombra esos documentos como Concept of Operations (ConOps, Document IEEE 1362-1998), System Requirements Specifications (IEEE 1233-1998), y Software Requirements Specification (IEEE Std 830-1998). Este último es el más conocido y es el que es siempre necesario, si no en términos de su estructura, si al menos en cuanto a contenido.

El documento ERS se describe en (IEEE Std 830-1998) y (Institute of Electrical and Electronics Engineers Staff, 1998), pero hay que tener en cuenta que no se está hablando estrictamente en términos de documentos, puede tratarse de su equivalente electrónico. De hecho, uno de los primeros grandes avances en la gestión de requisitos ha sido dejar de pensar en documentos para pensar en

información (Hood, Wiedemann, Fichtinger, & Pautz, 2008). Para poder manejar esta información, se tiene que recurrir a bases de datos y a herramientas que permitan la gestión de estos datos, las más conocidas son IRqA, Telelogic DOORS, Borland Caliber, y el IBM-Rational RequisitePro. Nosotros hemos desarrollado InSCo Requisite como prototipo de herramienta que utiliza InSCo. En general todas ellas poseen funcionalidades de generación de informes.

Portada
Tipo del documento
Nombre del proyecto
Versión
Fecha.
Equipo de desarrollo
Cliente
Lista de cambios
Listas de figuras y tablas
1. Introducción
    1.1 Propósito
    1.2 Alcance
    1.3 Definiciones, acrónimos y abreviaturas
    1.4 Referencias
    1.5 Resumen
2. Descripción general
    2.1 Perspectiva del producto
    2.2 Funciones del producto
    2.3 Características del usuario
    2.4 Restricciones
    2.5 Supuestos y dependencias
    2.6 Requisitos futuros
3. Requisitos específicos
    3.1 Interfaces externos
    3.2 Funciones
    3.3 Requisitos de rendimiento
    3.4 Requisitos de base de datos lógica
    3.5 Restricciones de diseño
    3.6 Atributos del sistema software
4. Apéndices
5. Índice

Figura 5.14 Formato del documento de especificación de requisitos.

Como ya se ha indicado, el objetivo de InSCo es la integración y la minimización del coste del cambio. Por lo tanto, **DOC_ERS** debe tener un formato similar en cuanto a estructura y contenido al estándar Std 830-1998. Como se indicó en la sección 4.3 "Artefactos generados por ORGANIZACIÓN", los documentos del modelo de proceso InSCo deben seguir un formato genérico como el que se muestra en la tabla 4.7 "Formato del documento"

Finalmente, debe tenerse en cuenta que debido a la naturaleza incremental del desarrollo de este tipo de sistemas, no se obtendrá una versión definitiva de este documento en la primera iteración y, al igual que el resto de artefactos definidos en esta propuesta, debe ser tolerante a incompletitud y extensible. En especial, debe permitir que durante la actividad fundamental Análisis se realice la clasificación disjunta a funcionalidad basada o no basada en conocimiento. La figura 5.14 muestra un posible formato para este documento.

5.5 Instanciación de Conocimiento para Redes Bayesianas

Una de las metodologías soporte en este trabajo de tesis es CK, que utiliza una representación del conocimiento a través tres tipos de conocimiento: el conocimiento del dominio, el conocimiento de las inferencias y el conocimiento de tareas. Pero en el mundo de la InCo existen otros muchos lenguajes de representación del conocimiento aplicado por los expertos en la resolución de una determinada tarea en un dominio concreto, sin ser exhaustivo podemos citar: representaciones formales (lógicas), representaciones estructuradas (reglas, marcos, redes semánticas), representaciones con incertidumbre (lógicas difusas, razonamiento probabilístico) y representaciones conexionistas (redes neuronales). El número es muy amplio y, en todos los casos, la forma de construir el modelo de conocimiento correspondiente, es decir, las actividades que deben llevar a cabo los ingenieros del conocimiento, tienen poco en común. No obstante, la actividad fundamental Conocimiento debería servir de marco de referencia a cualquiera de ellas, instanciándose a cada mecanismo de representación concreto. A continuación, tras introducir brevemente el concepto de red Bayesiana se mostrará una posible adaptación de la actividad fundamental Conocimiento para este medio de representación.

5.5.1 Definición de Red Bayesiana

Las redes bayesianas (BN), pueden utilizarse como mecanismo de modelado del conocimiento de un experto en un dominio concreto. Formalmente una Red Bayesiana (Pearl, 1988), (Cowell, 1999), (Jensen & Nielsen, 2007) y (Kjaerulff & Madsen, 2008) es un par ***(G, P)***, donde ***G =(U, A)*** es un grafo dirigido acíclico (DAG) con un conjunto de nodos $\boldsymbol{U} = \{V_1, V_2, \ldots, V_n\}$ (variables) y un conjunto de arcos ***A***, que representan relaciones de dependencia directa entre las variables. El segundo componente del par, ***P***, es la distribución de probabilidad condicionada, sobre ***U*** que puede factorizarse de acuerdo con la siguiente ecuación:

$$P(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} P(V_i | pa(V_i))$$

donde $P(V_i|pa(V_i))$ es la probabilidad condicionada de cada variable V_i en ***U*** dado el conjunto de padres $pa(V_i)$ en el grafo DAG. Los primeros trabajos sobre este tipo de representación del conocimiento se ocuparon de hacer computables los procesos de inferencia. Conforme se fueron desarrollando los algoritmos, la atención de los investigadores se centró en las dificultades de encontrar expertos dispuestos a compartir sus conocimientos para poder inyectarlos en los sistemas software. Esto llevó al desarrollo de métodos de aprendizaje automático. Si se revisa la definición de red Bayesiana está claro que tiene dos componentes bien diferenciadas, la cualitativa y la cuantitativa: el grafo y la distribución de probabilidad condicionada, respectivamente.

5.5.2 Flujo de trabajo de Conocimiento para Redes Bayesianas

Los tres pasos básicos que se deben realizar para construir una red Bayesiana (Korb & Nicholson, 2003), (Neapolitan, 2003), (Buntine, 1996), (Heckerman, 1999), (Henrion, 1987) y (Águila, Sagrado, Túnez, & Orellana, Julio 2010 - Aceptado pendiente celebración) son:

Estudio de viabilidad. Se decide si la el modelo de red Bayesiana puede ser utilizado en ese dominio y en el tipo de aplicación.

Identificación de variables. En esta etapa se toman decisiones sobre como representar el dominio y qué se va a representar sobre la red Bayesiana.

Identificación de la estructura. Debe identificarse la estructura de la red, capturando las relaciones entre las variables. En general una variable se conecta mediante un arco con otra si la primera afecta a la segunda. Si no existe arco entre las variables implica una independencia real en el dominio.

Obtención de parámetros. Tras la definición de la estructura del DAG, la fuerza de la relación entre las variables se cuantifica mediante la especificación de las probabilidades condicionadas (Cestnik, 1900), (Jensen & Nielsen, 2007) y (Kjaerulff & Madsen, 2008)

Validación y prueba. Verifica si el modelo cumple los criterios de utilización.

Antes de realizar la construcción de la red Bayesiana propiamente dicha, es una buena idea asegurarse de si es el modelo apropiado. Una BN podría no ser apropiada si (Korb & Nicholson, 2003): el problema es muy complejo, el problema tiene una utilización limitada (sólo se utilizará pocas veces), no hay expertos en el dominio o datos útiles para la búsqueda del conocimiento o el problema puede resolverse mediante el aprendizaje de una función explicita a partir de los datos disponibles (aplicando técnicas de aprendizaje automático). Este estudio de viabilidad, se debe realizar durante la actividad fundamental Organización, ya que es en ella cuando se estudia el alcance del proyecto y se definen los objetivos.

Conocimiento		
	Subtarea	Resultado
C_BN1_Identificación	Identificación de variables	$\boldsymbol{U} = \{V_1, V_2, \ldots, V_n\}$
	Identificación de la estructura	$\boldsymbol{G} = (\boldsymbol{U}, \boldsymbol{A})$
↓		
C_BN2_Cuantificación	Cuantificación de parámetros	$P(V_1, V_2, \ldots, V_n) = \prod_{i=1}^{n} P(V_i \mid pa(V_i))$
↓		
C_BN3_Validación	Validación de la red Bayesiana	$(\boldsymbol{G}, \boldsymbol{P})$

Figura 5.15 Actividades de Conocimiento con Redes Bayesianas.

La redefinición de la actividad fundamental InSCo Conocimiento para la utilización de BNs se muestra en la figura 5.15. De esta forma, tenemos las actividades C_BN1 a C_BN3, que tienen una correspondencia casi directa con la lista de pasos básicos para la construcción de una red bayesiana y consensuada por diversos autores (Águila, Sagrado, Túnez, & Orellana, Julio 2010 - Aceptado pendiente celebración).

La actividad **C_BN1_Identificación** conlleva el proceso de identificación de las variables y las relaciones entre ellas. Los expertos y los ingenieros del conocimiento deben acordar las variables implicadas en el problema. Esta identificación puede realizarse a través de la interacción con los expertos o por medio de la clasificación no supervisada.

Los resultados obtenidos durante la identificación y caracterización de los procesos de negocio (O2, O3) y toda la actividad fundamental Requisitos deben utilizarse para la identificación de variables. En especial los diagramas y formularios relativos al dominio y los servicios de información (D_Dominio y D_Información). Durante la identificación de variables (C_BN1) se detectan las hipótesis y se agrupan en conjuntos mutuamente exclusivos y exhaustivos para definir las variables hipótesis. Se recoge también la información relevante para estas hipótesis, agrupándose todos los datos en forma de variables de información. También se deben identificar los enlaces que conectan las variables (C_BN1). La topología del DAG representa las relaciones cualitativas entre las variables. Entonces si una variable influencia a otra, ambas deberían estar conectadas por un enlace directo, donde la dirección indica el efecto. Sin embargo, en el mundo real estos enlaces no pueden conseguirse fácilmente, las relaciones causales no siempre son obvias y existen diferentes técnicas que ayudan en esta labor y complementan la opinión de los expertos (Spirtes, Glamour, & Scheines, 1991) y (Cooper & Herskovitzs, 1992).

La cuantificación (**C_BN2_Cuantificación**) consiste en la adquisición de los números necesarios para estimar las probabilidades condicionadas para el modelo de red Bayesiana. Métodos estadísticos para el aprendizaje de los parámetros en redes bayesianas se puede encontrar en (Buntine, 1996) y (Cestnik, 1900). Cuando se dispone de suficientes datos, la identificación de la estructura y la obtención de parámetros también se puede hacer de forma automática por los métodos de aprendizaje automático. Una descripción detallada de los métodos

de aprendizaje está fuera del alcance de este trabajo, pero (Buntine, 1996) ofrece una amplia revisión y (Heckerman, 1999) es una excelente introducción.

Por último, el propósito de **C_BN3_Validación** es comprobar la idoneidad del modelo de redes Bayesiana en el ámbito del trabajo que para el que ha sido diseñada, contestando a preguntas como *¿La estructura refleja las relaciones fundamentales de la independencia del problema? ¿Cuál es el nivel de exactitud conseguida en la predicción?* También es importante no olvidar medir la usabilidad y el rendimiento y saber si la red bayesiana cumple con los criterios uso de los clientes (Korb & Nicholson, 2003). La aplicación de estas actividades para el Arrendamientos se describe en la siguiente sección.

5.5.3 Arrendamientos con Redes Bayesianas

Las variables relevantes para la decisión de si una vivienda es asignable a un solicitante se obtiene a partir de los datos del dominio y de los servicios de información. Así, la lista de variables para Arrendamientos esta en ejemplo 12.

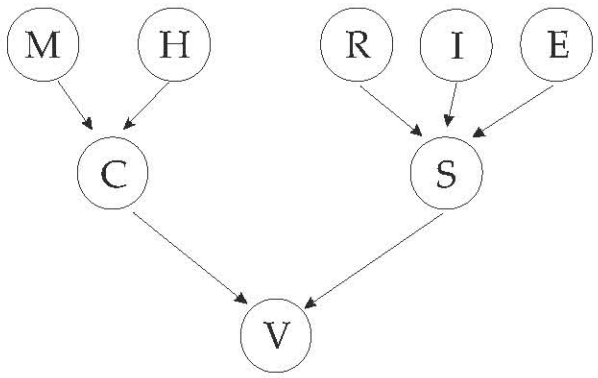
Ejemplo 12 Variables de Arrendamientos.

Descripción de la variable	Variable
Renta del arrendamiento	R
Ingresos del solicitante	I
Edad del solicitante	E
Número de miembros de la familia	M
Número de posibles habitantes	H
Capacidad adecuada de la vivienda	C
Solvencia económica	S
Vivienda asignable	V

Las relaciones entre las variables se pueden interpretar como relaciones causa-efecto. En este caso al tratarse de un caso sencillo las relaciones pueden interpretarse como dependencias funcionales. Si una variable influencia otra existe una conexión directa entre ellas, tomemos por ejemplo las variables *M*, *H* y *C*. Que una vivienda sea adecuada para un caso concreto (*C*) dependerá del número de miembros de la familia solicitante (*M*) y del número de habitaciones o posibles habitantes de esa vivienda (*H*). Así el modelo de red resultante se muestra en la figura 5.16, y es el resultado de la actividad C_BN1.

El paso siguiente es la cuantificación de los números necesarios para estimar las probabilidades condicionadas de C_BN2. Debido a la no disponibilidad de datos de partida, en el caso Arrendamientos se procede a la definición manual de estas probabilidades con ayuda de los expertos, en dos etapas. En primer lugar se valora y asignan probabilidades para el modelo durante entrevistas con los expertos, para después evaluar y revisar este modelo. A continuación, en segundo lugar, una vez obtenida la red se procede a revisar con los participantes en el proyecto Arrendamientos para comprobar su idoneidad. Este proceso manual es difícil, consume tiempo y es propenso a errores (Druzdel & van der Gaag, 1995), se pueden utilizar paquetes software para la creación y evaluación de redes bayesianas que ayuden a los desarrolladores y a los expertos (Elvira Consortium, 2002).

Ejemplo 13 Red Bayesiana para Arrendamientos.



5.6 Conclusiones del capítulo

Se han detallado las actividades fundamentales correspondientes a desarrollo orientado al cliente, que se desarrollan en paralelo en el ámbito de los requisitos para la InSo y en el ámbito de la obtención y representación del conocimiento para la InCo.

La actividad fundamental Requisitos se lleva a cabo mediante las actividades **R1_Adquirir los servicios**, **R2_Clasificar y analizar los servicios**, **R3_Especificar los servicios** y **R4_Validar los servicios**. Los métodos aplicados en estas actividades son los relacionados con la recogida y análisis de información, destacándose la utilización de los casos de uso y los guiones gráficos.

La actividad fundamental Conocimiento se lleva a cabo mediante las actividades **C1_Identificación**, **C2_Especificación** y **C3_Refinamiento**. Esta descomposición en actividades se aplica cuando se modela el conocimiento con el lenguaje propuesto por CK.

Al emplear otro mecanismo de modelado de conocimiento se deberán definir las actividades apropiadas. Se ha mostrado, a modo de ejemplo, como se instancia esta actividad fundamental en el caso de las redes Bayesianas, dando lugar a las siguientes actividades **C_BN1_Identificación**, **C_BN2_Cuantificación** y **C_BN3_Validación**.

Los artefactos principales resultantes son el **modelo de Servicios** (SER), con el documento asociado DOC_ERS y el **modelo de Conocimiento** (CON) bien se su versión CommonKADS o bayesiana.

El modelo de servicios se organiza en base a tres tipos de formularios: **SER_Función**, que caracteriza los servicios funcionales, **SER_no_funcional**, que representa las propiedades emergentes y **SER_Información**, que describe la información que debe almacenarse en el sistema y el **documento de especificación de los requisitos** (DOC_ERS).

Dependiendo el modelo de representación del conocimiento, el formato de CON es diferente. Para el caso de CK, este modelo contendrá la representación del conocimiento del dominio, de las inferencias y de las tareas. Para el caso de la una representación con redes bayesianas contendrá las variables, relaciones entre ellas y las distribuciones de probabilidad condicionada.

6 Desarrollo orientado al sistema

La actividad fundamental Análisis es aquella donde se finaliza el modelado conceptual desde el punto de vista de los desarrolladores y está estrechamente ligada al paradigma de desarrollo, siendo el punto de conexión con la implementación.

En la primera sección de este capítulo se justifica la necesidad del modelo de análisis que, al aplicar el paradigma orientado a objetos, extiende las clases de análisis propuestas por RUP con el estereotipo conocimiento, tal como se describe en la sección 6.2 "Clases de análisis". El resto del capítulo se estructura de forma similar a los dos anteriores, indicando las actividades de Análisis, los artefactos generados y las posibles técnicas a aplicar. Se debe que tener en cuenta que las propuestas hechas para la actividad Análisis son una primera aproximación que habrá que reconsiderar para la utilización de otros paradigmas, no sólo el orientado a objetos, ver capítulo 2.

6.1 Necesidad del modelo de ANÁLISIS

Los modelos conceptuales son el vínculo de unión entre el universo del discurso y el sistema a construir, si el problema modelado se resuelve con software, será un sistema software. Pero, cuáles son, cuántos y cuándo se desarrollan los modelos conceptuales en la propuesta InSCo.

En los orígenes de la utilización del principio del modelado siempre se definían dos grandes tipos de modelos: los modelos de análisis y los modelos de diseño. Los primeros evolucionaron hacia a la idea general del modelado conceptual y los modelos de diseño siguieron llamándose igual. La terminología

tradicional de la InSo puede llevar a confusión, pues por análisis se denomina tanto a la actividad InSCo Requisitos como a la InSCo Análisis. Para evitar este problema, en este trabajo de tesis se utiliza el término modelo conceptual, pero enfocado desde una perspectiva más cercana al cliente para Requisitos y otra más cercana al lenguaje del desarrollador para Análisis, que se han llamado respectivamente desarrollo orientado al cliente y desarrollo orientado al sistema.

En la actualidad, los modelos que se generan en el desarrollo de software siguiendo las recomendaciones marcadas por MDA son tres CIM, PIM y PSM. CIM se corresponde con el estudio del negocio y la actividad fundamental Organización. Como ya se ha indicado, la actividad fundamental Requisitos se ocupa de la obtención y documentación de los servicios, generando una parte del modelo conceptual orientado al problema o a la necesidad del cliente, se corresponde con los modelos PIM. La parte del modelado conceptual orientado al sistema software se realizará durante la actividad fundamental Análisis, que también entra dentro de los modelos PIM. PSM está fuera del nivel conceptual. Se aplica la esencia de las propuestas donde se independiza el modelado conceptual del paradigma de implementación (Dieste, Genero, Juristo, Mate, & Moreno, 2003), pero manteniendo el punto de conexión donde se describe el modelo conceptual en el lenguaje del desarrollador y supone una visión interna del sistema.

Otro de los aspectos a considerar es como se inyecta el conocimiento en el software a construir. El conocimiento se debe incorporar en el nivel conceptual de forma independiente de la plataforma o los detalles de implementación, lo que de nuevo viene a justificar la actividad fundamental de Análisis desde el punto de vista de la InCo. En resumen, el nivel conceptual debe ser independiente de la implementación, pero siempre es necesario un punto de conexión con los aspectos computacionales y en sistemas híbridos con más razón, para inyectar el conocimiento en el lugar oportuno, por lo que en el modelo de proceso InSCo se define la actividad fundamental ANÁLISIS con este objetivo.

Hay que tener en cuenta que, aunque los modelos en el nivel conceptual son independientes de la plataforma de implementación, no son independientes del paradigma o enfoque de desarrollo. Es decir, si se utiliza un enfoque orientado a objetos, el análisis se acercará al análisis del sistema orientado a objetos propuesto por Jacobson (Jacobson, Christerson, Jonsson, & Övergaard, 1992). Si se utiliza un enfoque estructurado, nos acercaremos a propuestas como

la de Yourdon (Yourdon, Análisis Estructurado Moderno, 1997). El modelo de análisis busca estructurar el sistema independientemente del entorno de implementación pero ligado a un paradigma de desarrollo concreto, que en el caso de esta tesis ha optado por el paradigma orientado a objetos.

6.2 Clases del modelo de Análisis

Un resultado importante de la actividad fundamental Análisis es el diseño preliminar de la arquitectura, entendiendo por arquitectura el conjunto de decisiones acerca de la organización del sistema software, los elementos estructurales que lo forman y su composición y las relaciones entre ellos. Los elementos estructurales serán de forma genérica las clases o grupos de clases que se relacionan siguiendo un determinado estilo arquitectónico.

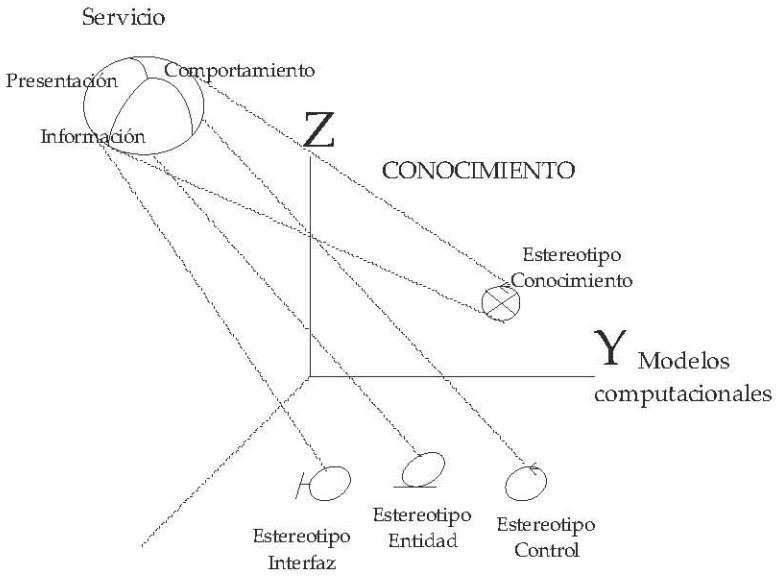


Figura 6.1 Estereotipos del modelo de análisis.

El espacio del modelo de análisis, extendiendo la figura 5.1 "Dimensiones del modelo conceptual" sobre la figura 6.1, está en los planos de la InSo y de la InCo, pero puesto que los sistemas software híbridos son software, es necesario capturar los tres aspectos básicos de cualquier sistema software, a saber: información, comportamiento y presentación. Esto se traslada en la utilización de estereotipos de las clases que representan estos aspectos. De esta forma, los casos de uso se describen en términos de los estereotipos: **entidad**, modelan la

información perdurable o persistente, capturando los datos; **control**, modelan la coordinación, secuencia, transiciones y control de flujo de información; **interfaz**, que modelan la interacción entre el sistema y sus actores; **conocimiento**, que modela las funcionalidades basadas en conocimiento (propuesto en este trabajo de tesis).

La orientación a objetos promueve el mapeo de los objetos del dominio sobre el sistema software y estos estereotipos infringen esta norma en cierta medida, pues promueven la utilización de clases estereotipadas, separando en lugar de encapsular. Muchos de los enfoques clásicos de orientación objetos sólo se han centrado en el aspecto información, extendiendo los objetos del mundo real al software, pero tanto Jacobson (Jacobson, Christerson, Jonsson, & Övergaard, 1992) como su heredero RUP (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999) han elegido utilizar estos estereotipos de objetos para de ser más adaptables a los cambios. Estos autores no creen que sean más estables los sistemas que se corresponden con entidades del mundo real, puesto que, por ejemplo, la presentación es dependiente del sistema de interfaz y si se ubica junto con una clase información, se hace depender la información del sistema de interfaz, lo que hace mas inflexible el sistema software, sobre todo pensando en su adaptación a nuevos sistemas de interfaz.

Esta aproximación es similar al enfoque MVC (Modelo-Vista-Controlador) originario de Smaltalk o al modelo PAC propuesto por (Coutaz, 1989), que distingue entre Presentación, Abstracción (aplicación) y Control, y al modelo de arquitectura propuesto por (Davis & Morgan, 1993) con los niveles de objetos de presentación, objetos proceso y objetos del negocio.

Pero en sistemas software híbridos tenemos la dimensión del conocimiento, y a los tres estereotipos típicos de la InSo se les añade el estereotipo Conocimiento. La funcionalidad no sólo puede derivarse a una clase entidad, control o interfaz, sino que también puede derivarse a una clase conocimiento, tal como vemos en el plano ZY de la figura 6.1. Además de las operaciones de interacción, persistencia de información y control, necesitamos ejecutar los procesos de inferencia, que pueden estar sustentados por datos persistentes o necesitar de interacciones.

Las clases de análisis InSCo se centran en el tratamiento de los servicios funcionales, posponiendo los no funcionales. En general tienen una granularidad

mayor que las clases de diseño e implementación y se organizan en torno a cuatro estereotipos básicos:

- **Entidad:** Modelan la información perdurable o persistente, capturando los datos. Modela la información y el comportamiento asociado a objetos del mundo real. Se derivan del dominio, son una versión redefinida de los objetos presentes en los servicios, pero sólo los objetos que son manejados por el sistema, ofreciendo a los desarrolladores un soporte para la implementación y el diseño. No tiene porque ser necesariamente pasivo ni sólo soportar la información, tendrá asociado el comportamiento relativo a la información que sustentan.

- **Control:** Unen los componentes para formar un caso de uso. Es un modelo de los cálculos y algoritmos complejos. Representa la lógica del negocio que no está en las entidades. Modelan la coordinación, secuencia, transacciones y control de otros objetos. Los aspectos dinámicos del sistema se almacenan en las clases control.

- **Interfaz**: Modelan la interacción entre el sistema y sus actores. Se suele asociar con la E/S. Traduce la acción del actor a los eventos del sistema. Representan abstracciones de ventanas, formularios, sensores, etc. Se asocian a un actor ya que cada actor tiene un conjunto de interfaces (IN-da datos, o pulsa botón, OUT-presenta datos). Las funcionalidades directamente dependientes del entorno en el que operará el sistema se colocan en objetos interfaz.

- **Conocimiento:** Representan una funcionalidad delegada al modelo de conocimiento. Es estrechamente dependiente del modelo de conocimiento elegido y su desarrollo depende de él. Se coloca en estas clases la funcionalidad que tiene una cierta similaridad con alguna porción del modelo de conocimiento ya desarrollado.

Se podía haber planteado que el estereotipo conocimiento también recogiese los tres aspectos, interfaz (toma de las demás clases aquello que necesita y lo lleva al modelo de conocimiento), entidad (soporta el modelo de conocimiento) y control (lleva a cabo la inferencia). Pero este enfoque no tiene porque ser válido en todas las formas de representar el conocimiento (no es igual la forma de actuar de una red bayesiana que la de un conjunto de reglas).

El modelo de servicios y de conocimiento se estructuran y mapean en el Modelo de Análisis. Cada caso de uso se divide en objetos de los estereotipos ya

descritos. En la práctica, esto se traduce en que el comportamiento especificado en los casos de uso debe asignarse y distribuirse entre distintos objetos, tal como se muestra en la figura 6.2. Esta asignación es difícil ya que es complejo clarificar los límites entre funcionalidades del sistema. La decisión final suele estar en manos del buen juicio de los desarrolladores. El flujo de trabajo de la actividad fundamental Análisis explica cómo encontrar estos estereotipos en los casos de uso.

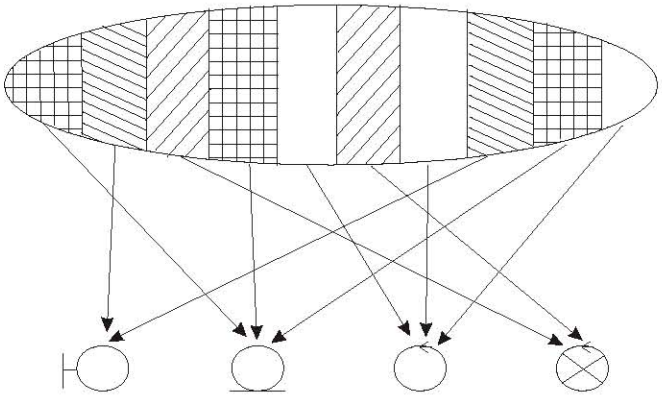


Figura 6.2 Distribución de comportamiento del caso de uso.

6.3 Flujo de trabajo

Esta actividad fundamental se ocupa de examinar y refinar los servicios y, al hacerlo, se logrará una comprensión detallada de los servicios, definiendo una estructura estable, robusta y mantenible del sistema. Alcanzando el hito LCA, donde se realiza la clasificación disjunta de las funcionalidades del sistema, este trabajo se plasma en el documento de definición de la arquitectura que, junto con las clases de análisis, forman el modelo de análisis ANA, tal como se muestra en la figura 6.3.

Tabla 6.1 Actividades de Análisis.

Análisis	
	A1 – Analizar los servicios (Casos de uso)
	Identificar Clases Entidad Identificar Clases Control Identificar Clases Interfaz
	A2 – Analizar las clases
	Identificar clases Conocimiento
	A3 – Refinar casos de uso.

La tabla 6.1 muestra la descomposición en actividades de la actividad fundamental Análisis. El modelo de servicios (SER) es el punto de partida de esta actividad fundamental. En SER se han definido lo que RUP llama clases entidad obvias y que son la propuesta inicial de las clases de análisis, en concreto, son los artefactos SER_Información que deben ser estudiados y refinados. Cada servicio es analizado identificando las clases cuyos objetos son necesarios para llevar a cabo el servicio (descrito como un caso de uso). El comportamiento mostrado en cada caso de uso se distribuye entre clases (**A1_Analizar_los_servicios**). Como resultado se obtendrán la propuesta de los estereotipos entidad, control e interfaz. En esta actividad también se esbozan las responsabilidades, atributos y relaciones entre las clases.

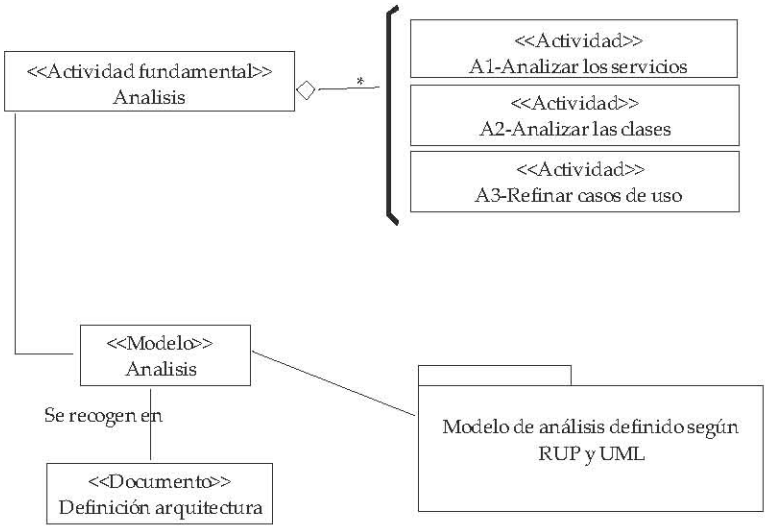


Figura 6.3 Actividad fundamental Análisis.

La asignación al estereotipo conocimiento del comportamiento se hace en la actividad **A2_Analizar_las_clases**. Se buscan similitudes entre el modelo de conocimiento, los casos de uso y las clases identificadas en A1. Esta medida de similaritud permitirá que el ingeniero del software y el ingeniero del conocimiento fijen cuales serán las clases conocimiento y qué comportamiento ofrecerán al resto del sistema. Para ejecutar sus responsabilidades, estas clases conocimiento emplearán parte o todo el modelo de conocimiento definido (CON).

En los desarrollos de SBC el modelo de conocimiento y sus mecanismos de inferencia eran codificados sobre componentes o módulos de software, las componentes basadas en conocimiento de los sistemas híbridos no son una excepción. CON debe transformarse a modelos de diseño utilizando por ejemplo arquitecturas MDA para automatizar la traducción a clases de diseño (Cañadas, Palma, & Túnez, 2009), o bien reutilizando jerarquías de clases en un lenguaje de programación ya implementadas con éxito que ofrecen los mecanismos de representación y realización de inferencias sobre redes Bayesianas (Elvira Consortium, 2002). El estereotipo conocimiento encapsula el comportamiento del modelo de conocimiento, no tomando en consideración los detalles en los que se profundizará en las actividades computacionales y que quedan fuera del Análisis.

Es necesario realizar una revisión de la robustez y coherencia de los cuatro tipos de clases, proceso que se realiza en la actividad **A3_Refinar_casos_de_uso**. En esta actividad se comprueba cómo se pueden realizar los casos de uso con las clases identificadas, haciendo los cambios y ajustes necesarios tanto en las clases como en los casos de uso.

6.3.1 Técnicas recomendadas

En el análisis de los casos de uso, la identificación de las clases entidad, interfaz y control, las técnicas más apropiadas son las marcadas por RUP (Jacobson, Booch, & Rumbaugh, The Unified Software Development Process, 1999) y Jacobson (Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, 2008) para su modelo de análisis.

Las clases entidad iniciales o también llamadas clases entidad obvias, vienen dadas desde el modelo de servicios. Hay que tener cuidado de no modelar demasiadas clases entidad porque, aunque se puede pensar que cuanta más información se maneje mejor, el límite lo deben marcar los casos de uso.

Partiendo de estas clases obvias como norma general ya propuesta por RUP se aplica el siguiente proceso:

- Identificar clases entidad revisando los casos de uso.
- Identificar una clase interfaz por cada actor.
- Identificar una clase interfaz para cada clase entidad previamente identificada.

- Identificar una clase control responsable de la coordinación y el tratamiento de control de cada caso de uso.

A partir de aquí se tienen que fijar las relaciones y la primera versión de los atributos y las responsabilidades. Para esto son muy útiles los diagramas de secuencia y colaboración que permitirán validar los escenarios de ejecución de los casos de uso en términos de las clases.

En general las técnicas aplicables en el análisis de RUP son aplicables aquí, basta con revisar la bibliografía al respecto, (Booch, Rumbaugh, & Jacobson, Unified Modeling Language User Guide, The (2nd Edition), 2005), (Rumbaugh, Jacobson, & Booch, The Unified Modeling Language Reference Manual (2nd Edition), 2004) y (Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, 2008).

La diferencia más importante aparece en la identificación del estereotipo conocimiento que representa la funcionalidad delegada al modelo de conocimiento. Se tendrá que estudiar la cercanía de cada elemento del modelo de servicios con las primitivas del modelo de conocimiento. Se deben buscar las correspondencias entre ellos y ofrecer una medida sobre qué servicios son los que tienen su reflejo en el modelo de conocimiento. Así, los desarrolladores fijarán, en base a esta información de similitud/distancia cuáles son las clases de estereotipo conocimiento. Desde un punto de vista formal diremos que una función de similitud $s_{i,j}$ entre dos elementos x_i, y_j o una distancia $d_{n,m}$ entre x_n, y_m, debe cumplir las propiedades que se muestran en la tabla 6.2, pudiendose definir una a partir de otra (Peña, 2002). A mayor similitud, más parecido. Por el contrario la distancia será mayor (siempre no negativa) cuando hay menos parecido.

Tabla 6.2 Similitud / Distancia.

	Similitud	Distancia
x_i, y_j	$0 \leq s_{i,j} \leq 1$ $s_{i,j} = s_{j,i}$ $s_{i,j} = 1$	$d_{i,j} = 2(1 - s_{i,j})$
x_n, y_m	$s_{n,m} = \frac{1}{1 + d_{n,m}}$	$d_{n,m} \leq 0$ $d_{n,m} = d_{n.m}$ $d_{n,n} = 0$ $d_{n,m} + d_{n,k} \leq d_{m,k}$

El problema aparece cuando los elementos a relacionar son de naturaleza diferente. Los índices de similitud más conocidos y utilizados, como son el coeficiente de Jaccard (Jaccard, 1901) o de Sørensen (Sørensen, 1948) que se utilizan en problemas de planteamiento similar al que nos ocupa como la recuperación de información (Manning, Raghavan, & Schütze, 2008) y (Singhal, 2001) y fusión de ontologías (Euzenat & Shvaiko, 2007), no se pueden extrapolar al problema de asignación de clases conocimiento.

Otra dificultad radica en es el lenguaje de modelado del conocimiento. Si se emplean redes bayesianas, las primitivas son las variables, con lo que habría que analizar donde aparecen estas variables, bien en los casos de usos, bien en las clases ya identificadas o en sus atributos o responsabilidades. Sin embargo, si se utilizan modelos CML, aparecen tres tipos de conocimiento: conocimiento del dominio, conocimiento de las inferencias y conocimiento de las tareas. Por lo tanto la pregunta que surge es ¿Con qué primitiva de conocimiento se calcula la similitud?

En este trabajo de tesis se plantea una propuesta de técnica a utilizar para la actividad, A2_Analizar_clases donde será necesario el uso de alguna medida de similitud, pero dejando fuera el estudio y propuesta de una medida concreta. Como resultado del estudio de la similitud se obtendrá una matriz de evaluación similar a la que se muestra en la figura 6.4, donde se confrontan el SER y CON. Las filas y las columnas serán respectivamente las primitivas del modelo de servicios (casos de uso o servicios de información), conceptos y/o clases ya identificadas y las primitivas del modelo de conocimiento (conceptos del dominio, relaciones entre conceptos, tareas, métodos, inferencias) representando cada posición de la matriz se tendrá que etiquetar con un índice de similitud. Este índice puede ser calculado, asignado automáticamente (análisis lingüístico de los modelos) o asignado de forma manual a un valor difuso, tal como proponen en el ámbito del alineamiento de ontologías (Fernández, Velasco, & López, 2010), en este último caso los valores asignados pueden ser: bajo, regular, medio, alto, muy alto. Con esta matriz se ofrece una primera aproximación a la identificación de que elementos están basados en conocimiento, y servirá para la posterior asignación de las clases conocimiento.

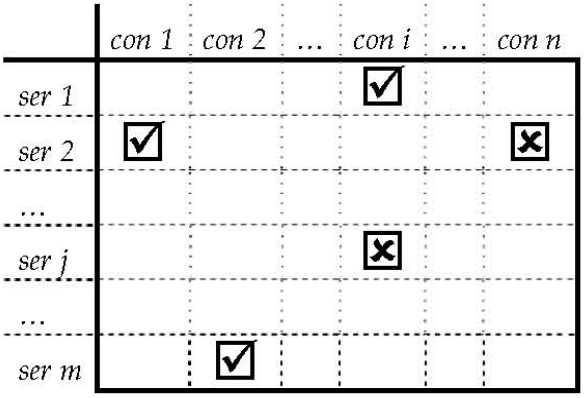


Figura 6.4 Matriz de similitud.

La actividad A3_Refinar_casos_de_uso, se centra principalmente en la verificación de que la distribución del comportamiento entre las clases cumple con las expectativas descritas en los escenarios de los servicios, distribución que tiene que ser consensuada entre los ingenieros del conocimiento y del software. Finalmente al ser los usuarios del futuro sistema software híbrido son los agentes de estos escenarios, son los encargados de validar el sistema completo, quedando oculto para ellos qué funcionalidad se sustenta sobre un modelo de conocimiento.

6.4 Artefactos generados por ANÁLISIS

Los artefactos generados por la ejecución de esta actividad fundamental están en el mundo de los desarrolladores y, por lo tanto, deben expresarse en un lenguaje propio de la InSo, siendo el contenido las clases conocimiento se expresado en términos de la InCo, dependiendo del mecanismo de representación del conocimiento utilizado.

Los artefactos de análisis constituyen el refinamiento de los modelos previamente elaborados. La figura 6.5 muestra como las clases de análisis se definen a partir de los artefactos generados en NEG, SER y CON. También se pueden generar artefactos intermedios de trabajo como las matrices de similitud.

Los artefactos generados suponen una vista interna del sistema que no debe tener redundancias e inconsistencias entre servicios o funcionalidades descritas. Constituye una primera indicación sobre cómo llevar a cabo esa funcionalidad dentro del sistema software híbrido a construir, siendo una primera aproximación al diseño. Las clases de conocimiento definidas en el

diseño deberán ser mapeadas sobre los elementos específicos de la herramienta software elegida como mecanismo de representación de conocimiento. Por ejemplo, si se utilizan redes bayesianas, el punto de partida en el diseño puede ser una herramienta al estilo de ELVIRA (Elvira Consortium, 2002). El sistema ELVIRA es una herramienta para construir tanto el modelo de redes bayesianas, como el SBC que lo utiliza. Está escrito en Java y dispone de métodos de inferencia, aprendizaje y toma de decisiones. Al estar su código disponible, es reutilizable sobre el sistema híbrido que se desarrollará en las siguientes actividades fundamentales del nivel computacional.

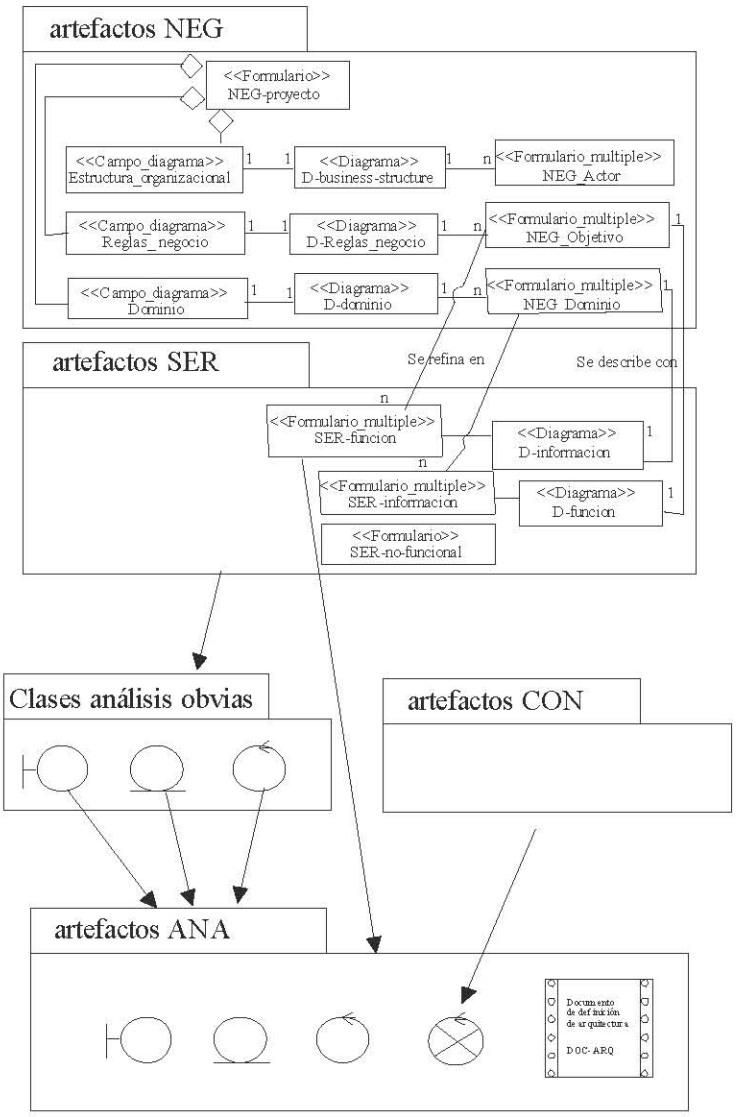
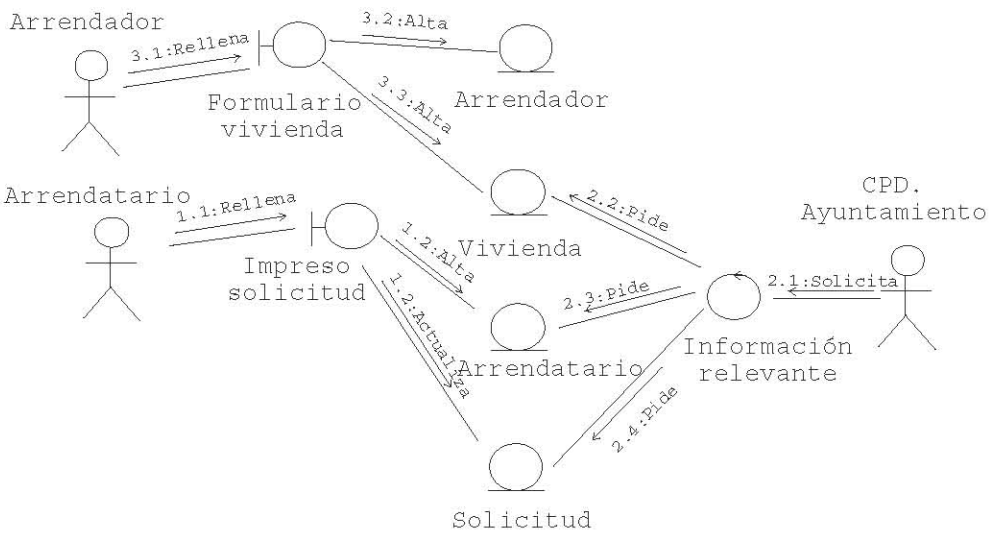


Figura 6.5 Artefactos del Modelo de análisis.

ANA se expresará en UML y los artefactos generados son los que se postula en este lenguaje, tal como se muestra en la figura 6.3. Tendremos diagramas del tipo estructural, para recogen los aspectos estáticos del sistema, donde el pilar son los diagramas de clase que pueden apoyarse en diagramas de objetos. También serán necesarios los diagramas de comportamiento para recoger los aspectos dinámicos que permitan expresar cómo se llevan a cabo los servicios (casos de uso) en términos de las clases y de sus objetos de. Para ellos se utilizarán diagramas de secuencia y colaboración de UML.

Ejemplo 14 Diagrama de clases obvias.

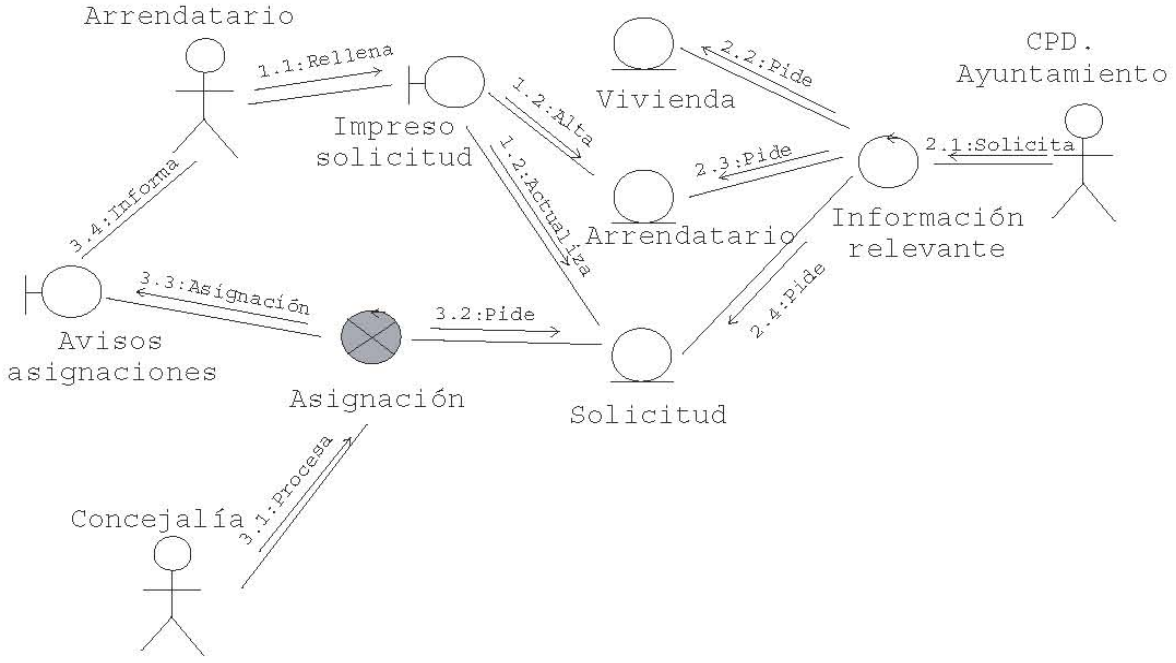


Para el proyecto Arrendamientos uno de los diagramas de clases obvias, mostrada en Ejemplo 14, donde se muestran tres escenarios de ejecución. A partir de la matriz de similitud, Ejemplo 15, podemos hacer una clasificación disjunta y definitiva de los servicios (casos de uso) y sus asignaciones a elementos basados o no en conocimiento. Ejemplo 16 muestra la primera una porción del diagrama de clases de análisis, con el estereotipo conocimiento representado en color oscuro.

Ejemplo 15 Similitud en Arrendamientos.

	Caso	*Min_num_ habitantes*	*Max_num_ habitantes*	*Renta acorde a ingresos*	*Método emparejar*
Asignación_vivienda					☑
Vivienda.Identificador	☑				
Vivienda.Dirección					
Vivienda.Renta					
Vivienda.nºHabitantes		☑	☑		
Vivienda.nºHabitaciones					
Vivienda.Referencia Catastral					
Vivienda.Gastos Comunes					
Registrar solicitudes					

Ejemplo 16 Diagrama de clases.



6.5 Documento de definición de arquitectura

El concepto de arquitectura es complejo. La literatura nos dice que una arquitectura de software describe cómo un sistema se descompone en componentes, como se interconexionan y cómo interactúan entre sí. Para una misma arquitectura existen varias perspectivas, pudiendo mostrar cada una de ellas distintas vistas dependiendo del punto de vista que interese. Normalmente la arquitectura se documenta a través de un conjunto de vistas, en donde cada una representa un aspecto o comportamiento particular del sistema. Kruchen (Kruchten, 1995) propone el modelo de vistas 4+1 donde: la *vista lógica* muestra los componentes principales y las relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada (diagramas de clases, secuencia y colaboración); la *vista de procesos* describe los procesos del sistema y cómo éste comunica con otros; la *vista de desarrollo* describe la estructura de módulos, archivos y paquetes en el sistema (diagramas de paquetes); la *vista física* describe como se instala y ejecuta el sistema en una red de computadores (diagramas de despliegue); y, por último la *vista de casos de uso* que describe la funcionalidad del sistema (diagramas de casos de uso) y que relaciona las otras vistas entre sí. Otros autores (Soni, Nord, & Hofmeister, 1995) proponen otros conjuntos de vistas similares, pero son comparables con el modelo 4+1.

El enfoque de Kruchten es el más utilizado, pues está estrechamente ligado a RUP, pero las nuevas tendencias buscan flexibilidad, no ajustándose a un número fijo de vistas y proponiendo la adaptabilidad a los intereses y las necesidades de los participantes en el proyecto. Esta nueva tendencia está respaldada por el SEI (Clements, y otros, 2002) y IEEE (Institute of Electrical and Electronics Engineers Staff, 2000), que plantean un proceso de identificación de cuáles son las vistas apropiadas en cada proyecto. Este tema queda fuera del trabajo de tesis, lo que sí cabe decir es que en sistemas software híbridos es preciso estudiar la necesidad de documentar el punto de vista de los ingenieros del conocimiento con lo que podríamos llamar *vista conocimiento*, que describa los componentes del modelo de conocimiento que se llevarán al software utilizando para ello un lenguaje apropiado para su documentación.

No obstante, vamos a realizar una propuesta de documento de definición de arquitectura, tal y como se puede ver en la figura 6.7 y que, siguiendo las

tendencias actuales, no es más que una plantilla inicial adaptable a cada tipo de proyecto. La plantilla propuesta es una adaptación de la propuesta del SEI[16] (Clements, y otros, 2002). En este documento se definen los puntos de vista para la descripción de la arquitectura siguiendo las nuevas tendencias en este campo. Si se utiliza el modelo 4+1, el apartado 3.0 del documento tendrá cinco subapartados.

Portada
Tipo del documento
Nombre del proyecto
Versión
Fecha.
Equipo de desarrollo
Cliente
Lista de cambios
Listas de figuras y tablas

1.0 Introducción
    1.1 Propósito y Alcance
    1.2 Estructura del documento
2.0 Representación de la arquitectura
    2.1 Descripción de los puntos de vista
3.0 Vistas
  `<para cada punto de vista X>`
    3.X.1 Vista general
    3.X.2 Descripción de cada componente
4.0 Relaciones entre vistas
    4.1 Relaciones generales entre vistas
    4.2 Relaciones entre cada dos vistas
5.0 Material de referencia

Figura 6.6 Formato del documento de definición de arquitectura.

[16] http://www.sei.cmu.edu/architecture/tools/viewsandbeyond/index.cfm

6.6 Conclusiones del capítulo

En este capítulo, se han descrito y justificado la necesidad del modelo de Análisis en el que se realice la clasificación disjunta de los servicios como basado o no en conocimiento, materializándose en la ejecución de la actividad fundamental Análisis.

La descripción de actividades, artefactos y técnicas de Análisis se realiza de forma somera y puede ser considerada como un conjunto de recomendaciones a seguir, que habrá que desarrollar en profundidad en trabajos futuros.

Se propone que la actividad fundamental Análisis se lleve a cabo mediante las actividades **A1_Analizar los servicios**, **A2_Analizar las clases**, **A3_Refinar casos de uso**. Estas actividades tienen por objetivo distribuir el comportamiento especificado en los casos de uso entre las diferentes clases que se estereotipan siguiendo las pautas RUP: clases interfaz, clases entidad y clases control. Finalmente, se añade un nuevo estereotipo formado por las clases conocimiento, que encapsulan el comportamiento basado en conocimiento del sistema software híbrido.

Las técnicas aplicadas son los propuestos en RUP, proponiéndose la utilización de matrices de similitud para la identificación de las clases conocimiento.

El artefacto resultante es el **modelo de Análisis (ANA)** que se documenta mediante un conjunto de vista recogidas en el **Documento de definición de la arquitectura**.

7 Conclusiones y trabajos futuros

7.1 Conclusiones generales y principales aportaciones

En este trabajo se ha definido el **modelo de proceso InSCo**, como un nuevo punto de unificación hacia una metodología común para el desarrollo de los sistemas híbridos, considerados éstos cómo aquellos sistemas en los que coexisten componentes que intentan dar solución a las partes del problema en las que los requisitos están poco estructurados y son subjetivos y en las que además están presentes la incertidumbre, la incompletitud y la inconsistencia (componentes basadas en conocimiento) y componentes que suponen una solución válida para las partes del problema cuyos requisitos son objetivos, pudiendo tener un resultado cuantitativo o cualitativo (componentes no basadas en conocimiento).

Una de las hipótesis de partida es qué problema que se plantea en esta tesis, es la construcción de software para sistemas híbridos, que no se resuelve con la definición de una nueva metodología debido al alto coste que supone el cambio de la misma. Los profesionales de la InCo y la InSo son ya usuarios de alguna metodología, y trasladarse a un nuevo enfoque no es una propuesta viable para casi nadie. Por tanto, la solución debe venir de la mano de extender, mejorar e integrar metodologías conocidas haciendo converger las metodologías usadas por ambas disciplinas.

En cada capítulo se describe una parte del trabajo realizado y las principales aportaciones de cada uno de ellos son:

- Se hace una propuesta global de una metodología integrada para InCo e InSo que abarca todo el proceso de desarrollo, pero en el capítulo 1 justifica porque se centra el trabajo de investigación en describir detalladamente las actividades del nivel conceptual

- En capítulo 2 se han estudiado los puntos comunes de las metodologías de la InCo y de la InSo, seleccionando como punto de partida tres de las metodologías: Estructurada, Rational Unified Process y CommonKADS. De esta forma InSCo podrá ser adoptado tanto por los ingenieros del software como por los ingenieros del conocimiento experimentados en ellas.

- Del estudio de los puntos comunes de las metodologías se llega a la conclusión de que el modelo de proceso InSCo debe ser un modelo flexible que emplea el concepto de actividad fundamental más general y adaptable. El modelo aúna características de las metodologías previas, siendo las características deseables de la propuesta InSCo que el modelo de proceso sea genérico y flexible, que se centre en la definición de actividades fundamentales, que sea iterativo e incremental, que se defina en el nivel de conocimiento y se centre en la definición de modelos, que separe los modelos conceptuales de los computacionales, que fomente la reutilización, que incluya el modelado del negocio y los requisitos y, por último, que minimice la notación.

- El modelo de proceso InSCo ha sido definido en el capítulo 3 a través de las actividades fundamentales: **O**rganización, **R**equisitos, **C**onocimiento, **A**nálisis, **D**iseño e **I**mplementación. Estas actividades suponen una cobertura de las metodologías soporte El principal artefacto generado durante la ejecución del modelo de proceso InSCo es un conjunto de modelos: Modelo de **Neg**ocio, Modelo de **Ser**vicios, Modelo de **Con**ocimiento, Modelo de **Aná**lisis, Modelo de **Dis**eño, Modelo de **Imp**lementación y Modelo de **Pru**eba.

- Las actividades fundamentales se descomponen en actividades y estas a su vez en tareas. Los modelos son una agregación de otros artefactos, que en el nivel conceptual son básicamente los formularios y los diagramas (pueden heredarse de lenguajes como UML).El conocimiento es un recurso crítico para la organización y se modela independientemente de que la solución venga de la mano del software.

- El modelo de conocimiento representa el conocimiento de una empresa u organización y pueden emplearse como medio para compartir experiencias o con fines de formación. Las actividades del nivel conceptual se definen independientes de la implementación, pero existe un punto de conexión con la implementación que es el modelo de Análisis, de especial importancia porque es el punto donde se inyecta el conocimiento en las funcionalidades del software.

- El nivel del estudio del negocio, capítulo 4, que tiene lugar durante Organización, equivale al análisis de sistemas en ingeniería de sistemas. Como resultado se genera el modelo de Negocio que contiene formularios y diagramas que recogen los objetivos de la organización y los conceptos del dominio y sus relaciones.

- Las actividades fundamentales Requisitos y Conocimiento se desarrollan en paralelo, como se describe en el capítulo 5. Los métodos aplicados en Requisitos son los relacionados con la recogida y análisis de información, se destaca la utilización de los casos de uso y los guiones gráficos. La actividad fundamental Conocimiento depende del mecanismo de representación de conocimiento elegido, en este trabajo se instancia esta actividad para el modelado de conocimiento con CML y con redes Bayesianas Los artefactos principales resultantes son el modelo de Servicios (SER), con el documento asociado DOC_ERS y el modelo de Conocimiento (CON) que depende del modelo de representación del conocimiento.

- La actividad fundamental Análisis, descrita en el capítulo 6, es donde se inyecta el conocimiento en las clases. Las propuestas hechas para la actividad Análisis son una primera aproximación en la que habrá que profundizar en trabajos futuros.

Por último y de forma general, consideramos que el modelo de proceso InSCo reduce los errores y vueltas atrás provocadas por asignaciones demasiado tempranas de un conjunto de funcionalidades a la categoría de software basado o no en conocimiento. InSCo permite posponer la decisión del tipo de técnicas y métodos a utilizar, hasta que se puedan aplicar criterios concretos que nos hagan optar por una o por otra. El proyecto software puede ejecutar sus primeras etapas manteniendo esta incertidumbre, con lo que se mantiene un marco común de referencia para los dos tipos de componentes software de naturaleza diferente.

7.2 Trabajos futuros

Se han abierto varios temas de investigación a partir de los resultados de esta tesis, entre los que destacamos:

- Completar la metodología con la inclusión de aspectos computacionales, en la representación mediante CML y redes Bayesianas, de forma que se realicen desde el modelo de análisis propuestas de diseño de forma automática.

- Desarrollar con profundidad las técnicas y actividades de la actividad fundamental Análisis, en esta tesis sólo se hace descripción somera.

- Completar InSCo con aspectos de gestión del proyecto y estimaciones, que históricamente ha sido los grandes olvidados de la InCo, pero con un gran peso en InSo.

- Validar y registrar el conjunto de herramientas InSCo ya construidas en entornos de proyectos de software híbrido reales, extendiendo el trabajo presentado en (Cañadas, Orellana, Águila, Palma, & Tunez, 2009).

- Implantar InSCo en entornos distribuidos, gracias a las herramientas Web desarrolladas, mejorando así la comunicación entre los miembros de los equipos de desarrollo.

- Explorar otras posibilidades de colaboración entre las disciplinas de la InCo y de la InSo, extendiendo la utilización de las técnicas de IA en la mejora del proceso InSo.

- Reformular problemas propios de la InSo como problemas de búsqueda que puedan ser resueltas con técnicas de IA.

7.3 Publicaciones relacionadas

En esta sección se realiza una revisión de las publicaciones relacionadas con este trabajo de tesis.

- El trabajo **Túnez, S., Águila, I. M., & Marín, R. (2001). An expertise model for therapy planning using abductive reasoning.** ***Cybernetics and Systems: An International Journal, 32*** **(8), 829 – 849**, supuso el inicio en el desarrollo práctico de los Sistemas Basados en conocimiento. Impact factor 0,367.
- Una primera colaboración entre InSo e InCo aparece en el trabajo: **Águila, I. M., Túnez, S., Cañadas, J., Bosch, A., & Marín, R. (2001). A proposal for Project Management Using CommonKADS. En Lecture Notes in Computer Science (Vol. 2178, págs. 160-171). Springer Berlin / Heidelberg**. En él se dota de un mecanismo de gestión a los proyectos de software basado en conocimiento que aplican CommonKADS como metodología de desarrollo de software. Se define la utilización de hitos o puntos de anclaje en el proyecto que doten de estabilidad al trabajo realizado por los desarrolladores. Impact factor 0,415.
- **Cañadas, J., Águila, I. M. D., Bosch, A., & Túnez, S. (2002). An Intelligent System for Therapy Control in a Distributed Organization.** ***Lecture Notes in Computer Science. EurAsia-ICT 2002: Information and Communication Technology*** **(Vol. 2510/2002, págs. 19-26). Springer Berlin / Heidelberg.** describe un sistema para la planificación de terapia aplicando la metodología CommonKADS. En estos dos trabajos se pusieron de manifiesto las primeras dificultades prácticas para aunar los objetivos de la InSo y de la InCo. A la hora de escribir el software no fue fácil utilizar los principios de ambas ingeniería. Impact factor 0,515.
- El trabajo **Águila, I. M., Cañadas, J. J., Bosch, A., Túnez, S., & Marín, R. (2003). Knowledge Model of a Therapy Administration Task - Applied to an Agricultural Domain. En Knowledge-Based Intelligent Information and Engineering Systems. LNCS (Vol. 2774, págs. 1227-1283). Springer Berlin / Heidelberg**. Describe la aplicación práctica de la metodología CommonKADS para el desarrollo del modelo de conocimiento, definiendo la una propuesta de arquitectura previa al diseño del sistema. Indexado DBLP.
- A partir de este punto se empezó a trabajar en el proceso de unificación y colaboración estrecha las dos disciplinas. El trabajo **Águila, I. M., Cañadas, J., Palma, J. T., & Túnez, S. (2003). Integration of development methodologies for the building of knowledge intensive multiagent systems. Integration of Knowledge Intensive Multi-Agent Systems, 2003. International Conference**

on (págs. 203-208). IEEE, se plantea la integración en el ámbito de los sistemas multiagente.

- El trabajo **Orellana, F. J., Guil, F., Águila, I. M., & Túnez, S. (2005). A WEB-CASE Tool Prototype for Hybrid Software Development. En LNCS Computer Aided Systems Theory – EUROCAST 2005 (Vol. 3643, págs. 217-222). La Laguna.** Es la primera version de una herramienta WEB que en permite a los desarrolladores aplicar las etapas iniciales InSCo. Impact factor 0,402.
- Ya como metodología completa el trabajo **Águila, I. M., Cañadas, J., Palma, J., & Túnez, S. (2006). Towards a Methodology for Hybrid Systems Software Development. International Conference of Software Engineering and Knowledge Engineering, (págs. 188-193)**, marca las líneas básicas a seguir en la propuesta InSCo. Indexado en DBLP.
- El capítulo de libro **Águila, I. M., & Túnez, S. (2007) Caso 5. El modelado del conocimiento en un sistema de asesoramiento experto en producción integrada en agricultura. En Paniagua, E (coordinador), Lopez, B., Martin, F., Campos, M., Carceles, A., Rodriguez, A., Palma, J., Flores, M, Jimenez, R., Fernandez, J., Martínez, R., Botia, J., Hernansaez, J., Águila, I.M. & Túnez, S., La gestión tecnológica del conocimiento (págs. 287-320) Universidad de Murcia, Servicio de Publicaciones.** Recoge el modelado de conocimiento realizado para un sistema de ayuda a la toma de decisiones par el control fitosanitario.
- El trabajo **Sagrado, J., & Águila, I. M. (2007). Olive Fly Infestation Prediction Using Machine Learning Techniques. En *Current Topics in Artificial Intelligence* (Vol. 4788, págs. 229-238). Springer** supone la primera aproximación al modelado del conocimiento utilizando redes bayesianas. Indexado DBLP.
- El trabajo **Orellana, F. J., Cañadas, J., Águila, I. M., & Túnez, S. (2008). INSCO Requisite - A Web-Based RM-Tool to support Hybrid Software Development.** *Proceedings of the Tenth International Conference on Enterprise Information Systems ICEIS (3-1),* **(pp. 326-329). Barcelona.** Muestra el primer nivel de la suite de herramientas para InSCo. Indexado DBLP.
- El trabajo **Cañadas, J. J., Orellana, F. J., del Aguila, I., Palma, J., & Tunez, S. (2009). A Tool Suite for Hybrid Intelligence Information Systems. XIII Conferencia de la Asociación Española para la Inteligencia Artificial, (págs. 147-156). Sevilla**, se ha especificado y construido un prototipo de lo que sería una suite de herramientas para la implantación del modelo de proceso InSCo.

- Este trabajo reciente es un ejemplo de los buenos resultados que pueden darse cuando colaborar de forma sinérgica InCo e InSo .**Sagrado, J. d., & Águila, I. M. (2009). A Bayesian Network for Predicting the Need for a Requirements Review. En F. Meziane, & S. Vadera, Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects (págs. 106-128). IGI Global**
- El registro de la propiedad intelectual de **InSCo Requisite**, del que son autores y titulares **Bosch, A., Cañadas, J. J., Águila, I. M., Guil F., Orellana F., & Túnez, S.** Esta herramienta software permite la aplicación de InSCo en sus primeras etapas y tiene el **Número de asiento registral 04/2010/90**
- La propuesta mostrada en **Sagrado, J. del., & Águila, I. M. del. (2009). Ant Colony Optimization for Requirement selection in Incremental Software development. En *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09)*. IEEE Computer Society,** muestra como abordar el problema de selección de requitos con técnicas de búsqueda.
- El trabajo **Águila, I. M., Sagrado, J. del, Túnez, S., & Orellana, F. J. (2010) Seamless Software Development For Systems Based on Bayesian Networks. An agricultural pest control system example** ha sido aceptado en ***5th International Conference on Software an Data Technologies, ICSOFT 2010*** y está pendiente de la celebración del congreso en Julio de este año. En él se describe un modelo de proceso donde las componentes basadas en conocimiento se abordan con el modelado basado en redes bayesianas. Indexado DBLP.

También cabe apuntar en esta sección que se han enviado y están pendientes de la evaluación final varios trabajos:

- "Ant Colony Optimization for Requirement selection in Incremental Software development", enviado a *Special Issue of Software—Practice and Experience: Practical Aspects of Search-Based Software Engineering*, que ha pasado el primer proceso de evaluación.
- "Requirement risk level forecast using bayesian networks classifiers" enviado a IJSEKE Featured Topic Issue: *Software Risk Assesment, International Journal of Software Engineering and Knowledge Engineering.*
- "Ant Colony Optimization for the Next Release Problem. A comparative study". Enviado al *2st International Symposium on Search Based Software Engineering (SSBSE '10).*

Referencias

1. Abran, A., Moore, J. W., Bourque, P., Dupuis, R., & Tripp, L. (2004). *Guide to the Software Engineering Body of Knowledge.* Los Alamitos: IEEE Computer Society.
2. Acuña, S. T., Lopez, M., Juristo, N., & Moreno, A. (1999). A process model applicable to software engineering and knowledge engineering. *International Journal of Software Engineering and Knowledge Engineering , 9* (5), 663-687.
3. Acuña, S. T., López, M., Mate, J. L., Ferre, X., & Chang, S. K. (2002). The software process: Modelling, evaluation and improvement. Pittsburg: World Scientific Publishing Company.
4. Águila, I. M., & Túnez, S. (2007). Caso 5. El modelado del conocimiento en un sistema de asesoramiento experto en producción integrada en agricultura. En E. Paniagua, B. Lopez, F. Martin, M. Campos, A. Carceles, A. Rodriguez, y otros, *La gestión tecnológica del conocimiento* (págs. 287-320). Murcia: Universidad de Murcia, Servicio de Publicaciones.
5. Águila, I. M., Cañadas, J. J., Bosch, A., Túnez, S., & Marín, R. (2003). Knowledge Model of a Therapy Administration Task - Applied to an Agricultural Domain. In *Knowledge-Based Intelligent Information and Engineering Systems. LNCS* (Vol. 2774, pp. 1227-1283). Springer Berlin / Heidelberg.
6. Águila, I. M., Cañadas, J., Palma, J. T., & Túnez, S. (2003). Integration of development methodologies for the building of knowledge intensive multiagent systems. *Integration of Knowledge Intensive Multi-Agent Systems, 2003. International Conference on* (pp. 203-208). IEEE .
7. Águila, I. M., Cañadas, J., Palma, J., & Túnez, S. (2006). Towards a Methodology for Hybrid Systems Software Development. *SEKE*, (pp. 188-193).
8. Águila, I. M., Sagrado, J. d., Túnez, S., & Orellana, F. J. (Julio 2010 - Aceptado pendiente celebración). Seamless Software Development for Systems Based on Bayesian Networks. An agricultural pest control system example. *5th International Conference on Software and Data Technologies.* Atenas.

9. Águila, I. M., Túnez, S., Cañadas, J., Bosch, A., & Marín, R. (2001). A proposal for Project Management Using CommonKADS. In *Lecture Notes in Computer Science* (Vol. 2178, pp. 160-171). Springer Berlin / Heidelberg.
10. Akkermans, H., van Harmelen, F., Schreiber, G., & Wielinga, B. (1993). A formalization of knowledge-level models for knowledge acquisition. In *Knowledge acquisition as modeling* (pp. 169–208.). New York: John Wiley & Sons, Inc.
11. Alford, M. W. (1977). A Requirements Engineering Methodology for Real-Time Processing Requirements. *IEEE Transactions on Software Engineering , SE-3* (1), 60-69.
12. Alonso, F., Antonio, A. d., Gonzalez, A. L., Fuertes, J. L., & Martinez, L. (1998). Towards a unified methodology for software engineering and knowledge engineering. *IEEE International Conference on Systems, Man, and Cybernetics, 1998., 5*, pp. 4890-4895.
13. Alonso, F., Fuertes, J. L., Martinez, L., & Montes, C. (2000). An incremental solution for developing knowledge-based software: its application to an expert system for isokinetics interpretation. *Expert Systems with Applications , 18* (3), 165-184.
14. Alonso, F., Juristo, N., & Pazos, J. (1995). Trends in Life-cycle Models for SE and KE proposal for Spiral-Conical life-cycle Approach. *International journal of Software Engineering and Knowledge Engineering , 5* (3), 445-465.
15. Alonso, F., Juristo, N., Mate, J. L., & Pazos, J. (1996). Software engineering and knowledge engineering: Towards a common life cycle. *Journal of Systems and Software , 33* (1), 65-79.
16. Angele, J., Fensel, D., Landes, D., & Studer, R. (1998). Developing Knowledge-Based Systems with MIKE. *Automated Software Engineering , 5* (4), 389-418.
17. Anjewierden, A., & Schreiber, G. (1994). *CML2 syntax (2.2.1).* Amsterdam: University of SWI.
18. Batini, C., Ceri, S., & Navathe, S. B. (1991). *Conceptual Database Design: An Entity-Relationship Approach.* Addison Wesley.
19. Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace change.* Addison-Wesley Professional.
20. Berners, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American* , Mayo.
21. Blum, B. I. (1996). *Beyond Programming: To a New Era of Design.* Oxford University Press, USA.
22. Boehm, B. (1996). Anchoring the software process. *Software, IEEE , 13* (4), 73-82.

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

24. Boehm, B. W. (1991). Software risk management: principles and practices. *Software, IEEE , 8* (1), 32-41.

25. Boehm, B., Bose, P., Horowitz, E., & Lee, M.-J. (1994). Software requirements as negotiated win conditions. *Requirements Engineering, 1994., Proceedings of the First International Conference on*, (pp. 74-83).

26. Booch, G. (1993). *Object-Oriented Analysis and Design with Application.* Benjamin-Cummings Publishing Company.

27. Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide.* Addison-Wesley Professional.

28. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified Modeling Language User Guide, The (2nd Edition).* Addison-Wesley Professional.

29. Braude, E. J. (2005). *Ingenieria de Software: Una Perspectiva Orientada A Objetos.* Alfaomega Grupo Editor.

30. Braude, E. J. (2000). *Software Engineering: An Object-Oriented Perspective* (1 ed.). Wiley.

31. Breuker, A. J., & Velde, W. V. (1994). *CommonKADS Library for Expertise Modelling: Reusable Problem Solving Components.* Netherlands: Ioss Press Inc.

32. Briand, L. C. (2002). On the many ways software engineering can benefit from knowledge engineering. *Proceedings of the 14th international conference on Software engineering and knowledge engineering* (pp. 3-6). Ischia, Italy: ACM.

33. Brinkkemper, S., Lyytinen, K., & Welke, R. (1996). *Method Engineering - Principles of method construction and tool support* (1 ed.). Springer.

34. Buchanan, B. G., Barstow, D., Bechtal, R., Bennett, J., Clancey, W., Kulikowski, C., y otros. (1983). Constructing an Expert System. En F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Edits.). London: Addison-Wesley.

35. Budd, T. (1994). *Introducción a la programación orientada a objetos.* Addison-Wesley Iberoaméricana.

36. Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering , 8*, 195-210.

37. Cañadas, J. J., Orellana, F. J., Águila, I., Palma, J., & Tunez, S. (2009). A Tool Suite for Hybrid Intelligence Information Systems. *XIII Conferencia de la Asociación Española para la Inteligencia Artificial*, (pp. 147-156). Sevilla.

38. Cañadas, J., Águila, I. d., Bosch, A., & Túnez, S. (2002). An Intelligent System for Therapy Control in a Distributed Organization. En *Lecture Notes in*

Computer Science. EurAsia-ICT 2002: Information and Communication Technology (Vol. 2510/2002, págs. 19-26). Springer Berlin / Heidelberg.

39. Cañadas, J., Palma, J., & Túnez, S. (2009). InSCo-Gen: A MDD Tool for Web Rule-Based Applications. En Web Engineering. In *Web Engineering. Lecture Notes in Computer Science* (Vol. 5648/2009, pp. 523-526). Springer Berlin / Heidelberg.
40. Cestnik, B. (1900). Estimating probabilities: A crucial task in Machine Learning. *Proc. Of 9th the European Conference on Artificial Intelligence,* (pp. 147-149). Stockholm, Sweden.
41. Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert , 1* (3), 23-30.
42. Chandrasekaran, B., & Johnson, T. R. (1993). Generic tasks and task structures: history, critique and new directions. Springer-Verlag New York, Inc.
43. Chang, S. K. (2002). *Handbook of Software Engineering and Knowledge Engineering, Vol 2 Emerging Technologies* (1st ed., Vol. 2). World Scientific Publishing Company.
44. Chang, S. K. (2002). *Handbook of Software Engineering and Knowledge Engineering, Volume 1* (1st ed., Vol. 1). World Scientific Publishing Company.
45. Chen, P. P. (1983). *Entity Relationship Approach.* Elsevier.
46. Chen, P. P. (1991). *The entity-relationship approach to logical database design.* QED Information Sciences.
47. Chen, X., Kendal, S., Pott, I., & Smith, P. (1996). The Development of Hybrid Information Systems: a Process Model an Methodology. *International Coference of Software Engineering and Knowlede Engineering,* (pp. 51-58). Nevada, USA.
48. Chen, X., Kendal, S., Potts, I., & Smith, P. (1997). Towards an integrated method for hybrid information system development. *IEE Proc., Softw. Eng., 144 (5-6),* pp. 261-269.
49. Clements, P., Bachmann, F., B. L., Garlan, D., Ivers, J., Little, R., y otros. (2002). *Documenting Software Architectures: Views and Beyond.* Addison-Wesley Professional.
50. Coad, P., & Yourdon, E. (1989). *Object-Oriented Analysis.* Prentice Hall PTR.
51. Coad, P., & Yourdon, E. (1991). *Object-Oriented Design* (Facsimile ed.). Prentice Hall PTR.
52. Coad, P., North, D., & Mayfield, M. (1996). *Object Models: Strategies, Patterns, and Applications (2nd Edition)* (2 ed.). Prentice Hall PTR.
53. Cockburn, A. (2001). *Agile Software Development.* Addison-Wesley Professional.

54. Cockburn, A. (2006). *Agile Software Development: The Cooperative Game.* Addison-Wesley Professional.
55. Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software , 17* (4), 64-71.
56. Cockburn, A. (2000). *Writing Effective Use Cases.* Addison-Wesley Professional.
57. Coleman, D. D., Arnold, P., Bodoff, S., Dollin, C., & Gilchrist, H. (1993). *Object-Oriented Development: The Fusion Method* (1st ed.). Prentice Hall.
58. Consejo_Superior_de_Administración_Electrónica. (2010). *MÉTRICA. VERSIÓN 3.* Retrieved 2010 йил 10-03 from http://www.csi.map.es/csi/metrica3/
59. Cooper, G., & Herskovitzs, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning , 9*, 309-347.
60. Coutaz, J. (1989). Architecture Models for Interactive Software. *ECOOP*, (pp. 383-399).
61. Cowell, R. D. (1999). *Probabilistic Networks and Expert Systems.* Springer-Verlag.
62. Davis, J., & Morgan, T. (1993). (1993). Object-Oriented Development at Brooklyn Union Gas. *IEEE Software , 10* (1), 67–74.
63. DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering , 11* (3), 231-258.
64. Demarco, T. (1979). *Structured Analysis and System Specification* (Facsimile ed.). Prentice Hall PTR.
65. Derniame, J.-C., Kaba, B. A., & Warboys, B. (1999). The Software Process: Modelling and Technology. En C. Montangero (Ed.). Berlin/ Heidelberg: Springer-Verlag.
66. Dieste, O. (2001). Development-Paradigm Independent Conceptual Models. *International Conference on Software Engineering and Knowledge Engineering*, (pp. 137-144).
67. Dieste, O., Genero, M., Juristo, N., Mate, J. L., & Moreno, A. M. (2003). A conceptual model completely independent of the implementation paradigm. *Journal of Systems and Software , 68* (3), 183-198.
68. Dieste, O., Juristo, N., Moreno, A. M., Pazos, J., Chang, S. K., & Sierra, A. (2002). Conceptual modelling in software engineering and knowledge engineering: concepts, techniques and trends. Pittsburg: World Scientific Publishing Company.
69. Dijkstra, E. (1976). Structure programming. In J. Buxton, & J. Buxton (Ed.). Van Nostrand-Reinhold.

70. Druzdel, M., & van der Gaag, L. (1995). Elicitation of Probabilities for Belief Networks: Combining Qualitative and Quantitative Information. *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence* (págs. 141-148). San Francisco: Morgan Kauffmann.

71. Duda, R. O., & Reboh, R. (1984). AI and decision making: The Prospector experience. In W. Reitman (Ed.), *Artificial Intelligence: Applications for Business* (pp. 111-147). N.J.: Ablex. Norwood.

72. Duran, A. (2000). *Un Entorno Metodológico de Ingeniería de Requisitos.* Sevilla: Phd. Tesis.

73. Duran, A., & Jiménez, B. B. (2001). *Metodología de análisis de requisitos de sistemas software. Versión 2.1.* Departamento de lenguajes y sistemas informáticos. Universidad de Sevilla.

74. Duran, A., & Jiménez, B. B. (2000). *Metodología de elicitación de requisitos de sistemas software. Versión 2.1.* Departamento de lenguajes y sistemas informÃ¡ticos. Universidad de Sevilla.

75. Duursma, C., Olsson, O., & Sundin, U. (1993). *Taks model definition and task analysis process.* Brussels: Free University Brussels.

76. Elvira Consortium. (2002). Elvira: An Environment for Probabilistc Graphical Models. *Proceedings of the First European Workshop on Probabilistic Graphical Models*, (pp. 222-230).

77. Endres. (1996). A synopsis of software engineering history: The industrial perspective. In *Report of seminar history of software engineering* (Vol. Dagstuhl Seminar, pp. 20-24). Dagstuhl Seminar.

78. Eriksson, H., & Penker, M. (1998). *Business Modeling With UML: Business Patterns at Work.* New York, NY, USA: John Wiley \& Sons, Inc.

79. Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., & Musen, M. A. (1995). Task modeling with reusable problem-solving methods. *Artificial Intelligence , 79* (2), 293-326.

80. Euzenat, J., & Shvaiko, P. (2007). *Ontology Matching.* Springer.

81. Fensel, D. (1995). *Knowledge Acquisition and Representation Language Karl.* Kluwer Academic Publishers.

82. Fernández, S., Velasco, J., & López, M. (2010). Sistema basado en reglas difusas para el mapeo de ontologías. En ESTYLF (págs. 363-368). Presented at the ESTYLF, Huelva. *ESTYLF - XV Congreso Español sobre tecnologías y lógica Fuzzy*, (pp. 363-368). Huelva.

83. Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence , 2* (3-4), 189-208.

84. Finkelstein, A., & Kramer, J. (2000). Software engineering: a roadmap. *ICSE '00: Proceedings of the Conference on The Future of Software Engineering* (pp. 3–22). New York, NY, USA: ACM.

85. Gachet, A., & Haettenschwiler, P. (2003). Developing intelligent decision support systems: A bipartite approach. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), 2774 PART 2*, pp. 87-93.

86. Gane, C., & Trish, S. (1982). *Structured Systems Analysis Tools & Tech.* MCDONNELL DOUGLAS.

87. García, J., Ortin, M. J., Moros, B., Nicolás, J., & Toval, A. (2000). De los procesos del negocio a los casos de uso. *V Jornadas de Ingeniería del Software y Bases de Datos.* Universidad de Valladolid.

88. García, J., Ortín, M. J., Moros, B., Nicolás, J., & Toval, A. (2000). Towards Use Case and Conceptual Models through Business Modeling. *19th International Conference on Conceptual Modeling* (pp. 281–294). Springer.

89. Glass, A. R. (2002). *Facts and Fallacies of Software Engineering.* Addison Wesley.

90. Goguen, J. A. (1993). Social issues in requirements engineering. *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, (pp. 194-195).

91. Gómez, A., Corcho, O., & Fernández, M. (2004). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. First Edition* (1st ed. 2004. 2nd printing ed.). Springer.

92. Gómez, A., Juristo, N., Montes, C., & Pazos, J. (1997). *Ingeniería del conocimiento.* Madrid: Centro de estudios Ramon Areces.

93. Gómez-Skarmenta, A., Vila, A., & Túnez, S. (2000). *An intelligence Decision Support Systems for the South-East Agricutural Environment.* CICYT-TIC-INFO. Valladolid: Universidad de Valladolid y Ministerio de Ciencia y Tecnología.

94. Gottesdiener, E. (2002). *Requirements by collaboration.* Addison-Wesley.

95. Guida, G., & Tasso, C. (1994). *Design and Development of Knowledge-based Systems: From Life Cycle to Methodology.* John Wiley & Sons.

96. Han, J., Kamber, M., & Pei, J. (2006). *Data Mining: Concepts and Techniques.* Morgan Kaufmann.

97. Harman, M. (2007). The Current State and Future of Search Based Software Engineering. *2007 Future of Software Engineering* (pp. 342-357). IEEE Computer Society.

98. Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology , 43* (14), 833-839.

99. Hart, P., Duda, R., & Einaudi, M. (1978). PROSPECTOR—A computer-based consultation system for mineral explorationç. *Mathematical Geology , 10* (5), 589-610.
100. Hatley, D. J., & Pirbhai, I. A. (1988). *Strategies for Real Time System Specification.* John Wiley and Sons Ltd.
101. Hayes-Roth, F. (1983). *Building Expert Systems.* Addison-Wesley Pub (Sd).
102. Heckerman, D. (1999). A tutorial on learning with Bayesian networks. En Learning in Graphical Models M. I. Jordan . Cambridge, MA: Ed. MIT Press.
103. Heineman, G. T., & Councill, W. T. (2001). *Component-Based Software Engineering: Putting the Pieces Together.* Addison-Wesley Professional.
104. Henrion, M. (1987). Some Practical Issues in Constructing Belief Networks. In L. Kanal, T. Levitt, & J. Lemmer (Ed.), *Proceedings of the Third Annual Conference on Uncertainty in Artificial Intelligence* (pp. 161-174). Amsterdam: Elsevier Science Publishers.
105. Henry, E., & Faller, B. (1995). Large-scale industrial reuse to reduce cost and cycle time. *Software, IEEE , 12* (5), 47-53.
106. Hood, C., Wiedemann, S., Fichtinger, S., & Pautz, U. (2008). *Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes.* Berlin: Springer.
107. Hoog, R. D., Benus, B., Metselaar, C., Vogler, M., & Menezes, W. (1994). *Organization model: Model definition document.* University of Amsterdam, Cap Programmator. University of Amsterdam, Cap Programmator.
108. Hoog, R. D., Benus, B., Vogler, M., & Metselaar, C. (1996). The commonKADS organization model: Content, usage and computer support. *Expert Systems with Applications , 11* (1), 29-40.
109. Hoog, R. D., Martil, R., Wielinga, B., Taylor, R., Bright, C., & Velde, W. V. (1994). *The CommonKADS model set.* University of Amsterdam, Lloyd's Register, Touche Ross Management Consultants and Free University of Brussels.
110. Iglesias, C. (1998). *Definición de una metodología.* Madrid: Phd. Tesis.
111. Institute of Electrical and Electronics Engineers Staff . (1990). *IEEE Computer Dictionary - Compilation of IEEE Standard Computer Glossaries, 610-1990* . The Institute of Electrical and Electronics Engineers. New York, USA: The Institute of Electrical and Electronics Engineers.
112. Institute of Electrical and Electronics Engineers Staff. (1998). *IEEE guide for developing system requirements specifications. Std 830.* IEEE Computer Society.
113. Institute of Electrical and Electronics Engineers Staff. (1999). *IEEE recommended practice for internet practices - Web page engineering - intranet/extranet applications.*

IEEE Computer Society, Internet Best Practices Working Group. New York, USA: The Institute of Electrical and Electronics Engineers.

114. Institute of Electrical and Electronics Engineers Staff. (2006). *IEEE Standard for Developing a.* IEEE Computer Society, New York.
115. Institute of Electrical and Electronics Engineers Staff. (1998). Industry implementation of International Standard ISO/IEC 12207: 1995. (ISO/IEC 12207 standard for information technology - software life cycle processes - implementation considerations. IEEE Corporation.
116. Institute of Electrical and Electronics Engineers Staff. (2000). *Recommended Practice for Architectural Description of Software-intensive System, IEEE Std 1471–2000.* IEEE Computer Society.
117. International Council on System Engineering. (n.d.). *International Council on System Engineering.* Retrieved 2010 йил 01-03 from INCOSE: http://www.incose.org/
118. ISO - International Organization for Standardization. (1994). *ISO 9001:1994 Quality Systems - Model for Quality Assurance in Design, Development, Production, Installation and Servicing.* ISO - International Organization for Standardization.
119. Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles , 37*, 547–579.
120. Jackson, M. (1975). *Principles Of Program Design* (1 ed.). Academic Press.
121. Jackson, M. (1983). *System Development.* Prentice-Hall.
122. Jacobson, I. (2008). *Object-Oriented Software Engineering: A Use Case Driven Approach* (2 Sub ed.). Addison-Wesley Pub (Sd).
123. Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison-Wesley Professional.
124. Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992). *Object Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley Professional.
125. Jensen, F., & Nielsen, T. (2007). *Bayesian networks and decision graphs.* Springer-Verlag New York.
126. Julián, V., & Botti, V. (2003). Desarrollo de sistemas multi-agente en tiempo real. *Inteligencia Artificial, Revista Iberoamericana de IA , 7* (18), 65-80.
127. Juristo, N., & Acuña, S. T. (2002). Software Engineering and Knowledge Engineering. *Expert Systems with Applications , 23* (4), 345-347.
128. Kingston, J. K. (1998). Designing knowledge based systems: the CommonKADS design model. *Knowledge-Based Systems , 11* (5-6), 311-319.

129. Kjaerulff, U. B., & Madsen, A. (2008). *Bayesian networks and influence diagrams: a guide to construction and analysis.* Springer-Verlag New York, Inc.

130. Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture(TM): Practice and Promise.* Addison-Wesley Professional.

131. Korb, K., & Nicholson, A. (2003). *Bayesian Artificial Intelligence.* Chapman & Hall.

132. Kotonya, G., & Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques.* Wiley.

133. Kruchten, P. (1995). The "4+1" View Model of Software Architecture. *IEEE Software , 12* (6), 42-50.

134. Lang, M., & Duggan, J. (2001). A Tool to Support Collaborative Software Requirements Management. *Requirements Engineering , 6* (3), 161-172.

135. Loucopoulos, A. P., & Karakostas, V. (1995). *System Requirements Engineering.* New York, NY, USA: McGraw-Hill, Inc.

136. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

137. McDermot, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus, *Automating Knowledge Acquisition for Expert Systems* (pp. 225-256). Boston: Kluwer Academic Publishers.

138. Mellor, S. J., & Shlaer, S. (1991). *Object Life Cycles: Modeling the World In States* (Facsimile ed.). Prentice Hall PTR.

139. Meziane, F., & Vadera, S. (Edits.). (2009). *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects.* IGI publishing.

140. Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill Science/Engineering/Math.

141. Murugesan, S., & Deshpande, Y. (2001). *Web Engineering : Managing Diversity and Complexity of Web Application Development* (1 ed.). Springer.

142. National Aeronautics and Space Administration. (1995). *NASA Systems Engineering Handbook SP-610S.* NASA.

143. Naur, P., & Randell, B. (1969). *Software engineering.* Garminch, Germany: NATO SCIENCE COMMITEE.

144. Neapolitan, R. E. (2003). *Learning Bayesian Networks (1º ed).* Prentice Hall.

145. Newell, A. (1982). The knowledge level. *Artificial Intelligence , 18* (1), 87-127.

146. Newell, A., & Simon, H. (1963). GPS, a program that simulates human thought. *Computers and Thought* , 279–293.

147. Object Management Group (OMG). (2003). *MDA Guide version 1.0.1.* OMG.

148. Olivé Ramon, A. (2002). *Modelització conceptual de sistemes d'informació l'estructura.* Barcelona: Barcelona Edicions UPC.

149. Orellana, F. J., Cañadas, J., Águila, I. M., & Túnez, S. (2008). INSCO Requisite - A Web-Based RM-Tool to support Hybrid Software Development. *Proceedings of the Tenth International Conference on Enterprise Information Systems ICEIS (3-1),* (pp. 326-329). Barcelona.

150. Orellana, F. J., Guil, F., Águila, I. M., & Túnez, S. (2005). A WEB-CASE Tool Prototype for Hybrid Software Development. En *Computer Aided Systems Theory – EUROCAST 2005* (Vol. 3643, págs. 217-222). La Laguna.

151. Orr, K. T. (1977). *Structured Systems Development.* Yourdon.

152. Osborn, A. F. (1957). *Applied imagination; principles and procedures of creative thinking.* Scribner.

153. Palma, J. T., & Marín, R. (2008). *Inteligencia artificial: Métodos, técnicas y aplicaciones.* McGraw-Hill.

154. Palma, J. T., Paniagua, E., Martín, F., & Marín, R. (2000). Ingeniería del Conocimiento. De la Extracción al Modelado de Conocimimiento. *Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial , 4* (11), 46-72.

155. Paulk, M., Curtis, B., Chrissis, M., & Weber, C. (1993). *Capability Maturity Model for Software (Version 1.1).* SEI. Carnigie Melon: SEI.

156. Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference.* San Mateo, Ca: Morgan Kaufman.

157. Peña, D. (2002). *Análisis multivariante de datos.* Mc-GrawHill.

158. Piattini, M. (2003). *Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software.* RA-MA.

159. Piattini, M., Calvo, J., Joaquín, C., & Férnandez, L. (1996). *Análisis y diseño de Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software.* RA-MA.

160. Plant, R. T., & Gamble, R. (1997). Using meta-knowledge within a multilevel framework for KBS development. *International Journal of Human-Computer Studies , 46* (4), 523-547.

161. Post, W., Wielinga, B., Hoog, R. D., & Schreiber, G. (1997). Organizational modeling in CommonKADS: the emergency medical service. *IEEE Expert [see also IEEE Intelligent Systems and Their Applications] , 12* (6), 46-52.

162. Pressman, R. S. (1993). *Ingeniería del Software - Un Enfoque Práctico* (3º ed.). McGraw-Hill Companies.

163. Pressman, R. S. (2002). *Ingeniería del Software - Un Enfoque Práctico* (5º ed.). McGraw-Hill Companies.

164. Randell, B., & Naur, P. (1970). *Software engineering techniques.* Rome, Italy: NATO SCIENCE COMMITEE.

165. Rashad, U., Arulledran, P., Hawthorne, M., & Kendal, S. (2001). A Hybrid Medical Information System for the Diagnosis of Dizziness. *Proceedings 4th International Conference Neural Networks and Expert Systems in Medicine and Healthcare.* Greece.

166. Rolland, C., & Prakash, N. (2000). From conceptual modelling to requirements engineering. *Annals of Software Engineering , 10* (1-4), 151-176.

167. Rolland, C., Souveyet, C., & Achour, C. B. (1998). Guiding goal modeling using scenarios. *Software Engineering, IEEE Transactions on , 24* (12), 1055-1071.

168. Rosenberg, D., & Scott, K. (1999). *Use Case Driven Object Modeling with UML : A Practical Approach.* Addison-Wesley Professional.

169. Ross, D. T., & Schoman, K. E. (1977). Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering , SE-3* (1), 6-15.

170. Rumbaugh, J. R., Blaha, M. R., Lorensen, W., Eddy, F., & Premerlani, W. (1990). *Object-Oriented Modeling and Design* (United States Ed ed.). Prentice Hall.

171. Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *The Unified Modeling Language Reference Manual (2nd Edition)* (2 ed.). Addison-Wesley Professional.

172. Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Addison-Wesley Professional.

173. Saeki, M., & Chang, S. K. (2002). Software specification and design methods. In *Handbook on Software Engineering and Knowledge Engineering.* (Vol. 1, pp. 263-306). Pittsburg: World Scientific Publishing Company.

174. Sage, A. P., & Armstrong, J. E. (2000). *Introduction to Systems Engineering.* Wiley-Interscience.

175. Sagrado, J. d., & Águila, I. M. (2009). A Bayesian Network for Predicting the Need for a Requirements Review. In F. Meziane, & S. Vadera, *Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects* (pp. 106-128). IGI Global.

176. Sagrado, J. d., & Águila, I. M. (2009). Ant Colony Optimization for Requirement selection in Incremental Software development. *Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE '09).* IEEE Computer Society.

177. Sagrado, J., & Águila, I. M. (2007). Olive Fly Infestation Prediction Using Machine Learning Techniques. En *Current Topics in Artificial Intelligence* (Vol. 4788, págs. 229-238). Springer.

178. Schneider, G., & Winters, J. P. (1998). *Applying Use Cases: A Practical Guide.* Addison-Wesley Professional.

179. Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R. D., Shadbolt, N., Velde, W. V., et al. (1999). *Knowledge Engineering and Management: The CommonKADS Methodology.* Massachusetts: MIT Press.

180. Schreiber, G., Crubézy, M., & Musen, M. (2000). A Case Study in Using Protégé-2000 as a Tool for CommonKADS. In M. a. Knowledge Engineering and Knowledge Management Methods.

181. Schreiber, G., Wielinga, B., Akkermans, H., Velde, W. V., & Anjewierden, A. (1994). CML: The commonKADS conceptual modelling language. *A Future for Knowledge Acquisition* .

182. Schreiber, G., Wielinga, B., Hoog, R. d., Akkermans, H., & Velde, W. V. (1994). CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert [see also IEEE Intelligent Systems and Their Applications] , 9* (6), 28-37.

183. Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentice Hall.

184. Shadbolt, N., Motta, E., & Rouge, A. (1993). Constructing knowledge-based systems. *Software, IEEE , 10* (6), 34-38.

185. Shortliffe, E. H. (1976). *Computer-based Medical Consultations.* Elsevier.

186. Shu-Hsien, L. (2005). Expert system methodologies and applications--a decade review from 1995 to 2004. *Expert Systems with Applications , 28* (1), 93-103.

187. Shumate, K., & Keller, M. (1992). *Software Specification and Design: Disciplined Approach for Real-time Systems.* John Wiley & Sons.

188. Singhal, A. (2001). Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bulletin , 24* (4).

189. Sinha, V., Sengupta, B., & Chandra, S. (2006). Enabling Collaboration in Distributed Requirements Management. *IEEE Software , 23* (5), 52-61.

190. Sommerville, I. (2000). *Software Engineering (6th Edition)* (6 ed.). Addison Wesley.

191. Sommerville, I. (2004). *Software Engineering (7th Edition)* (7 ed.). Addison Wesley.

192. Sommerville, I. (2006). *Software Engineering: (Update) (8th Edition)* (8 ed.). Addison Wesley.

193. Sommerville, I., & Sawyer, P. (1997). *Requirements Engineering: A Good Practice Guide* (1 ed.). Wiley.

194. Soni, D., Nord, R. L., & Hofmeister, C. (1995). Software architecture in industrial applications. *17th international conference on Software engineering. ICSE* (págs. 196-207). Seattle, Washington, United States: ACM.

195. Sørensen, T. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biologiske Skrifter / Kongelige Danske Videnskabernes Selskab , 5* (4), 1-34.

196. Speel, Schreiber, G., Joolingen, W. V., Heijst, G. V., & Beijer, G. (2001). Conceptual Models for Knowledge-Based System. In *Encyclopedia of Computer Science and Technology.* New York: Marcel Dekker Inc.

197. Spirtes, P., Glamour, C., & Scheines, R. (1991). An algorithm for fast recovery of sparse causal graphs. *Social Science Computing Reviews, , 9* (1), 62-72.

198. Storer, R., & Jackson, M. A. (1987). *Practical Program Development Using Jsp: A Manual of Program Design Using the Design Method Developed by M.A. Jackson.* Alfred Waller Ltd.

199. Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering , 25* (1-2), 161-197.

200. Studer, R., Fensel, D., Decker, S., & Benjamins, V. (1999). Knowledge Engineering: Survey and Future Directions. In S. B. Heidelberg, & S. B. Heidelberg (Ed.), *XPS-99: Knowledge-Based Systems* (Vol. LNCS 1570/1999, pp. 1-23). Berlin: Springer Berlin / Heidelberg.

201. Sutcliffe, A. (1988). *Jackson System Development.* Prentice Hall.

202. Sutcliffe, A. (2002). *User-centred requirements engineering.* Springer.

203. Taboada, M., Des, J., Mira, J., & Marín, R. (2001). Diagnosis systems in medicine with reusable knowledge components. *Intelligent Systems, IEEE , 16* (6), 68-73.

204. Thronesbery, C., Molin, A., & Schreckenghost, D. (2007). A Storyboard Tool to Assist Concept of Operations Development. . *Aerospace Conference, 2007 IEEE.,* (pp. 1-8).

205. Túnez, S., Águila, I. M., & Marín, R. (2001). An expertise model dor therapy planning using abductive reasoning. *Cybernetics and Systems: An International Journal , 32* (8), 829 - 849.

206. Túnez, S., Marín, R., Águila, I. M., Bosch, A., & Torres, M. (1998). An abductive method for solving a treatment problem. *Proceedings. 24th Euromicro Conference 1998, 2,* pp. 737-744.

207. Valente, A., Breuker, J., & Velde, W. v. (1998). TheCommonKADS library in perspective. *International Journal of Human-Computer Studies , 49* (4), 391-416.

208. Velde, W. V., Duursma, C., Schreiber, G., Terpstra, P., Schrooten, R., Golfinopoulos, V., et al. (1994). *Design model and process.* Brussels: Free University Brussels, University of Amsterdam, Swedish Institute of Computer Science and Cap Programmator.

209. Waern, A., & Gala, S. (1993). *The CommonKADS agent model.* Swedish Institute of Computer Science and EIRTEL.

210. Waern, A., Hook, K., Gustavsson, R., & Holm, P. (1993). *The CommonKADS communication model.* Swedish Institute of Computer Science and EIRTEL.

211. Wang, Y., & Patel, D. (2000). Editors' introduction: Comparative software engineering: Review and perspectives. *Annals of Software Engineering , 10* (1), 1-10.

212. Ward, P. T., & Mellor, S. J. (1985). *Structured Development for Real-Time Systems : Essential Modeling Techniques* (Facsimile ed.). Prentice Hall.

213. Whitten, J. (1996). *Analisis y Diseno de Sistemas de Informacion.* Harcourt.

214. Wielinga, B., Akkermans, H., Hassan, H., Olsson, O., Orsvärn, K., Schreiber, G., et al. (1994). *Expertise model definition document.* University of Amsterdam, Free University of Brussels and Netherland Energy Research Centre ECN.

215. Wood, J., & Silver, D. (1995). *Joint Application Development* (2 ed.). Wiley.

216. Wood, M. F., & DeLoach, S. A. (2001). An Overview of the Multiagent Systems Engineering Methodology. In *Agent-Oriented Software Engineering* (Vol. LNCS 1957/2001). Springer Berlin / Heidelberg.

217. Yourdon, E. (1997). *Análisis Estructurado Moderno.* Prentice Hall.

218. Yourdon, E. (1988). *Modern Structured Analysis* (United States Ed ed.). United States: Prentice Hall PTR.

219. Yourdon, E., & Constantine, L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (Facsimile ed.). Prentice Hall.

Apéndices

A.1 Descripción del proyecto Arrendamientos

Debido a la falta de viviendas en alquiler, desde los gobiernos locales se ha promovido la iniciativa de disponer de un servicio de control y asignación de arrendamientos.

La asignación de viviendas se hace desde el ayuntamiento. Las personas que desean alquilar una vivienda se registran como potenciales "arrendatarios". Cada mes se revisa la lista de viviendas y de potenciales arrendatarios para estudiar su asignación. A partir de esta asignación se publica un resumen de los datos de ese proceso de asignación para que los potenciales arrendatarios si lo desean cambien sus prioridades, por ejemplo, seleccionando un área menos popular. Por una parte se hace necesario disponer de una base de datos con la información relativa a los arrendatarios y a las viviendas, así como quien es el arrendador de cada una de ellas.

Ingresos \ Edad	18-27	28-64	+65	Renta mensual
	0-7.999	0-6.999	0-4.999	250
	8.000-11.999	7.000-10.999	5.000-7.999	400
	12.000-15.999	11.000-15-999	8.000-11.999	500
	16.000-19.999	16.000-19.999	12.000-19.999	600
	20.000-24.999			700
	25.000-29.999			800
	+30.000			900

Ligado a cada contrato de alquiler se negocia un seguro que permitirá caso de falta de pago o caso de desperfectos en la vivienda cubrir los gastos ocasionados al finalizar el arrendamiento. Para elegir una vivienda, los potenciales arrendatarios deberán satisfacer una serie de criterios. Existen cuatro tipos de criterios, primero se selecciona la categoría de la vivienda (zona), después el número de habitaciones correspondiente con las necesidades del arrendatario y la vivienda. El tercero sería la correspondencia entre los ingresos del potencial arrendatario y la renta pactada para la vivienda. Finalmente se pueden especificar condiciones particulares para cada vivienda como no fumadores o sin mascotas, etc.

A.2 Modelo de negocio de Arrendamientos

NEG_Proyecto	
Descripción del proyecto	Gestión del sistema de asignación y seguimiento de viviendas arrendadas dependiente de la administración local.
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local: Servicios sociales, Área de apoyo a la juventud Ley de Arrendamientos Urbanos (LAU, 1994) Compañías aseguradoras
Problemas y oportunidades	La asignación y valoración consume tiempo Imposibilidad de tramitar casos urgentes Falta de conocimiento del estado real del parque de viviendas en alquiler Gestión administrativa del arrendamiento Gestión administrativa de los seguros asociados Peritaciones de las viviendas
Contexto organizacional	Misión, objetivos generales de la organización: Facilitar el acceso a viviendas sobre todo a jóvenes siendo ellos mismo los que analicen sus posibilidades Abrir el mercado de viviendas en alquiler dando garantías a los arrendadores
	Factores externos importantes con los que se relaciona la organización: - LAU - Mercado inmobiliario externo (Compra y alquiler) - Variaciones de precios
	Estrategia de la organización: Servicio público (arrendatarios y arrendadores)
	Escala de valores por la que se rige: Bienestar socio-económico del municipal

Soluciones	Base de datos de viviendas Análisis del mercado para ajustar las tablas SBC par la asignación de viviendas Software para el control del estado de los arrendamientos
Estructura	Figura 1
Proceso	Figura 2
Personas	Arrendatarios Personal evaluador Peritos de las compañías Administrativos del ayuntamiento
Conceptos	Figura 3
Recursos	Se dispone de 2 personas para la gestión del trabajo. Un presupuesto inicial de 20.000 Euros Despacho en la segunda planta de Ayuntamiento completamente dotado.
Conocimiento	Criterios de asignación

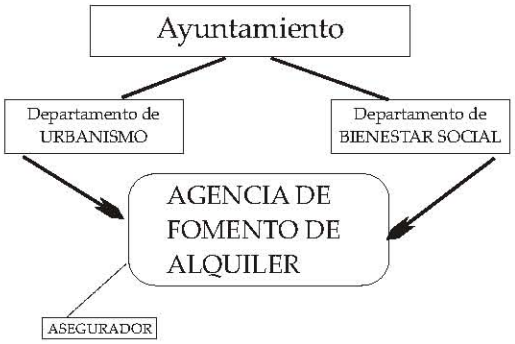


Figura A.1 Estructura.

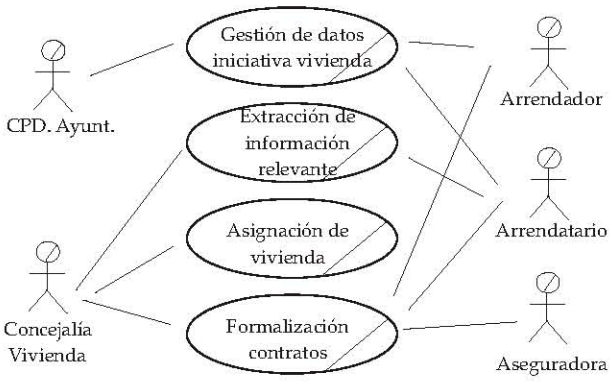


Figura A.2 Procesos de negocio.

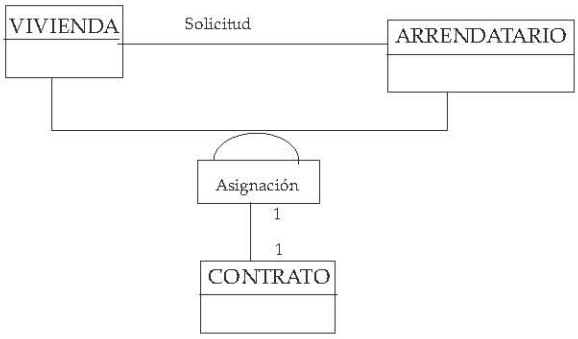


Figura A.3 Dominio.

NEG_Objetivo_1	**Asignación viviendas**
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local
Descripción	El sistema debe asegurar la correcta asignación de las viviendas a los potenciales arrendatarios de forma que se verifiquen los criterios fijados por la administración local
Subobjetivos	No relevante
Realizado por	Personal administrativo del ayuntamiento
¿Intensivo en conocimiento?	Si
Importancia	Vital
Urgencia	Alta
Comentarios	

NEG_Objetivo-2	Extracción de información relevante
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local
Descripción	Esta tarea revisa el proceso de asignación que se ha realizado ese mes, generando informes tipo que la concejalía de vivienda empleará para ajustar las posibles subvenciones de vivienda u otras medidas que mejoren el servicio prestado por el proyecto iniciativa vivienda
Subobjetivos	No relevante
Realizado por	Concejalía de vivienda
¿Intensivo en conocimiento?	No actualmente
Importancia	Quedaría bien
Urgencia	Baja
Comentarios	En el futuro podría plantearse la automatización de esta tarea en ciertos parámetros.

NEG_Objetivo_3	Gestión de datos iniciativa vivienda
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local
Descripción	Esta tarea debe realizar el mantenimiento de la base de datos del proyecto iniciativa vivienda, tanto de vivienda como de peticiones y de las personas ligadas a cada uno de estos conceptos
Subobjetivos	No relevante
Realizado por	Personal del centro de procesamiento de datos del ayuntamiento
¿Intensivo en conocimiento?	No
Importancia	Vital
Urgencia	Alta
Comentarios	Deben mantenerse los estándares en gestión de datos que se han definido en toda las gestión de la administración local.

NEG_Objetivo_4	Formalización de contratos
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Administración local
Descripción	Esta tarea gestiona los contraltos realizados, anotando los arrendamientos llevados a efecto y las fechas de los contraltos, servirán como referencia para las pólizas de seguros
Subobjetivos	No relevante
Realizado por	Personal administrativo del ayuntamiento
¿Intensivo en conocimiento?	No
Importancia	Importante
Urgencia	Media
Comentarios	

NEG_Dominio_1	Solicitud
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Asignación vivienda
Responsabilidad del agente	Concejalía de vivienda
Descripción	Petición elaborada por los posibles arrendatarios
¿Formato correcto?	Si
¿Lugar correcto?	
¿Momento correcto?	Si, Cada mes, pero podrían acumularse
¿Calidad correcta?	
Comentarios	

NEG_Dominio_2	Contrato
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano – Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Formalización de contratos
Responsabilidad del agente	Concejalía de vivienda
Descripción	Documento contractual por el que se rige el arrendamiento conforme a la ley de arrendamientos urbanos vigente
¿Formato correcto?	Si
¿Lugar correcto?	Si
¿Momento correcto?	Si
¿Calidad correcta?	Si
Comentarios	Es necesario guardar copia en papel rubricada por los tomadores del contrato.

NEG_Dominio_3	Arrendatario
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano – Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Gestión de datos de iniciativa vivienda, Asignación de vivienda
Responsabilidad del agente	Concejalía de vivienda
Descripción	Persona o unidad familiar que solicita al ayuntamiento una vivienda en alquiler
¿Formato correcto?	Si
¿Lugar correcto?	Si
¿Momento correcto?	Si
¿Calidad correcta?	Si
Comentarios	

NEG_Dominio_4	Vivienda
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano – Universidad de Almería
Fuentes	Responsable programa de vivienda
Objetivos asociados	Gestión de datos de iniciativa vivienda, Asignación de vivienda
Responsabilidad del agente	Concejalía de vivienda
Descripción	Inmueble destinado al uso como vivienda cuyo propietarios o arrendador decide poner en el mercado de alquiler
¿Formato correcto?	Si
¿Lugar correcto?	Si
¿Momento correcto?	Si
¿Calidad correcta?	Si
Comentarios	.

NEG_Actor_1	Aseguradora
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Formalización de contratos
Comunicado con	Concejalía de vivienda
Conceptos relacionados	Contrato, Póliza de seguros
Descripción	Este actor representa el rol de los miembros de la compañía de seguros con la que se contratan las pólizas para cada contrato de arrendamiento formalizado
Cargo	Agente externo a la organización Sin relaciones de subordinación con los demás actores
Otras competencias	Gestión de los partes de seguro
Comentarios	

NEG_Actor_2	Arrendador
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Gestión de datos de iniciativa vivienda, Formalización de contratos
Comunicado con	Concejalía de vivienda
Conceptos relacionados	Vivienda
Descripción	Este actor representa el rol de los propietarios de las viviendas que deciden incluir sus inmuebles en el mercado de alquiler
Cargo	Es el administrador del inmueble
Otras competencias	
Comentarios	A nivel fiscal podríamos tener otro titular del arrendamiento.

NEG_Actor_3	Arrendatario
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Gestión de datos de iniciativa vivienda, Extracción de información relevante, Formalización de contratos
Comunicado con	Concejalía de vivienda
Conceptos relacionados	Arrendatario, Solicitud
Descripción	Representa a la unidad familiar que busca una nueva vivienda en alquiler
Cargo	Actúa a título propio
Otras competencias	
Comentarios	

NEG_Actor_4	CPD Ayuntamiento
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Gestión de datos iniciativa vivienda
Comunicado con	
Conceptos relacionados	Arrendatario, contrato, vivienda
Descripción	Departamento del ayuntamiento responsable de mantener la base de datos necesaria y garantizan el cumplimiento de la normativa en cuanto a protección de datos.
Cargo	
Otras competencias	
Comentarios	

NEG_Actor_5	Concejalía de vivienda del ayuntamiento
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Responsable programa vivienda
Implicado en	Asignación de vivienda, formalización de contraltos, extracción de información relevante
Comunicado con	
Conceptos relacionados	Vivienda, arrendatario, contrato, solicitud
Descripción	Departamento básico que gestionará el sistema
Cargo	
Otras competencias	
Comentarios	

A.3 Modelo de servicios de Arrendamientos

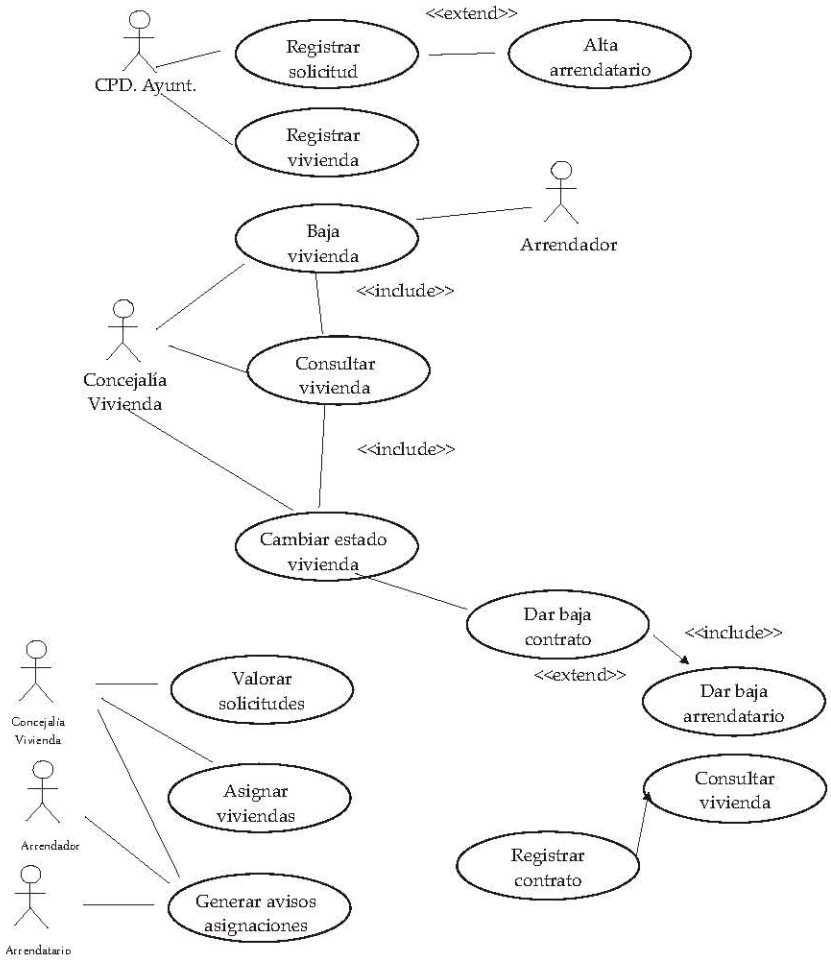


Figura A.4 Servicios.

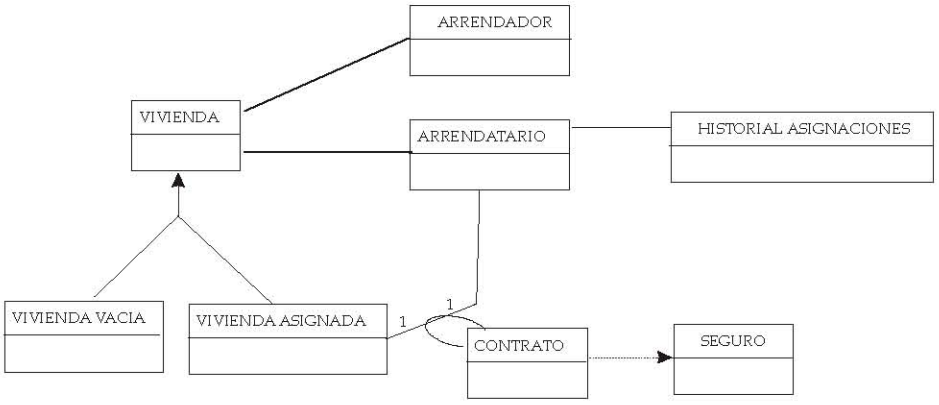


Figura A.5 Información.

RF-1	Registrar solicitud	
Versión	Versión 1, 05-Marzo-2010	
Autores	Isabel María del Águila Cano - Universidad de Almería	
Fuentes	Centro de procesamiento de datos del Ayuntamiento	
Objetivos asociados	Gestión de datos de iniciativa vivienda	
Requisitos asociados	RI-1 Solicitud RI-2 Arrendatario	
Descripción	El sistema deberá comportarse tal como se describe cuando llegue una nueva solicitud al ayuntamiento	
Precondición	El plazo de solicitudes debe estar abierto	
	1	El arrendatario inicial el registro de la solicitud
	2	El sistema pide que el NIF del arrendatario
	3	Si no existe una solicitud anterior con ese DNI dar alta de arrendatario, ejecutando el caso de uso Alta Arrendatario
	4	Si existe obtener datos de la solicitud anterior
	5	El sistema pide que se introduzcan los datos de la solicitud, o se modifiquen
Postcondición	El sistema ha almacenado la información de la solicitud	
Excepciones		Si es la segunda solicitud de este mes comprobar la fecha de la última válida
Rendimiento		
¿Intensivo en conocimiento?	No	
Frecuencia	Una vez al mes	
Importancia	Alta	
Urgencia	Vital	
Estado	Validada	
Estabilidad	Estable	
Referencia		
Comentario	Posibilidad de extender el registro vía Web	

SER_no_funcional_1	Confidencialidad
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Centro de procesamiento de datos del Ayuntamiento
Objetivos asociados	Gestión datos iniciativa vivienda
Requisitos asociados	Alta vivienda, consulta vivienda
Descripción	Confidencialidad en la dirección exacta de la vivienda
Importancia	Media
Urgencia	Baja
Estado	
Estabilidad	Descrito
Referencia	
Comentarios	El objetivo es evitar ocupaciones o asaltos

Rinfo-1	Vivienda
Versión	Versión 1, 05-Marzo-2010
Autores	Isabel María del Águila Cano - Universidad de Almería
Fuentes	Centro de procesamiento de datos del Ayuntamiento
Objetivos asociados	Gestión de datos iniciativa vivienda
Requisitos asociados	Alta vivienda
Descripción	El sistema deberá almacenar la información correspondiente a una vivienda a alquilar. En concreto:
Datos específicos	Identificador: entero Dirección: cadena Renta: entero Ciudad: cadena N° habitantes: entero N° habitaciones:entero Referencia catastral: cadena Agua: logico Luz: logico Telefono: logico ADSL: logico Electrodomésticos: logico Gastos Comunes: entero Otros: cadena Fumadores: logico Mascotas: lógico
Intervalo temporal	Persistente
Importancia	Alta
Urgencia	Vital
Estado	Validado
Estabilidad	Alta
Referencia	-
Comentarios	Posibilidad de incluir plano

A.4 Modelo de conocimiento de Arrendamientos

```
KNOWLEDGE-MODEL Arrendamientos;

/* Modelo de conocimiento para la tarea de asignación de peticiones en el dominio
de la asignación de vivienda*/

DOMAIN-KNOWLEDGE vivienda-dominio;

DOMAIN-KNOWLEDGE-SCHEMA esquema-valoracion;

/* tipos de información vivienda/arrendatario */

CONCEPT vivienda-registrada;
END CONCEPT vivienda-registrada;

CONCEPT vivienda-nueva;
END CONCEPT vivienda-nueva;
```

```
CONCEPT vivienda- disponible;
    DESCRIPTION:
        "vivienda que puede ser asignada";
    VIEWPOINTS:
        DIMENSION: categoria;
            vivienda-nueva, vivienda-registrada;
            DISJOINT: YES;
            COMPLETE: NO;
    HAS-PARTS:
        vivienda;
            ROLE: descripcion;
            CARDINALITY: 1-1;
    PROPERTIES:
        vivienda-num: NATURAL;
    AXIOMS:
           categoria = vivienda-nueva OR    categoria = vivienda-registrada
        <->        descripcion.tipo-subvencion = subvencionable;
END CONCEPT disponible-vivienda;

CONCEPT vivienda;
    DESCRIPTION:
        "Descripción del vivienda disponible en la base de datos";
    SUPER-TYPE-OF:
        casa, apartamento;
            DISJOINT: YES;
            COMPLETE: YES;
    PROPERTIES:
        dirección: STRING;
        ciudad: STRING;
        num-habitaciones: NATURAL;
        renta: REAL;
        min-num-habitantes: NATURAL;
        max-num-habitantes: NATURAL;
        tipo-subvencion: valor-tipo-subvencion;
    AXIOMS:
            min-num-habitantes <= max-num-habitantes;
END CONCEPT vivienda;

VALUE-TYPE valor-tipo-subvención;
    TYPE: NOMINAL;
    VALUE-LIST: {subvencionada, libre};
END VALUE-TYPE valor-tipo-subvención;

CONCEPT casa;
    SUB-TYPE-OF: vivienda;
    PROPERTIES:
        superficie-habitable: NATURAL;
END CONCEPT casa;

CONCEPT apartamento;
    SUB-TYPE-OF: vivienda;
    PROPERTIES:
        planta: NATURAL;
        camas-disponibles: BOOLEAN;
```

```
END CONCEPT apartamento;

CONCEPT potencial-arrendatario;
    DESCRIPTION:
        "Una persona o grupo de personas registradas como posibles peticionarios de
un vivienda";
    SUPER-TYPE-OF: nuevo, residente-registrado;
        DISJOINT: YES;
        COMPLETE: YES;
    PROPERTIES:
        nombre: STRING;
        dirección-actual: STRING;
        ciudad: STRING;
        fecha-nacimiento: STRING;
        categoria-por-edad: valor-categoria-por-edad;
        año-registro: NATURAL;
        num-miembros: NATURAL;
        miembros: valor-miembros;
END CONCEPT potencial-arrendatario;

CONCEPT nuevo;
    DESCRIPTION:
        "Persona que no tiene asignada una casa en el sistema ni está registrada";
    SUB-TYPE-OF:
        potencial-arrendatario;
    PROPERTIES:
        fecha-registro: STRING;
END CONCEPT nuevo;

CONCEPT residente-registrado;
    DESCRIPTION:
        "Alguien que ya tiene una casa asignada en el sistema";
    SUB-TYPE-OF:
        potencial-arrendatario;
    PROPERTIES:
        fecha-ultima-asignacion: STRING;
END CONCEPT residente-registrado;

VALUE-TYPE valor-categoria-por-edad;
     TYPE: ORDINAL;
     VALUE-LIST: {'18-27', '28-64', '+65'};
END VALUE-TYPE valor-categoria-por-edad;

VALUE-TYPE valor-miembros;
     TYPE: NOMINAL;
     VALUE-LIST: {individual, varias-personas};
END VALUE-TYPE valor-miembros;

BINARY-RELATION vivienda-solicitante;
    DESCRIPTION:
        "Solicitud de un arrendatario a una cierta vivienda";
    ARGUMENT-1: potencial-arrendatario;
        ROLE: arrendatario;
        CARDINALITY: 0+;
    ARGUMENT-2: vivienda;
```

```
        CARDINALITY: 0-2;
    PROPERTIES:
        fecha-solicitud: DATE;
END BINARY-RELATION vivienda-solicitante;

/* Tipos de conocimiento de valoracion  */

RULE-SCHEMA abstraccion-vivienda;
    ANTECEDENT:
        vivienda-solicitante;
            CARDINALITY: 1+;
    CONSEQUENT:
        vivienda-solicitante;
            CARDINALITY: 1;
    CONNECTION-SYMBOL:
        se-abstrae-por;
END RULE-SCHEMA abstraccion-vivienda;

CONCEPT vivienda-criterio;
    PROPERTIES:
        valor-de-verdad: BOOLEAN;
END CONCEPT vivienda-criterio;

CONCEPT correct-tañamo-familia;
    SUB-TYPE-OF: vivienda-criterio;
END CONCEPT correct-tamaño-familia;

CONCEPT correct-tipo-vivienda;
    SUB-TYPE-OF: vivienda-criterio;
END CONCEPT correct-tipo-vivienda;

CONCEPT restriccion-concreta-vivienda;
    SUB-TYPE-OF: vivienda-criterio;
END CONCEPT restriccion-concreta-vivienda;

CONCEPT renta-acorde-ingresos;
    SUB-TYPE-OF: vivienda-criterio;
END CONCEPT renta-acorde-ingresos;

RULE-SCHEMA requisito-vivienda;
    ANTECEDENT:
        vivienda-solicitante;
            CARDINALITY: 1+;
    CONSEQUENT:
        vivienda-criterio;
            CARDINALITY: 1;
    CONNECTION-SYMBOL:
        indica;
END RULE-SCHEMA requisito-vivienda;

CONCEPT vivienda-decision;
    PROPERTIES:
        value: {eligible, no-eligible };
END CONCEPT vivienda-decision;
```

```
RULE-SCHEMA vivienda-regla-decision;
    ANTECEDENT:
       vivienda-criterio;
    CONSEQUENT:
       vivienda-decision;
    CONNECTION-SYMBOL:
       implica;
END RULE-SCHEMA vivienda-regla-decision;

END DOMAIN-KNOWLEDGE-SCHEMA esquema-valoracion;

DOMAIN-MODEL valoracion-conocimiento;
    USES:
       abstraccion-vivienda FROM esquema-valoracion,
       requisito-requisito-vivienda FROM esquema-valoracion,
       vivienda-regla-decision FROM esquema-valoracion;
    EXPRESSIONS:
       /* reglas de abstracción TODAY() función que devuelve la fecha*/

       TODAY() - arrendatario.fecha-nac < 28
           SE-ABSTRAE-POR
       arrendatario.categoria-por-edad = '18-27';

       TODAY() - arrendatario.fecha-nac >= 28 AND
       TODAY() - arrendatario.fecha-nac < 65
           SE-ABSTRAE-POR
       arrendatario.categoria-por-edad = '28-64';

       TODAY() - arrendatario.fecha-nac >= 65
           SE-ABSTRAE-POR
       arrendatario.categoria-por-edad = '+65';

       arrendatario.num-miembros = 1
           SE-ABSTRAE-POR
       arrendatario.miembros = individual;

       arrendatario.num-miembros > 1
           SE-ABSTRAE-POR
       arrendatario.miembros = varias-personas;

       /* Requisitos */
       /* ¿vivienda de la categoría correcta? */

       vivienda.descripcion.tip-subvención = libre
           INDICA
       correct-vivienda-categoria.valor-de-verdad = true;

       vivienda.categoria = nueva-vivienda AND
       arrendatario.sub-type = nuevo
           INDICA
       correct-vivienda-categoria.valor_de_verdad = true;

       vivienda.categoria = registrada-vivienda AND
       arrendatario.sub-type = residente-registrado
           INDICA
```

```
correct-vivienda-categoria.valor-de-verdad = true;

/* ¿tamaño casa correcto? */

vivienda.descripcion.min-num-habitantes <= arrendatario.num-miembros AND
vivienda.descripcion.max-num-habitantes >= arrendatario.num-miembros
    INDICA
correct-tamaño-familia.valor-de-verdad = true;

/* renta acorde con ingresos */

arrendatario.ingresos-anuales>= 20000 AND
arrendatario.ingresos-anuales<  24999 AND
vivienda.descripcion.renta < 700
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.ingresos-anuales>= 25000 AND
arrendatario.ingresos-anuales<  29999AND
vivienda.descripcion.renta < 800
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.ingresos-anuales> 30000 AND
vivienda.descripcion.renta < 900
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* ¿Renta acorde con los ingresos para individual 18 y 27  */

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '18-27' AND
arrendatario.ingresos-anuales<afgafdgafg  AND
vivienda.descripcion.renta < 524
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales>= 27000 AND
arrendatario.ingresos-anuales<  35000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales>= 35000 AND
arrendatario.ingresos-anuales<  45000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+18' AND
```

```
arrendatario.ingresos-anuales>= 45000 AND
arrendatario.ingresos-anuales<  70000 AND
vivienda.descripcion.renta >= 810
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* Renta acorde ingresos  varias-personas +18  */

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales< 38000 AND
vivienda.descripcion.renta < 524
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales>= 38000 AND
arrendatario.ingresos-anuales<  46000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales>= 46000 AND
arrendatario.ingresos-anuales<  56000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+18' AND
arrendatario.ingresos-anuales>= 56000 AND
arrendatario.ingresos-anuales<  70000 AND
vivienda.descripcion.renta >= 810
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* Renta acorde ingresos  individual 28-64 */

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales< 25000 AND
vivienda.descripcion.renta < 679
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 25000 AND
arrendatario.ingresos-anuales<  35000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;
```

```
arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 35000 AND
arrendatario.ingresos-anuales<  45000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 45000 AND
arrendatario.ingresos-anuales<  70000 AND
vivienda.descripcion.renta >= 810
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* renta acorde ingresos   varias-personas 28-64 */

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales< 34000 AND
vivienda.descripcion.renta < 679
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 24000 AND
arrendatario.ingresos-anuales<  46000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 46000 AND
arrendatario.ingresos-anuales<  56000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '28-64' AND
arrendatario.ingresos-anuales>= 56000 AND
arrendatario.ingresos-anuales<  70000 AND
vivienda.descripcion.renta >= 810
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* Renta acorde ingresos  individual +65 */

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales< 25000 AND
```

```
vivienda.descripcion.renta < 679
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales>= 25000 AND
arrendatario.ingresos-anuales<  29000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales>= 29000 AND
arrendatario.ingresos-anuales<  45000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = individual AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales>= 45000 AND
arrendatario.ingresos-anuales<  70000 AND
vivienda.descripcion.renta >= 810
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

/* renta acorde ingresos  varias-personas +65 */

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales< 31000 AND
vivienda.descripcion.renta < 679
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales>= 31000 AND
arrendatario.ingresos-anuales<  39000 AND
vivienda.descripcion.renta < 1007
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+65' AND
arrendatario.ingresos-anuales>= 39000 AND
arrendatario.ingresos-anuales<  56000 AND
vivienda.descripcion.renta >= 600
     INDICA
 renta-acorde-ingresos.valor-de-verdad= true;

arrendatario.miembros = varias-personas AND
arrendatario.categoria-por-edad = '+65' AND
```

```
        arrendatario.ingresos-anuales>= 56000 AND
        arrendatario.ingresos-anuales<  70000 AND
        vivienda.descripcion.renta >= 810
             INDICA
         renta-acorde-ingresos.valor-de-verdad= true;

        /* reglas de decisión */

        correcto-vivienda-categoria.valor-de-verdad= true AND
        correcto-tamaño-familia.valor-de-verdad= true AND
        renta-acorde-ingresos.valor-de-verdad= true AND
        vivienda-specific-constraints.valor-de-verdad= true
            IMPLICA
        decision.valor = eligible;

        correcto-vivienda-categoria.valor-de-verdad= false
            IMPLICA
        decision.valor = not-eligible;

        correcto-tamaño-familia.valor-de-verdad= false
            IMPLICA
        decision.valor = not-eligible;

        renta-acorde-ingresos.valor-de-verdad= false
            IMPLICA
        decision.valor = not-eligible;

        vivienda-specific-constraints.valor-de-verdad= false
            IMPLICA
        decision.valor = no-eligible;
END DOMAIN-MODEL valoracion-conocimiento;

END DOMAIN-KNOWLEDGE

INFERENCE-KNOWLEDGE valoracion-inferencias;

KNOWLEDGE-ROLE caso-abstracto;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
        vivienda-solicitante;
END KNOWLEDGE-ROLE caso-abstracto;

KNOWLEDGE-ROLE abstraccion-conocimiento;
    TYPE: STATIC;
    DOMAIN-MAPPING:
        abstraccion-vivienda FROM valoracion-conocimiento;
END KNOWLEDGE-ROLE caso-abstracto;

KNOWLEDGE-ROLE descripcion-caso;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
        vivienda-solicitante;
END KNOWLEDGE-ROLE descripcion-caso;

KNOWLEDGE-ROLE requisitos-especificos-caso;
```

```
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       SET-OF requisito-vivienda;
END KNOWLEDGE-ROLE requisitos-especificos-caso;

KNOWLEDGE-ROLE criterios;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       vivienda-criterios;
END KNOWLEDGE-ROLE criterios;

KNOWLEDGE-ROLE valor-criterios;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       vivienda-criterios;
END KNOWLEDGE-ROLE criterios;

KNOWLEDGE-ROLE resultados-evaluacion;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       vivienda-criterios;
END KNOWLEDGE-ROLE resultados-evaluacion;

KNOWLEDGE-ROLE decision;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       vivienda-decision;
END KNOWLEDGE-ROLE decision;

KNOWLEDGE-ROLE criterio-evaluacion;
    TYPE: DYNAMIC;
    DOMAIN-MAPPING:
       SET-OF vivienda-criterios;
END KNOWLEDGE-ROLE descripcion-caso;

KNOWLEDGE-ROLE requisitos;
    TYPE: STATIC;
    DOMAIN-MAPPING:
       requisito-vivienda FROM valoracion-knowledge;
END KNOWLEDGE-ROLE descripcion-caso;

KNOWLEDGE-ROLE decision-knowledge;
    TYPE: STATIC;
    DOMAIN-MAPPING:
       vivienda-regla-decision FROM valoracion-knowledge;
END KNOWLEDGE-ROLE descripcion-caso;

INFERENCE abstract;
    ROLES:
      INPUT:
       descripcion-caso;
      OUTPUT:
       caso-abstracto;
      STATIC:
       abstraccion-conocimiento;
```

```
    SPECIFICATION: "La entrada es un conjunto de datos de casos. La salida es el
mismo conjunto de datos ampliados con una función abstracta que se pueden derivar
de los datos utilizando el cuerpo de conocimientos abstracción";
END INFERENCE abstract;

INFERENCE specify;
    OPERATION-TYPE: lookup;
    ROLES:
      INPUT:
        descripcion-caso;
      OUTPUT:
        criterio-evaluacion;
    SPECIFICATION:
        "Esta inferencia es sólo una revisión simple de los criterios";
END INFERENCE specify;

INFERENCE select;
    ROLES:
        INPUT:
            criterio-evaluacion;
        OUTPUT:
            criterios;
    SPECIFICATION:
        "No se usa ningún conocimiento en la selección de hace al azar";
END INFERENCE select;

INFERENCE evaluate;
    ROLES:
      INPUT:
        criterios,
        descripcion-caso,
        requisitos-especificos-caso;
      OUTPUT:
        valor-criterios;
      STATIC:
        requisitos;
    SPECIFICATION: "Establece el valor de verdad de los criterios de entrada para
la descripcion caso dado. El conocimiento de dominio subyacente está formado tanto
por los requisitos de la base de conocimientos, así como de requisitos adicionales
del caso, que forman parte de la entrada";
END INFERENCE evaluate;

INFERENCE match;
    ROLES:
      INPUT:
        valor-criterios;
      OUTPUT:
        decision;
      STATIC:
        decision-knowledge;
    SPECIFICATION:
        "Comprueba si los resultados de la evaluación disponible, facilitar una
toma de decisión. Sino falla. ";
END INFERENCE match;
```

```
END INFERENCE-KNOWLEDGE

/* Tareas */

TASK-KNOWLEDGE valoracion-tareas;

TASK valorar-caso;
    DOMAIN-NAME: valorar-vivienda-solicitante;
    GOAL:
       "Valora si una solicitud para una vivienda de un determinado arrendatario
satisface los criterios ";
    ROLES:
      INPUT:
       descripcion-caso: "Datos sobre arredatario y vivienda";
       criterios-especificos-caso: "criterios especificos del caso";
      OUTPUT:
       decision: "eligible o no eligible para una vivienda";
END TASK valorar-caso;

TASK-METHOD valoración-por-abstraccion-y-emparejamiento;
    REALIZES:
       valorar-caso;
    DECOMPOSITION:
       TASKS: abstraer-caso, emparejar-caso;
    ROLES:
      INTERMEDIATE:
       caso-abstracto: "Caso original y sus abstracciones";
    CONTROL-STRUCTURE:
       caso-abstracto(descripcion-caso -> caso-abstracto);
       match-case(caso-abstracto+ requisitos-especificos-caso
              -> decision);
END TASK-METHOD valoración-por-abstraccion-y-emparejamiento;

TASK caso-abstracto;
    DOMAIN-NAME: abstraccion-datos-arrendatario;
    GOAL:
       "Añade las abstracciones del caso a la descripción";
    ROLES:
      INPUT:
       descripcion-caso: "Datos en bruto del caso";
      OUTPUT:
       caso-abstracto: "Datos en bruto y abstracciones";
END TASK caso-abstracto;

TASK-METHOD metodo-abstraccion;
    REALIZES:
       caso-abstracto;
    DECOMPOSITION:
       INFERENCES: abstract;
    CONTROL-STRUCTURE:
       WHILE NEW-SOLUTION abstract(descripcion-caso -> caso-abstracto) DO
       /* utilizar el caso abstracto como entreada en la invocación de la
inferencia abstracción*/
          descripcion-caso := caso-abstracto;
       END WHILE
```

```
END TASK-METHOD metodo-abstraccion;

TASK emparejar-caso;
    DOMAIN-NAME: emparejar-vivienda-solicitante;
    GOAL: "
        Aplicar los criterios al caso para determinar si los cumple";
    ROLES:
      INPUT:
        caso-abstracto: "descripción del caso y abstracciones";
        requisitos-especificos-caso: "Criterios específicos para una vivienda.";
      OUTPUT:
        decision: "Eligible o no eligible";
END TASK emparejar-caso;

TASK-METHOD metodo-emparejar;
    REALIZES:
        Emparejar-caso;
    DECOMPOSITION:
        INFERENCES: specify, select, evaluate, match;
    ROLES:
      INTERMEDIATE:
        criterio-evaluacion: "El conjunto completo de criterios de valoración";
        criterios: "Criterios de valoración";
        valor-criterios: "Valor de verdad de los criterios para este caso";
        resultados-evaluacion: "Lista de criterios y sus valores";
    CONTROL-STRUCTURE:
        specify(caso-abstracto-> criterio-evaluacion);
        REPEAT
            select(criterio-evaluacion -> criterios);
            evaluate(caso-abstracto+ requisitos-especificos-caso + criterios
                -> valor-criterios);
            resultados-evaluacion := valor-criterios ADD resultados-evaluacion;
        UNTIL
             HAS-SOLUTION match(resultados-evaluacion -> decision);
         END REPEAT
END TASK-METHOD metodo-emparejar;

END TASK-KNOWLEDGE valoracion-tareas;

END KNOWLEDGE-MODEL
```

B.1 Descripción del proyecto Producción Integrada en agricultura

La agricultura actual y en concreto la del sureste de España está sometida a un gran dinamismo en todos sus ámbitos, y en especial la "Sanidad Vegetal". La cual requiere ante un problema más o menos complejo una toma de decisiones rápida basada en muchos factores interrelacionados de tipo agronómico, tecnológico, ecológico, legal y económico.

La mayor inquietud alimentaria de los consumidores europeos, ante las recientes y continuadas crisis provocadas por el mal de las vacas locas, los pollos con dioxinas y la fiebre aftosa, entre otras, está exigiendo la puesta en marcha de normas de aseguramiento de la salubridad de los productos.

El respeto al medio ambiente en la obtención de los productos hortofrutícolas, la seguridad y salud de los productores, la exigencia de frutas y hortalizas limpias de residuos químicos son algunas de las demandas de los clientes en destino.

Esta situación está conduciendo al campo almeriense a una apuesta seria por la implantación de Normas de Calidad, como uno de los principales criterios de diferenciación y competitividad, existiendo en la actualidad un total de 31 cooperativas y Sociedades Agrarias de Transformación con sistemas de producción acogidos a normas de calidad.

La Junta de Andalucía a través de la Consejería de Agricultura y Pesca lleva ya más de 11 años desarrollando la Producción Integrada, Norma de Calidad pionera en nuestra comunidad autónoma y en fuerte desarrollo en otras regiones españolas como Valencia , Cataluña y países como Italia. La experiencia de todos estos años y lugares, han puesto de manifiesto que tanto ésta como el resto de Normas de Calidad tienen una fuerte carga técnica, que lleva implícita una alta exigencia de información en los diversos campos de la producción vegetal.

Para conocer algo de la terminología de este dominio de aplicación conviene revisar la evolución que ha tenido el control de plagas hasta llegar a esta norma de calidad de la producción integrada

La lucha integrada como traducción de "integrated control" tuvo su origen en los años 50, como término para expresar la integración, sentida como necesaria, de la lucha química y de la lucha biológica. Este término ha ido

evolucionando: "Lucha integrada contra plagas""gestión integrada de plagas", "protección integrada de plantas", cada una aportando un nuevo matiz.

En 1967 la FAO definió la lucha integrada como: "Un sistema de protección contra los enemigos de los cultivos teniendo en cuenta el medio particular y la dinámica de las poblaciones consideradas. Utiliza todas las técnicas y medios apropiados, de forma tan compatible como sea posible, con el objetivo de mantener las plagas a niveles tales que sus perjuicios sean económicamente tolerables".

En 1977 la organización para la lucha biológica e integrada sustituyó esta definición por la siguiente: "Un proceso del lucha contra organismos nocivos utilizando un conjunto de métodos que satisfagan las exigencias económicas, ecológicas y toxicológicas, dando un carácter prioritario a las acciones que fomentan la regulación de los enemigos de los cultivos empleando medios naturales y respetando los umbrales económicos de tratamiento".

Antes de llegar al concepto de lucha integrada se ha pasado por diferentes fases:

- Lucha química indiscriminada, basada en calendarios
- Lucha aconsejada, guiada por las estaciones de avisos
- Lucha dirigida o razonada, introducción de los umbrales de tolerancia y elección de plaguicidas según su repercusión ecológica.
- Lucha integrada que no desprecia la lucha química, sino que la limita su aplicación a los casos en los que es realmente necesaria.

En la situación actual de la evolución del medio agrícola se habla de producción integrada, este último concepto va un poco más lejos, aparece la necesidad de abordar no sólo la lucha contra plagas, sino también cuestiones relativas a rentabilidad y el resto de tareas propias de la producción agrícola. Con el objetivo de general productos de mayor calidad.

Se entiende por producción integrada al sistema agrícola de producción de vegetales que utiliza al máximo los recursos y mecanismos de producción naturales, asegurando a largo plazo una agricultura sostenible. En ella los métodos biológicos, químicos y otras técnicas son cuidadosamente elegidos y

equilibrados, teniendo en cuenta las exigencias de la sociedad, la rentabilidad y la protección del medio ambiente.

Para la preservación de estos modelos de producción agrícola se ha creado la marca de calidad Producción Integrada que se realiza de acuerdo con la oficina española de patentes y marcas y cuenta con una extensa normativa que la regula. El seguimiento y la inspección del cumplimiento de las normativas la realizan departamentos de las comunidades autónomas.

Un aspecto muy importante dentro de la producción integrada es la lucha integrada contra la plagas de plagas definida anteriormente, aunque no es el único.

El objetivo del sistema a definir es el seguimiento, asesoramiento y control de los cultivos de producción integrada. Cubre dos perspectivas la de asesoramiento y la de inspección. Por una parte se debe vigilar el cumplimiento de una normativa y por otra es necesario asesorar a agricultores y técnicos agrícolas de las prácticas más apropiadas para el desarrollo de su cultivo

B.2 Modelo de Negocio. Producción integrada

NEG-Proyecto 1

Definición del proyecto	Estudio del protocolo de producción integrada en agricultura, en especial para cultivos de tomate bajo plástico y parral, intentando mejorar su implantación que según datos de la consejería Andaluza de agricultura y pesca es menor de lo esperado.
Versión	Versión 1, 22-04-2006
Autores `	Isabel María del Águila Cano – Universidad de Almería
Fuentes	Representantes del sector agrícola, responsables de la elaboración y estudio del protocolo de la producción integrada en Almería
Problemas y oportunidades	Gestión de la marca Producción Integrada (P.I.) Asesoramiento en técnicas aplicables Ofrecer acceso a información
Contexto organizacional	Misión, objetivos generales de la organización: Controlar el cumplimiento de la marca de calidad Racionalizar el usos de pesticidas Dar asesoramiento al agricultor.
	Factores externos importantes con los que se relaciona la organización: - Contexto cambiante. Cambiante entre campañas resistencias) - Mercado - Se trabaja dentro de un invernadero. El técnico y / o agricultor necesita asesoramiento "inmediato" a la vista de las plantas. - Normativa
	Estrategia de la organización: Preservar el medio ambiente
	Escala de valores por la que se rige: Ecología. Rentabilidad sin dañar el medio.
Soluciones	SBC- Asesoramiento sobre qué actuación se ha de realizar FITO- Acceso a datos fitosanitarios Comunicación eficiente INTERNET Recolección "automática" de los datos. Muestreos. Control de los datos de las asociaciones de producción integrada APIs
Estructura	Se presenta un gráfico de la organización bajo análisis en términos de departamentos, grupos, unidades,...**G1**
Proceso	Diagrama de procesos a llevar a cabo. **G2.** Básicamente existen dos grandes procesos. Gestión de la marca de calidad (con revisiones que ello implica) Proceso de asesoramiento en cuanto a la gestión del cultivo (Control integrado) Supervisión del estado fitosanitario de la provincia. En el diagrama G2 aparece otro gran proceso que son las tareas de comercialización de las que se sabe poco.

Personas	Personal ligado al SPV. Técnicos agrícolas y personal auxiliar de los diferentes departamentos. Técnicos de API.
Recursos	1.- Sistemas de información y otros recursos utilizados: -Registro de Madrid, -Consejería. -Sistema de gestión y acceso a información fitosanitaria que se tiene que construir. 2.- Equipamiento y material. -Red de Ordenadores del servicio SPV -Recursos asignados al proyecto -Material de recogida de datos de campo, sensores, portátiles,.... 3.- Experiencia social, interpersonal que no sean intensas en conocimiento. - Colaboración con cooperativas 4.- Tecnología, patentes, etc - Normativa 5.- Trabajos de investigación previos Contexto fitosanitario Contexto medico
Conocimiento	Normativa en Umbrales + Normativa general Normas en el conteo Esquemas de corrección de umbrales. Reflejan experiencia de los técnicos. Criterios de valoración de las actuaciones sobre cultivo (t. biológicos, t. químicos y medidas culturales) Planes de riego y fertilización Historia pasada --- Factores de riesgo
Cultura y capacidad	El trabajo con los agricultores no suele ser fácil. Están los técnicos que llevan cultivos y los que supervisan las actuaciones con los cultivos. Existe una normativa estricta que hay que cumplir. Nosotros trabajamos con quienes han hecho la normativa, no con quienes la utilizan.

La figura B.1 muestra como casos de uso los procesos de negocio desarrollados dentro del marco de la aplicación de la norma de calidad PI. El flujo de procesos se puede describir diciendo que tras crear la asociación de agricultores que recogerá la marca de calidad se inicializan los cultivos que son sometidos a un seguimiento, y sobre el que se realizará actuaciones como la recolección, los tratamientos con fitosanitarios, la fertirrigación o el laboreo según la decisión del técnico. De estas actuaciones, el sistema descrito en este trabajo se centra en el control integrado de plagas. Además, los productos recolectados y el propio invernadero deben ser inspeccionados para controlar el cumplimiento de

la marca de calidad y la producción debe ser puesta en el mercado tras ser tratada en los centros de distribución

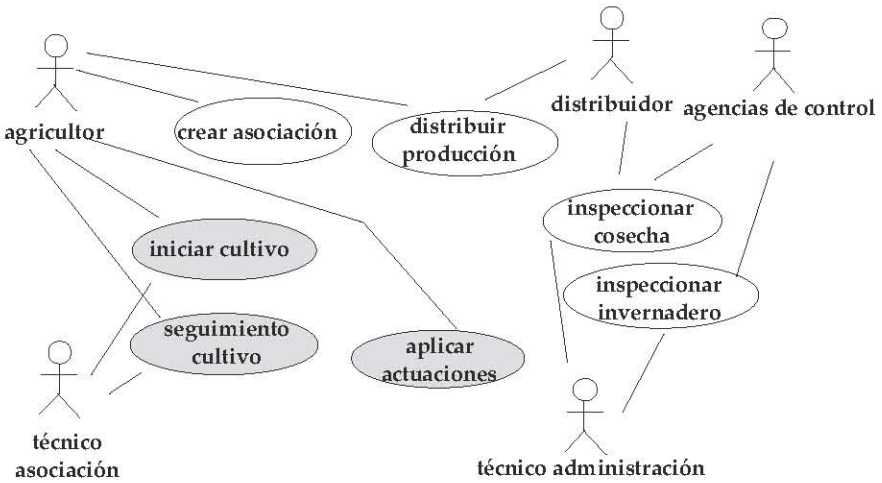


Figura B.1 Procesos NEG en la Producción Integrada.

NEG-Objetivos

OBJ-1	Crear Asociación
Versión	22-04-2006
Autores	Isabel María del Águila Cano
Fuentes	SPV de Almería.
Descripción	Se deben formar las agrupaciones de tratamiento integrado formadas por varios agricultores y normalmente ligadas a una cooperativa o entidad ya constituida
Subobjetivos	No se consideran
Realizado por	Grupo de agricultores
Donde se desarrolla	No relevante
¿Intensivo en conocimiento?	No
Importancia	5
Urgencia	..
Estado	..
Estabilidad	Máxima
Comentarios	Conlleva

OBJ-3	Seguimiento
Versión	22-04-2006
Autores	Isabel María del Águila Cano
Fuentes	SPV Almería
Descripción	En las visitas semanales, el técnico revisa el cultivo, muestreando los estados de los patógenos para en base a estos datos proceder a recomendar una actuación sobre el cultivo
Subobjetivos	
Realizado por	Técnico agrícola y agricultor
Donde se desarrolla	En el campo
¿Intensivo en conocimiento?	SI
Importancia	10
Urgencia	10
Estado	Inestable, muchos cambios en protocolos
Estabilidad	
Comentarios	Lleva mucho tiempo, implica la elaboración de recetas y registro de datos de muestreos

B.3 Modelo de servicios de Producción Integrada

El modelo de servicios realiza un refinamiento de los objetivos de la organización, identificando el conjunto de servicios que se llevan a cabo para alcanzar dichos objetivos. En nuestro caso, el objetivo perseguido por el sistema de asesoramiento experto es el seguimiento del cultivo, donde se indica al agricultor las actuaciones a realizar sobre el cultivo en base a las observaciones realizadas en las visitas semanales del técnico y ajustándose al protocolo de la marca de calidad PI.

Modelamos el objetivo seguimiento del cultivo aplicando descomposición funcional y se obtienen los servicios mostrados mediante un diagrama de flujo de datos en la figura B.2 Tenemos cinco procesos: Muestrear, Proponer actuación, Estimar riesgo, Fijar actuación y Anotar actuación.

Durante el proceso de muestreo se seleccionan las plantas a observar bien de forma aleatoria o supervisada (en las bandas o zonas de riesgo), recogiendo la información necesaria para indicar que actuación se debe realizar dependiendo del cultivo y de la plaga. La actividad más importante es el conteo de la plaga y la fauna auxiliar, es decir, cuantos individuos (o los síntomas de su presencia) hay de cada especie. Además de calcular la incidencia de cada plaga, en este proceso se obtiene información de factores externos y del estado de las plantas. Se trata de una tarea lenta y mecánica.

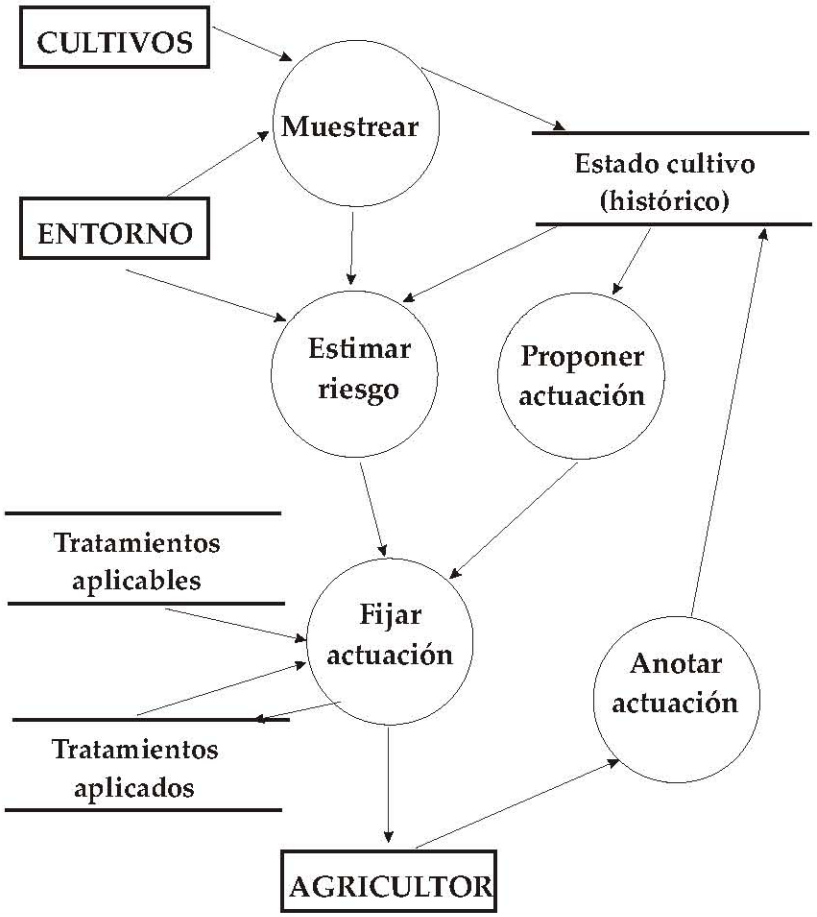


Figura B.2 SER en la Producción Integrada.

En base al historial del cultivo se puede adelantar que tipo de actuaciones se han de realizar proponiendo una actuación. Para la fertirrigación existen protocolos donde según la fenología del cultivo se ha de regar con una determinada concentración de abonos. Igual ocurre con las medidas culturales como la poda, destallado o recolección. Estas actuaciones se deben confirmar tras la revisión y el muestreo. Pero en el control integrado de plagas no existen protocolos. Se fundamenta en el recuento de agentes nocivos y sus enemigos naturales. Solo se aplica este proceso para los protocolos en fertirrigación, medidas culturales y recolección.

La **estimación del riesgo** consiste en decidir en base a la información del muestreo y los datos de semanas anteriores, si existen factores de riesgo que puedan afectar las actuaciones o llevar a fijar nuevas actuaciones, es decir, se

decide si es necesario realizar una actuación de emergencia. En el caso del control de plagas es la tarea en donde según los umbrales de tratamiento fijados por la marca de calidad (valores mínimos que se han de alcanzar para poder actuar) se decide si es necesario aplicar un tratamiento, pero incluso superando los umbrales, puede que no se decida tratar porque otras observaciones no lo hagan necesario.

Durante el proceso **fijar actuación** se concretan las actuaciones a realizar, indicando dosis y cuál es el producto fitosanitario (materia activa) a aplicar según los datos recogidos.

La labor administrativa **anotar actuación** consiste en anotar sobre el cuaderno de campo del cultivo que actuación se ha realizado para los tratamientos, donde se registra el producto comercial aplicado por el agricultor de entre los posibles en el mercado que contengan la materia activa recomendada por el técnico.

Se refleja todas las actuaciones que se pueden realizar sobre el cultivo. Pero recordemos que el control fitosanitario integrado sólo se ocupa del control de los agentes nocivos. Los dos tipos de tratamientos aplicables son: tratamientos químicos y biológicos. Por tanto, en la definición del modelo de conocimiento sólo nos quedamos con los procesos: Muestrear, Estimar riesgo y Fijar Actuación de tratamiento, de los cuales los dos últimos son los que necesitan intensamente el conocimiento, ya que como hemos indicado el muestreo es mecánico y su automatización vendría de la mano de técnicas robóticas.

Un resultado importante que en el proceso de asesoramiento, el experto diferencia entre dos tareas intensivas en conocimiento: la estimación del riesgo debido a la plaga y el fijar o asesorar un tratamiento. La decisión se toma a dos niveles. En primer lugar, se decide si es necesario una acción de control sobre el cultivo, estimando el riesgo asociado a las plagas muestreadas. En segundo lugar, se selecciona la combinación de productos químicos o biológicos a aplicar, respetando lo más posible la fauna útil y otros productos biológicos aplicados previamente y la normativa de la marca de calidad.

B.4 Modelo de conocimiento de Producción Integrada

La construcción del modelo de conocimiento se realiza siguiendo las pautas de CK, aplicando las tareas descritas en la sección 5.3 "Flujo de trabajo de Conocimiento" : C1_Identificación, C2_Especificación y C3_Refinamiento.

En C1_Identificación se identifican las fuentes de información útiles para modelado del conocimiento. Se construye un glosario de términos del dominio. Se revisan componentes de modelado ya construidos como modelos de tareas básicas o esquemas y ontologías del conocimiento del dominio. Estos componentes podrán ser reutilizados en el proyecto, partimos del modelo de organización, centrándonos los elementos descritos sobre el dominio y las tareas. En el caso de SAEPI gran parte del conocimiento del dominio se ha reutilizado de trabajos anteriores del grupo de investigación (Águila & Túnez, 2007), (Túnez, Águila, & Marín, 2001) y (Túnez, Marín, Águila, Bosch, & Torres, 1998), adaptándolos para el uso de normas de calidad como la PI. En este trabajo también se desarrollo un método para la tarea de planificación de terapia que se basada en razonamiento abductivo para resolver el problema de tratamiento de plagas sólo para la lucha química, que es un elemento importante a ser reutilizado en este nuevo proceso de modelado.

En C2_Especificación, el ingeniero del conocimiento comienza con la construcción de la especificación del modelo de conocimiento empleando diagramas y CML. Los componentes reutilizables seleccionados en la fase anterior son el marco para la especificación. Para modelar el conocimiento de las tareas que aborda el SBC se selecciona una plantilla de tarea básica (o una combinación de varias) que se corresponda con las características de la tarea a modelar. El ingeniero del conocimiento tendrá que "rellenar los agujeros" que aparezcan en la reutilización de componentes ya existentes.

En C3_ Refinamiento, pretende la validación del modelo de conocimiento y la captura de la mayor cantidad de conocimiento o instancias de conocimiento para insertarlas en la base de conocimiento ligada a ese modelo. La validación consiste en determinar si el modelo construido puede generar el comportamiento requerido en la resolución del problema. La técnica utilizada durante la validación suele ser la simulación en papel o a través de un prototipo del modelo de conocimiento.

Estas tres actividades están entrelazadas, no se tienen porqué desarrollar de forma secuencial, es habitual necesitar iteraciones, por ejemplo, para rellenar el modelo de dominio podemos necesitar recurrir a nuevas fuentes de información que han de ser identificadas.

B.4.1 Selección de componentes básicos de modelado

El catalogo de plantillas de tareas de CommonKADS se puede utilizar para modelar cualquier tipo de problema. En nuestro caso, el problema es asesorar acerca de cuál es la terapia a administrar para reparar un problema en el dominio del control de plagas dentro de la norma de calidad PI. Así, asignaremos el nombre de Administración de Terapia al problema a modelar que se instanciará sobre el dominio de la PI en agricultura, que también tiene que ser modelado. Para la InCo en general se distinguen dos tipos de tareas básicas: de análisis y de síntesis. Las primeras tienen por objetivo identificar o clasificar los componentes o atributos desconocidos de un sistema conocido. El objetivo de las segundas es ensamblar una descripción estructural de un sistema, inicialmente desconocido, a partir de sus partes conocidas. La tabla B.1 muestra la taxonomía de plantillas de tareas básicas de CommonKADS.

Tabla B.1 Tipos de tareas de CommonKADS.

Tipo de tarea	**Entrada**	**Salida**	**Conocimiento**	**Características**
Análisis	Observaciones sobre el sistema	Caracterización del sistema	Modelo del sistema	Se da una descripción del sistema
Valoración	Descripción del caso	Tipo de decisión	Criterios Normas	La valoración se realiza en un momento puntual en el tiempo.
Diagnóstico	Síntomas/ quejas	Tipo de fallo	Modelo de comportamiento del sistema	La salida varia (cadena causal, estado, componente) y depende del uso que se haga (reparación)
Monitorización	Datos del sistema	Clase de discrepancia	Comportamiento normal del sistema	El sistema cambia con el tiempo. La tarea se repite cíclicamente
Clasificación	Características del objeto	Clase del objeto	Asociaciones clase-característica	Conjunto de clases predefinidas

Predicción	Datos del sistema	Estado del sistema	Modelo de comportamiento del sistema	La salida es la descripción del sistema en un tiempo futuro
Síntesis	Requisitos	Estructura del sistema	Elementos, Restricciones, Preferencias	Se necesita generar una descripción del sistema
Planificación	Objetivos Requisitos	Plan de acción	Funciones, Componentes, Diseños esqueletales, Restricciones, Preferencias	Puede incluir el diseño creativo de los componentes
Diseño	Requisitos	Descripción del artefacto	Funciones, Componentes, Restricciones, Preferencias	Es un subtipo del diseño en el que todos los componentes están predefinidos.
Diseño de configuraciones	Requisitos	Descripción del artefacto	Acciones, Restricciones, Preferencias	Las acciones están parcialmente ordenadas en el tiempo.
Programación Temporal	Trabajos, recursos, unidades de tiempo, recursos	Trabajos asignados a momentos de tiempo y recursos	Restricciones y preferencias	El carácter orientado al tiempo la distingue de la asignación
Asignación	Dos conjuntos de objetos	Correspondencias entre conjuntos	Restricciones, preferencias	El reparto no tiene porque ser uno a uno

Para problemas de administración de terapia como el considerado, no existe coincidencia exacta con ninguna de las tareas básicas anteriores, una alternativa es recurrir a componentes más elementales para construir el modelo de conocimiento, olvidando la reutilización a nivel de tarea genérica; pero tal como propone la propia metodología, en ocasiones se puede abordar el modelado desde la combinación de algunas de las plantillas de tarea. De hecho, existen un conjunto clásico de posibles combinaciones de las tareas básicas, que se describen en la tabla B.2. Si enlazamos estas combinaciones podemos proponer cadenas como: monitorización + valoración + planificación o bien monitorización + diagnóstico + planificación, como se muestran en la figura B.3.

Cabe decir que las tareas de planificación y tratamiento se pueden considerar equivalentes (Túnez, Águila, & Marín, 2001). El objetivo del tratamiento es diseñar un plan terapéutico, es decir, un plan cuyas acciones consisten en la administración de productos terapéuticos, y cuyo objetivo es

remediar un proceso anómalo previamente identificado. La estructura del plan consiste en una secuencia de acciones de administración y los elementos del plan son las propias acciones de administración. Cada una de estas acciones especifica un producto terapéutico y unos detalles de administración (dosis, forma de aplicación,...). Por esta razón, es frecuente referirse a la tarea de tratamiento como una tarea de planificación terapéutica. Como ya hemos indicado la planificación de terapia ya ha sido modelada con éxito en trabajos anteriores y disponemos de una plantilla modificada apropiada para este dominio (Túnez, Marín, Águila, Bosch, & Torres, 1998).

Tabla B.2 Combinación de tareas básicas.

	Descripción
monitorización + diagnóstico	Combinación habitual, puesto que la salida de una es la entrada de la siguiente. Muchos de los sistemas técnicos actuales incluyen esta funcionalidad de monitorización y diagnóstico
monitorización + valoración	En aplicaciones donde no hay la posibilidad "real" del diagnóstico debido a la no disponibilidad de los expertos o la ausencia de modelos causales apropiados la monitorización va seguida de una valoración, en la que la decisión se toma según la situación.
diagnóstico + planificación	El resultado del diagnóstico suele utilizarse como un objetivo para la planificación de acciones correctivas (trouble-shooting)
valoración + planificación	Un ejemplo son los planes de recuperación

En otros trabajos también se han relacionado estos tres tipos de tareas monitorización+diagnóstico+tratamiento (Breuker & Velde, 1994). El proceso comienza con una tarea de monitorización que genera un conjunto de situaciones de error (discrepancias con el comportamiento normal esperado). A continuación, se busca la causa de la discrepancia capaz de explicar el comportamiento observado. Finalmente se prescribe un tratamiento para detener el problema. Esta es la combinación que mejor se adapta a la administración de terapia, sin embargo las tareas de monitorización y diagnóstico tienen diferentes matices en el dominio agrícola y necesitan ser adaptadas. El daño se detecta mediante la inspección del cultivo y la hipótesis de diagnóstico es siempre una plaga o enfermedad que afecta al cultivo. El experto identifica el agente

perjudicial mediante la observación directa de las plantas en el muestreo y después analiza los datos recolectados para detectar un desequilibrio, lo que corresponde también con una tarea de valoración, más que un diagnostico o lo que llamaremos diagnóstico por valoración.

Consecuentemente, hemos seleccionado como componentes potenciales para la reutilización las plantillas de las tareas básicas de CommonKADS de monitorización, diagnostico y valoración, además el modelo previo de plantificación de terapia ya desarrollado por el grupo. La figura B.3 muestra la versión cero del modelo de conocimiento, que se obtiene de la fase de identificación, el ingeniero del conocimiento debería adaptar y completar estos componentes para construir el modelo de conocimiento completo para nuestro problema, en la misma figura se muestran los diagramas de inferencias correspondientes a estas tareas básicas y en las zonas sombreadas aparece la porción de la plantilla que se ha adaptado.

Los diagramas de inferencias de la figura B.3 nos describen de forma general el proceso de razonamiento modelado en las tareas (se destacan las inferencias utilizadas en estos procesos de razonamiento):

La tarea de **monitorización** recibe un nuevo dato normalmente cada cierto intervalo de tiempo y se selecciona un parámetro del sistema que nos pueda decir algo acerca del nuevo dato, a partir de un modelo de lo que se espera del sistema, especificando el valor normal para ese parámetro y este valor normal se compara con este nuevo dato. Si se detecta una diferencia y se considera o no una discrepancia clasificándola según los valores recogidos anteriormente en otros ciclos de monitorización. Se hace necesario tener conocimiento del comportamiento normal del sistema y esta tarea se repite cíclicamente, cada intervalo de tiempo o cada vez que se recibe u se obtiene un nuevo dato.

La tarea de **diagnóstico** parte de unos síntomas y busca cual es el conjunto de posibles hipótesis diagnósticas que cubran o justifiquen estos síntomas. Una a una estas hipótesis se van seleccionando para verificar si los datos observados en el sistema la confirman o la descartan. En base a las posibles hipótesis diagnósticas se especifican los datos a obtener del sistema.

La tarea de **valoración** fija una categoría de decisión según los datos del caso. Aplicando abstracción se define un caso abstracto partiendo de los datos concretos del caso, que permite especificar los criterios a aplicar en la valoración. Uno a uno se selecciona cada uno de estos criterios que es evaluado, y tras esto se

comprueba si concuerdan con los valores establecidos para tomar una decisión. La valoración se realiza en un momento puntual de tiempo

La tarea de **tratamiento** resuelta con el método abductivo es una tarea compleja que se no se describe mediante inferencias, sino con su descomposición en otras tareas que han de seguir descomponiéndose hasta llegar a las inferencias. Sería demasiado extenso mostrar el proceso de adaptación del tratamiento que tendría que comenzar por describir y comprender este modelo hasta el nivel de inferencias. En este trabajo nos centraremos en la descripción del proceso de adaptación de las tareas de monitorización y diagnóstico por valoración, descritas en términos de inferencias, para resolver la tarea seleccionar objetivos terapéuticos.

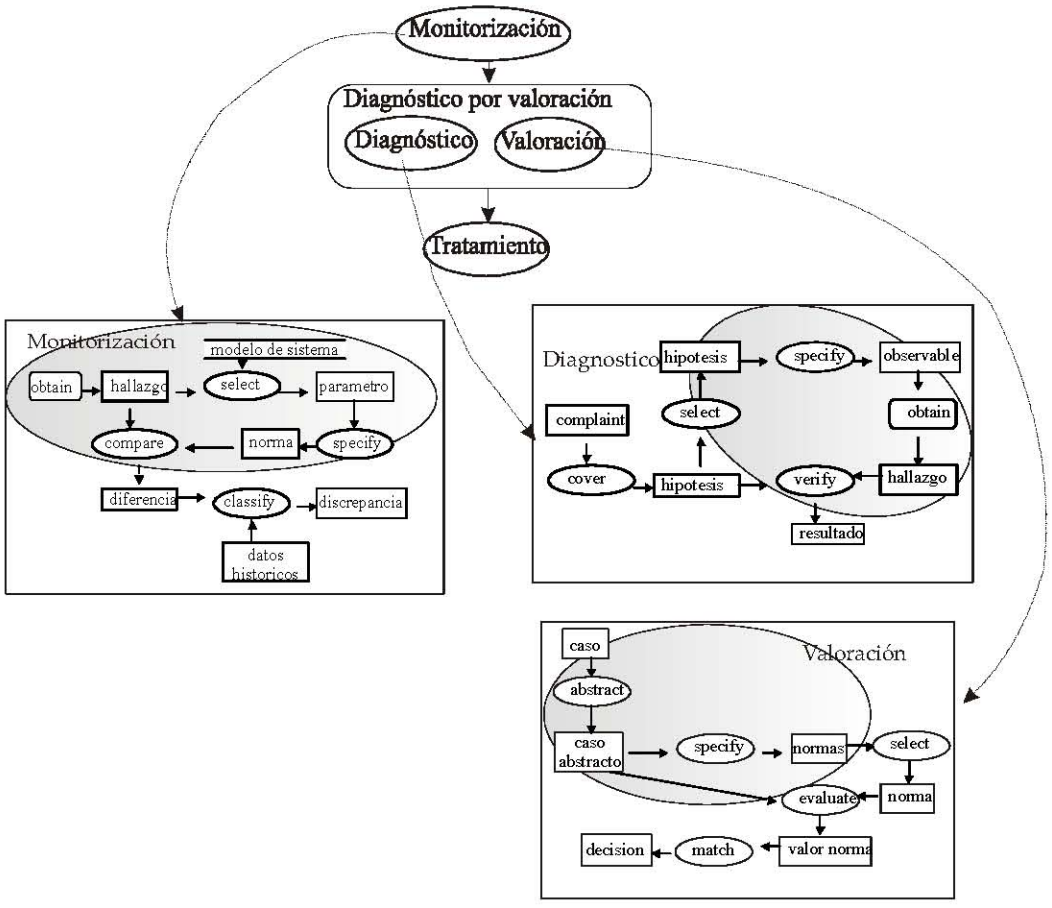


Figura B.3 Tareas básicas reutilizadas en CON.

B.4.2 Conocimiento de tareas y del dominio

Se modela el conocimiento de tareas y del dominio, después se hará corresponder con el proceso de razonamiento del conocimiento de las inferencias. Las tareas e inferencias se modelan de forma general para poder reutilizarse en dominios similares, después es necesario hacer un mapeado sobre los conceptos del dominio o roles.

De esta forma, se parte de una tarea genérica **administración de terapia** que se resuelve por descomposición en dos subtareas: **selección de objetivos terapéuticos** y **tratamiento**, reflejando el modo en que el experto realiza la tarea de asesoramiento terapéutico para la PI, tal como mostramos en la figura B.4. Para resolver la tarea de tratamiento hemos reutilizado el método de planificación de terapia basada en razonamiento abductivo, cuyo proceso de adaptación para la producción integrada no se trata en este trabajo tal como ya hemos justificado en el apartado anterior y nos centraremos por tanto en la tarea selección de objetivos terapéuticos.

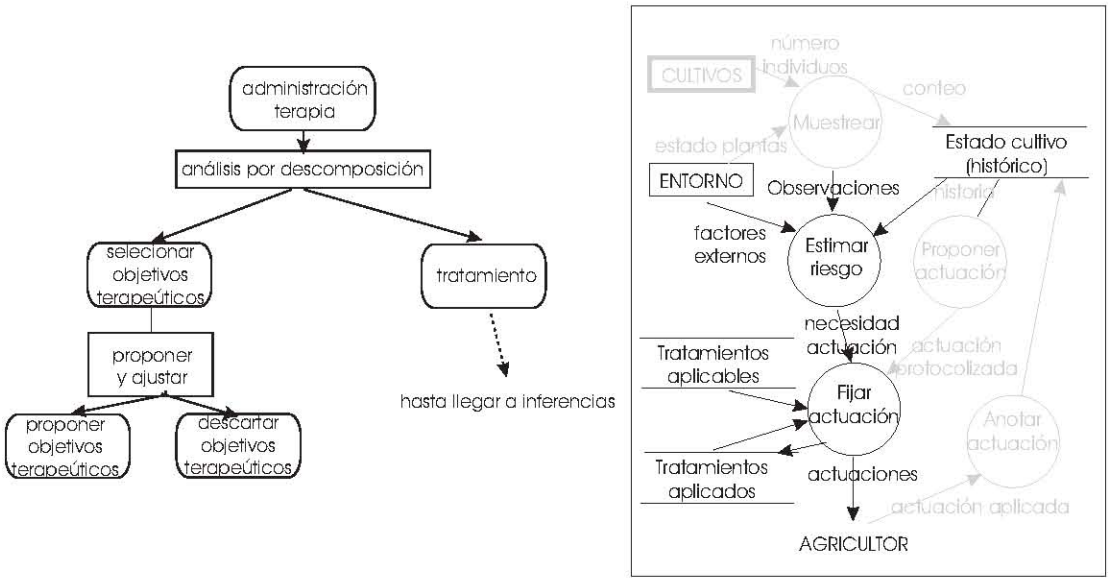


Figura B.4 Conocimiento de tareas.

La selección de objetivos terapéuticos cuyo objetivo, en nuestro caso, será seleccionar de entre todas las posibles plagas que puedan afectar al cultivo, aquellas que por su incidencia lleguen a desequilibrar el sistema, se resuelve aplicando el método **proponer y ajustar.** Este método realiza la selección de objetivos terapéuticos gracias a dos las tareas: **proponer objetivos terapéuticos** y **descartar objetivo terapéutico**. Estas dos tareas se definen en términos de inferencias primitivas. CommonKADS plantea la descripción de la tarea de forma independiente al método que se aplique para resolverla; de esta forma, puede producirse un cambio de método que afecte sólo a las tareas descendientes. Una tarea puede resolverse mediante varios métodos alternativos. La descripción CML de cada tarea tiene dos partes diferenciadas tal como se muestra en la tabla B.3, la propia tarea (TASK) y el método por el que se resuelve (TASK-METHOD), en la tabla se incluyen las descripciones CML de las tareas.

Este modelo de conocimiento refleja el mecanismo que aplica el técnico agrícola durante las visitas a los cultivos. Partiendo de unos formularios donde se registran las observaciones sobre plantas elegidas al azar, el técnico fija la presencia perjudicial de determinadas plagas, es decir propone unas plagas como objetivos terapéuticos. Para poder tomar la decisión sobre la necesidad de actuar sobre estas plagas candidatas, se consideran datos adicionales que permiten convertirlas definitivamente en objetivos terapéuticos sobre los que recomendar una actuación o descartarlos para la tarea de tratamiento. Para entender el proceso de modelado del conocimiento completo, este conocimiento de tareas se debe relacionar con los conceptos del dominio y lo que se llaman objetivos terapéuticos, después serán patologías concretas.

En paralelo al proceso anterior, se tiene que definir el modelo del conocimiento del dominio para modelar los conceptos y sus relaciones. Este modelo refleja la ontología del dominio y las estructuras y relaciones entre los conceptos para la administración de terapia fitosanitaria. La figura B.5 muestra una porción de este modelo, la relativa a la tarea de selección de objetivos terapéuticos. En las **Visitas** semanales a las **Parcelas** se **Muestrea** el cultivo y se realizan **Observaciones** de otra información de interés que puede detectar la Infestación de las Plantas con algún **Agente** nocivo lo que detecta una posible **Patología** que debe llevar asociada un conjunto de **Acciones**.

Tabla B.3 Código CML del modelo de tareas.

TASKS	METHODS APPLIED
TASK **administración_terapia** ROLES: INPUT: caso:"Datos de la visita al cultivo"; OUTPUT: plan_tratamiento: "secuencia de acciones terapeuticas "; END TASK administración_terapia;	TASK METHOD analisis_por_descomposicion REALIZES:administracion_terapia; DECOMPOSITION: seleccionar-objetivos-terapeuticos, tratamiento CONTROL-STRUCTURE: seleccionar-objetivos-terapeuticos (caso -> objetivos-terapeuticos); tratamiento(objetivos-terapeuticos -> plan-tratamiento); END TAKS METHOD analisis_por_descomposicion
TASK **seleccionar- objetivos-terapeúticos** ROLES: INPUT: :"Datos de la visita al cultivo"; OUTPUT: objetivos-terapeúticos-:"problemas a resolver"; END TASK seleccionar-objetivos-terapeúticos;	TASK METHOD proponer-y-ajustar REALIZES: seleccionar-objetivos-terapeúticos DECOMPOSITION: proponer-objetivos-terapeúticos, descartar-objetivos-terapeúticos; CONTROL-STRUCTURE: proponer-objetivos-terapeúticos (caso -> objetivos-terapeúticos); *FOR EACH objetivos-terapeútico IN* objetivos-terapeúticos *DO* descartar-objetivos-terapeúticos (objetivos-terapeúticos + nuevos_hallazgos -> resultado); IF resultado = false objetivos-terapeúticos:= objetivos-terapeúticos SUBTRACT objetivos-terapeútico; END FOR-EACH END TAKS METHOD proponer-y-ajustar
TASK **tratamiento** ROLES: INPUT: objetivos-terapeúticos-:"problemas a resolver"; OUTPUT: plan_tratamiento: "secuencia de acciones terapeuticas "; END TASK tratamiento;	TASK METHOD tratamiento-abductivo REALIZES:tratamiento; DECOMPOSITION: ..... CONTROL-STRUCTURE: ..... END TAKS METHOD tratamiento-abductivo
TASK **proponer-objetivos-terapeúticos** ROLES: INPUT: caso:"Datos de la visita al cultivo"; OUTPUT: objetivos-terapeúticos-:"problemas candidatos a ser resueltos"; END TASK proponer-objetivos terapeúticos;	(heredados de las plantillas de tareas, descritos en B.4.3)

TASK **descartar-objetivos-terapeúticos** ROLES: INPUT: objetivos-terapeúticos-:"problemas candidatos a ser resueltos", nuevos-hallazgos; OUTPUT: objetivos-terapeúticos-:"problemas candidatos a ser resueltos";' END TASK **descartar-objetivos-terapeúticos;**	

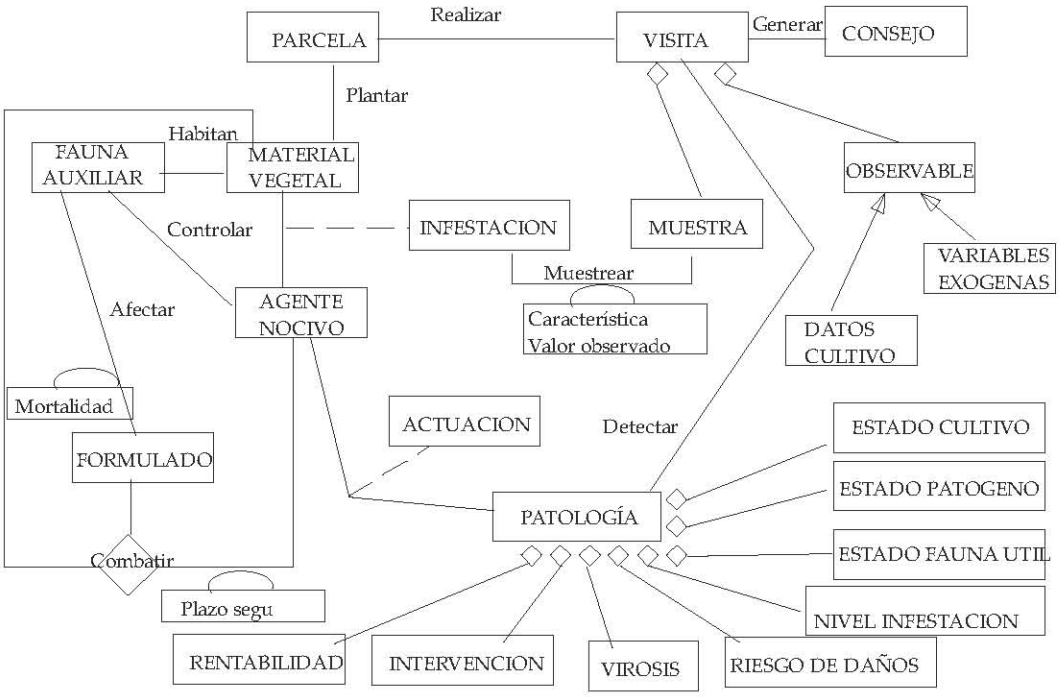


Figura B.5 Conocimiento del dominio.

B.4.3 Conocimiento de las inferencias

Para resolver las subtareas *proponer y descartar objetivos terapéuticos* se adaptan de las plantillas de las tareas básicas monitorización, diagnóstico y valoración, como se ha indicado en el apartado anterior. A continuación analizaremos los ajustes realizados para generar el modelo conocimiento completo.

En el problema de administración de terapia en agricultura se parece a la tarea básica *monitorización*, donde se obtiene un dato y se compara con lo que se

espera del sistema para detectar discrepancias, pero sólo se consideran los datos recolectados en una visita semanal, no se mantienen datos de la evolución. Por otra parte, la discrepancia es siempre la presencia descontrolada de un agente nocivo, del cual se están obteniendo los datos o medidas. Esto lleva a que sólo sea útil en nuestro caso la primera parte de la plantilla, tal como se muestra de forma aproximada en la zona sombreada en la figura B.4. Pero esta tarea de monitorización no es suficiente, puesto que no refleja el análisis o reconsideración de otros datos para descartar los objetivos terapéuticos aun superados los umbrales de tratamiento, ni tampoco se puede expresar el hecho de que los datos que se obtienen y se registran; es decir, el patrón de observación, depende del caso, puesto que no se observa lo mismo si el cultivo es de tomate o de pepino. La primera deficiencia se inyecta sobre el conocimiento de las inferencias de la tarea de diagnóstico que veremos a continuación y la segunda de la de valoración.

La tarea básica *diagnóstico* propuesta por CK se basa en la cobertura causal. Se parte de unos síntomas y genera un conjunto de posibles hipótesis diagnósticas que puedan justificarlos, éstas son verificadas o descartadas una a una con nuevos datos observados del problema. En nuestro caso, la hipótesis es siempre una plaga sobre la que se debe actuar o potencial objetivo terapéutico (que ha sido definido como tal en base a los umbrales de tratamiento) y que, según los datos, pueden ser descartados como objetivos terapéuticos. También es necesario indicar recomendaciones adicionales y en cada visita se generan valores indicadores del estado del cultivo, que reflejan el nivel de infestación del cultivo debido a todos los objetivos terapéuticos ya verificados.

La plantilla de *valoración* parte de los datos del caso y se define un caso abstracto para especificar los criterios o normas a aplicar que se evalúan para tomar una decisión. Para nuestro caso, los criterios corresponderían a los umbrales de tratamiento de la PI, y la evaluación es sencilla, consistiría en comprobar si se superan los umbrales. Lo que no se recoge en esta plantilla de tarea es que estos umbrales se seleccionan en función de un patrón de comportamiento y del caso concreto, que vienen definidos por la visita.

Una cuestión importante para el modelado de las inferencias, que no aparece recogido en ninguna de las tres plantillas es que el proceso de proponer y descartar un objetivo terapéutico se repite para todos los agentes nocivos de un cultivo que son muestreados.

El resultado del modelado del conocimiento de inferencias para las tareas *proponer y descartar objetivos terapéuticos* se muestra en la figura B 6. En primer lugar, se fijan los parámetros según el caso, al modo de la plantilla valoración. Estos parámetros se comparan con unos umbrales, obtenidos como se propone en la plantilla monitorización, y se fija un objetivo terapéutico cuando se detecta una discrepancia. Cada posible objetivo terapéutico puede ser descartado posteriormente en función de otro conjunto de datos a valorar o descubrimientos que se seleccionan dependiendo de cada objetivo terapéutico concreto, al modo de la plantilla diagnóstico, y se generan las recomendaciones y el estado del cultivo.

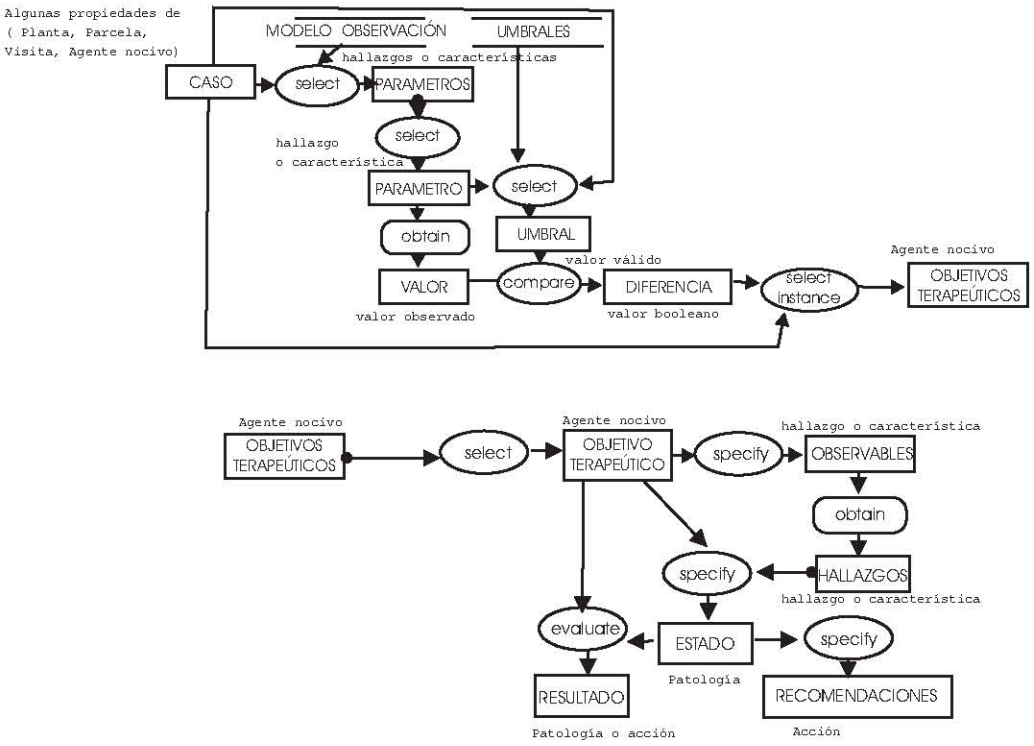


Figura B.6 Conocimiento de inferencias de la tarea seleccionar objetivos terapéuticos.

El proceso de razonamiento modelado en el conocimiento de inferencias, también queda reflejado en el código CML de los métodos de resolución seleccionados para cada tarea, como se muestra en la tabla B.4. En la figura B.6, también se refleja la correspondencia entre los elementos del dominio y los conceptos descritos en las inferencias. Utilizando esta correspondencia podemos describir el proceso de razonamiento general modelado en los diagramas de

inferencias, pero ya instanciado para nuestro dominio concreto: durante las visitas, el técnico selecciona un conjunto aleatorio de plantas, detecta la población de agentes perjudiciales, y registra estos y otros datos sobre el estado de las plagas, de las plantas y del contexto que pueda afectarle. El experto determina si es necesario realizar un tratamiento, basándose en estos datos y los umbrales de tratamiento. Finalmente, para comprender mejor el proceso de razonamiento modelado, volvemos a describir el mismo mediante un ejemplo que refleja una visita concreta del técnico agrícola, cuyo conocimiento hemos modelado: *Tenemos un cultivo de tomate. Durante la visita se ha detectado el virus TCLV (provocado por la mosca blanca) en un 20% de las plantas muestreadas. La norma indica que hay que tratar la mosca a la primera presencia si la planta tiene menos de 6 ramilletes cuajados. La mosca blanca sería un objetivo terapéutico pero si añadimos el dato de que esa planta está recién trasplantada, el virus procede del semillero, se descarta la hipótesis de tratamiento y habría que reclamar al semillero.*

Tabla B.4 Métodos aplicados en las tareas.

TASK-METHOD **monitorización-basada-en-umbrales**
REALIZES: proponer-objetivos-terapeúticos
CONTROL STRUCTURE:
select (caso -> parametros);
FOR EACH parametro IN parametros
select (parametro -> umbral)
obtain(parametro-> valor);
compare (umbral + valor -> diferencia);
IF diferencia = true
THEN select-instance(caso + diferencia -> problema-elemental)
objetivos-terapeúticos:= objetivos-terapeúticos ADD problema-elemental;
END IF
END FOR EACH
END TASK-METHOD monitorización-basada-en umbrales
	TASK-METHOD **valoración-nuevos-hallazgos**
	REALIZES: descartar-objetivos-terapeúticos
	CONTROL STRUCTURE:
	specify(-objetivo-terapeútico ->observables)
	FOR EACH observable IN observables DO
	obtain(observables ->hallazgos)
	END FOR EACH
	specify (objetivos-terapeúticos +hallazgos -> estado)
	specify (estado - > recomendacion)
	evaluate(estado + objetivos-terapeúticos -> resultado)
	END TASK-METHOD valoración-nuevos-hallazgos

C.1 Herramienta InSCo Requisite

La implantación de procesos de desarrollo como InSCo tiene sentido cuando se desarrollan proyectos de tamaño mediano o grande. Estos proyectos de sistemas software híbridos necesitan manejar y compartir mucha información para conseguir un producto válido y de calidad. Los agentes de estos proyectos son los ingenieros del software y del conocimiento, los clientes, los probadores o cualquier posible usuario del sistema.

Para obtener el éxito en el proyecto es necesario mantener esta información, utilizando herramientas software que ayuden en la elaboración de los artefactos generados por la metodología utilizada, y sobre todo herramientas para mantener la integridad frente a cambios y la administración de varios proyectos de desarrollo a la vez y que ayuden en la ejecución de todas las tareas de InSCo. En el mercado existen herramientas CASE que facilitan todas estas tareas y que están disponibles para muchas de las metodologías de desarrollo actuales y para las distintas fases o actividades de cada una de ellas, así como para las tareas de gestión del proyecto software.

En el desarrollo de la propuesta InSCo y en el marco del proyecto de investigación "Metodología para el análisis y diseño de sistemas híbridos: Integración de técnicas de modelado de software y de conocimiento" (TIN2004-05694) se ha especificado y construido un prototipo de lo que serían una suite de herramientas para la implantación del modelo de proceso InSCo(Cañadas, Orellana, Águila, Palma, & Tunez, 2009). Se han desarrollado tres herramientas software. Estas herramientas son interoperables y soportan InSCo. La figura C.1 muestra las tres herramientas.

InSCo Requisite permite la especificación de los aspectos conceptuales (Orellana, Cañadas, Águila, & Túnez, 2008) y se utilice como entrada a Rule-Modeler, donde se especifican las partes conceptuales basadas en conocimiento aplicando la notación CML. Finalmente, estos modelos conceptuales son al entrada a InSCo Gen que aplica MDA para trasladar estos modelos conceptuales a modelos computacionales y que ha sido desarrollada por otros miembros del equipo de investigación en el ámbito del proyecto (Cañadas, Orellana, Águila, Palma, & Tunez, 2009).

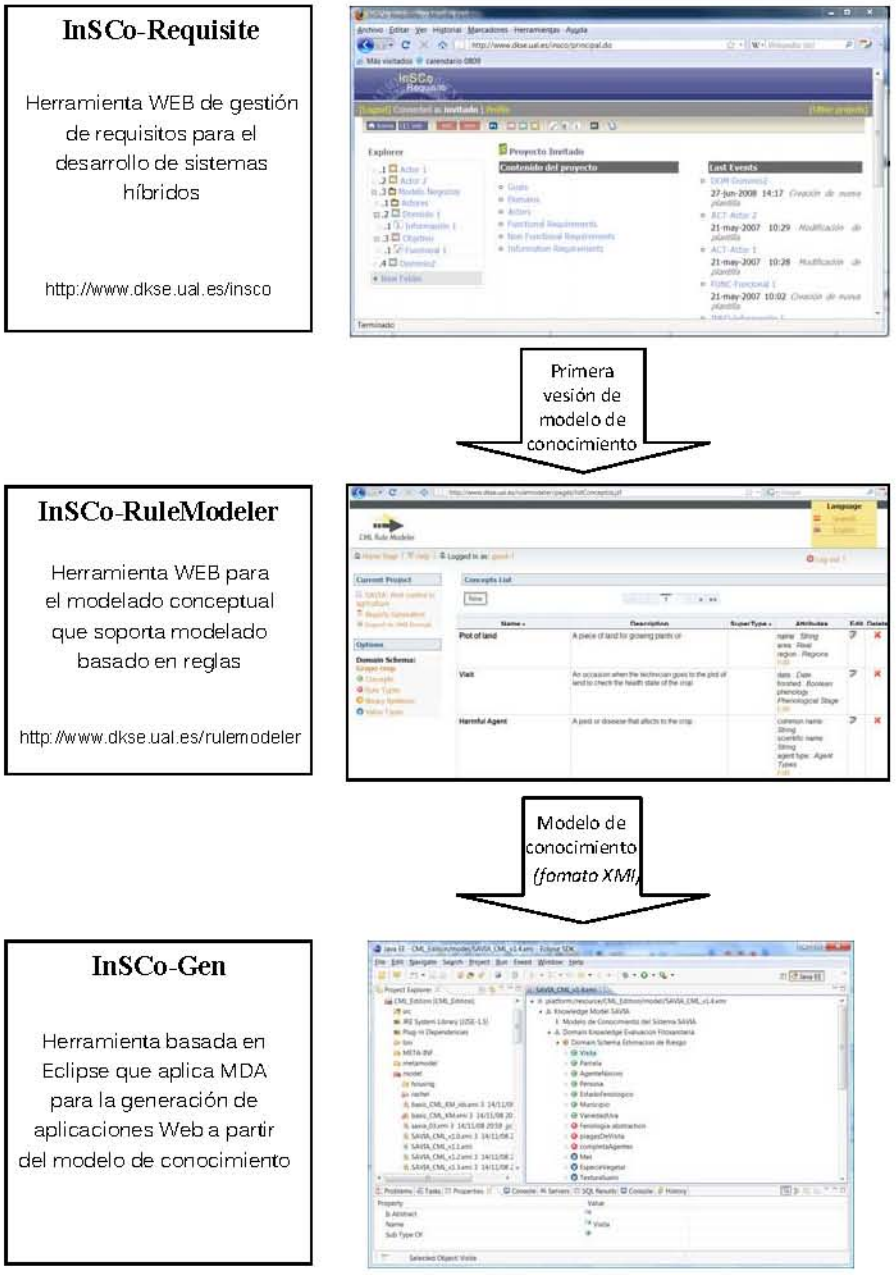


Figura C.1 InSCo Suite.

InSCo Requisite es una herramienta Web que sustenta las etapas iniciales del proceso InSCo, permitiendo llegar hasta la definición de los servicios de un sistemas software hibrido. El objetivo es proporcionar un entorno accesible para buscar el éxito en el proyecto de desarrollo del software.

La herramienta ofrece una gestión completa de los servicios funcionales de información y no funcionales. Se ha establecido de una estructura jerárquica que los mantiene organizados. Los servicios pueden ser también pre-clasificados como intensivos en conocimiento o no conocimiento intensivo.

Esta herramienta no pretende competir con otras herramientas comerciales, como Rational Requisite, DOORS o IRqA. Todas ellos son grandes desarrollos de equipos de gran tamaño. InSCo Requisite es una herramienta sencilla y de fácil uso, que incorpora algunas características deseables toda herramienta de gestión de requisitos debe tener: Gestionar un número de proyectos ilimitado, soportar la escalabilidad de los proyectos, administrar usuarios, grupos de usuarios y derechos definidos para cada proyecto, identificar cada objeto del proyecto a lo largo de su vida, permitir formatear de texto e incluir objetos que no son de texto, el control de cambios, la trazabilidad a través de enlaces entre los requisitos, la interoperabilidad con otras herramientas para continuar con el proceso de desarrollo y la generación de documentos.

InSCo Requisite ofrecen una serie de funcionalidades para interoperar con otras herramientas a fin de continuar el proceso de desarrollo, exportando el modelo de negocio (NEG) y de servicios (SER) al formato XMI (XML Metadata Interchange).

La gestión se realiza a través de formularios HTML, las plantillas y enlaces que permiten la navegación de objetivo o servicio a otro. La mayoría de las plantillas permiten la inclusión de diagramas que representan las entidades que se describen en los niveles inferiores, y archivos externos que ayudan a los agentes que tengan una mejor comprensión del dominio del problema. Proporciona algunas facilidades para la colaboración informal permitiendo debates en torno los servicios. Los agentes interesados pueden crear discusiones proponer cambios o sugerir cualquier mejora.

Esta herramienta ha sido desarrollada con la plataforma JavaEE. Con el fin de mejorar la robustez, la reutilización de código y la organización de la herramienta, se ha aplicado un patrón MVC mediante Struts, un código abierto y entorno multiplataforma creado por Apache Software Fundations. La persistencia de datos se soporta sobre Oracle. La aplicación se ejecuta en un

servidor Apache Tomcat, que apoya serverlet y JSP. La herramienta está disponible [17] .

InSCo RuleModeler es una herramienta basada en web, que modela en CML. Todos los componentes de modelado de un proyecto pueden ser fácilmente manejados con gran interacción del usuario. Utiliza plantillas y formularios web de manera eficiente, interfaz intuitiva y similar a InSCo Requisite. La tecnología Web proporciona acceso distribuido, y fomenta la participación del equipo de desarrollo, especialmente cuando los miembros del proyecto están geográficamente dispersos. La herramienta está disponible [18].

[17] http://www.dkse.ual.es/insco

[18] http://www.dkse.ual.es/rulemodeler