UNIVERSIDAD DE GRANADA

Departamento de Lenguajes y Sistemas Informáticos

TESIS DOCTORAL

MODELADO Y ANÁLISIS DE SISTEMAS CSCW SIGUIENDO UN ENFOQUE DE INGENIERÍA DIRIGIDA POR ONTOLOGÍAS

Presentada por:

Manuel Noguera García

Dirigida por:

José Luis Garrido Bullejos

María V. Hurtado Torres

Granada, Abril de 2009

Editor: Editorial de la Universidad de Granada
Autor: Manuel Noguera García
D.L.: GR 2289-2009
ISBN: 978-84-692-3088-6

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Preface

MOTIVATIONS AND PROBLEM STATEMENT

Collaboration is a human activity that crosscuts most of our daily-work routines. Nowadays, technology plays an important role in collaborative tasks. In particular, thanks to advances in Information and Communication Technologies, especially those related to Internet, collaborative tasks have transcended physical barriers and work environments, and thus, extended to other ambits such as entertainment, competition, robotics, etc.

One major concern of the Computer-supported Cooperative Work (CSCW) research discipline is to study how to build computer-based systems that help people engaged in workgroups to work effectively.

Additionally, the steadily increasing needs in scalability, performance, distribution, and interoperability of collaborative systems of today requires new capabilities that raise the level of abstraction and assist the specification of Systems and Software Engineering (SSE) processes for this kind of systems.

Model-driven Engineering (MDE) approaches to SSE have emerged in the last years arguing for the benefits of the extensive use of different types of models, i.e., high abstraction (rather declarative than imperative) models to capture business logic, and low level models to capture particular implementation specificities. High level models are refined and transformed into platform specific models in a downstream manner [Schmidt 2006]. High-level models also enable participatory design of different stakeholders in system development. This characteristic is crucial when it comes to developing CSCW systems.

The most prominent example of the MDE paradigm is the Model-driven Architecture (MDA) approach, endorsed by the Object Management Group (OMG) [OMG 2003], which distinguishes between three kinds of models

depending on their degree of dependence to specific technological platforms, i.e., Computation Independent Models (CIM), Platform Independent Models (PIM) and Platform Specific Models (PSM).

MDA highly rests on the *Unified Modeling Languaje* (UML), a de-facto standard in the SSE field [OMG 2007]. However, this formalism still lacks of a formal model-theoretic semantics that allow automated consistency checks between models to be carried out. This feature is important in order to enable systems integration and/or implementation across different technological platforms claimed by the MDA philosophy. For example, on the basis of a clearly-defined semantics, a reasoner may infer if two models are compatible by comparing the definitions of the concepts they use and how they interrelate with each other [OMG 2007c].

Moreover, despite the amount of technologies supporting the MDA initiative [OMG 2007b], there exists a considerable gap between real-world models –conceptual models, mainly–, that describe social interactions and collaboration between organizations and practitioners, and system (both hardware and software) models [Garrido 2003]. Therefore, this gap primarily exists between the so-called CIM's and PIM's. In fact, most of current model-based developments and the MDA specification itself have focused on the translation of PIM's into PSM's (see above). Although, MIC's are described in the MDA specification, nothing is said about the higher abstraction level mappings between CIM's and PIM's.

In this context, ontologies ("*a specification of a conceptualization*" [Gruber 1995]) are proving to be a suitable technology to capture the relevant aspects of a domain and detect inconsistencies in models, possibly unnoticed by analysts.

However, usually ontological descriptions are not user-friendly, and therefore, suitable as a communication vehicle with stakeholders. In this sense, they can be leveraged as a complementary technology to MDE/MDA approaches based on UML by defining appropriate mappings between concepts and relationships in both types of models, i.e., ontological models and UML-based models [Hurtado 2002].

Furthermore, ontologies make possible to define formal underlying CIM's of a system model and PIM's for multi-platform technologies. These CIM's might be subsequently transformed into the PIM's defined, and thus, enable the same business logic to be implemented across different platforms. The clearly-defined

semantics of both ontology-based CIM's and PIM's would allow consistency and compatibility checks between them to be performed. As for collaboration between organizations, this fact may be a valuable add-on to set up CSCW systems across heterogeneous partners around the world.

The extensive use of ontologies as primary models to SSE leads to the perspective of Ontology-Driven Engineering (ODE). However, ontologies present several drawbacks:

- They are mainly targeted to describe the structural (rather than behavioural) knowledge about a system, i.e., concepts and relationships between them.
- Their formal underlying semantics limits the use and flexibility in the use of expressive constructs of other conceptual modelling languages.

This is an important drawback of ontology languages when describing CSCW systems, since the description of collaboration processes is an essential component in its modelling that sometimes requires complex constructs. Therefore, further mechanisms are to be provided in order to allow collaborative processes to be described by means of ontologies. The first step to adopt an ODE approach in the development of a CSCW system consists in defining the CIM's of the system that capture both their structure and behaviour.

GOALS

The aim of this thesis is to set the foundations to the adoption of an ODE approach to the modelling and analysis of CIM's for CSCW systems. Our work is based on an existing methodology to analyse CSCW systems called AMENITIES [Garrido 2003]. This research aim is made effective through the achievement of the following goals:

- Analyse the state of the art in conceptual modelling and ontology specification languages with the pros and cons of different formalisms to represent domain knowledge.

- Specify an ontology with the set of classes and relationships of the conceptual framework proposed in the AMENITIES methodology intended to:
    - Check the consistency and soundness of such a conceptual framework.
    - Check the consistency and validate eventual changes on it.

  This ontology will constitute a first reference CIM that guides the description of subsequent ontological CIM's for specific CSCW systems.
- Define a set of extra classes and relationships so that behavioural models of AMENITIES can also be represented in the aforementioned conceptual framework ontology.
- Provide a set of ontology design patterns intended to represent common conceptual modelling constructs and/or avoid some limitations in its use, in a systematic fashion.
- Illustrate the benefits of the ontologies proposed by means of automated reasoning to detect possible inconsistencies or infer knowledge not explicitly declared.
- Asses the proposed techniques on several real case studies: the design of an interaction observation system and a case-based reasoner
- Begin the development of a tool to enable the visual edition of ontologies and assist analysts in the adoption of an ODE to ontology construction.

STRUCTURE OF THIS THESIS

The thesis is structured as follows:

- Chapter 1 provides an overview of conceptual modelling in general and its connections to systems and software development.
- Chapter 2 outlines the aims and scope of the CSCW discipline and summarizes the AMENITIES methodology.

- Chapter 3 addresses several issued related to ontologies: what an ontology is, types of ontologies, ontology languages, underlying semantics and the application of ontology-related technologies to conceptual modelling and system development.
- Chapter 4 presents a systematic approach to the development of ontologies for collaborative systems based on the AMENITIES methodology.
- Chapter 5 explains some reasoning and inference examples on the basis of the ontological descriptions proposed.
- Chapter 6 presents some applications of the proposal.
- Conclusions and ongoing work are presented in chapter 7.

Acknowledgements

This research has been supported by the Spanish Ministry of Education and Science (MEC) under its R&D programs TIN2004-08000-C03-02 and TIN2008-05995/TSI.

CHAPTER REFERENCES

[Garrido 2003] Garrido, J.L.: "AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas". Tesis Doctoral, Granada 2003

[Gruber 1995] Gruber, T.R.: "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928

[Hurtado 2002] Hurtado, M.V.: "Un Modelo de Integración Evolutivo entre Sistemas de Información y Sistemas de Ayuda a la Decisión". Tesis Doctoral, Granada 2002

[OMG 2003] Object Management Group: Model Driven Architecture (MDA) Guide, v1.0.1, OMG, omg/03-06-01

[OMG 2007] OMG: "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[OMG 2007b] OMG: MDA: "Committed Companies and Their Products" http://www.omg.org/mda/committed-products.htm

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09. Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Schmidt 2006] Schmidt, D.C.: "Guest Editor's Introduction: Model-Driven Engineering". IEEE Computer, 39 (2006), pp. 25-31

Capítulo I

MODELADO CONCEPTUAL DE SISTEMAS DE INFORMACIÓN

1. INTRODUCCIÓN

El desarrollo de sistemas de cierta envergadura requiere la participación activa y conjunta de equipos humanos llamados a entenderse por el bien y el éxito del producto final. El punto de partida es una adecuada comprensión y descripción del dominio en la fase de análisis del sistema. En este proceso, son muchas las cuestiones a considerar: recursos, requisitos (funcionales y no funcionales), grupos, usuarios, etc. La sofisticación de la propia realidad o la distancia que pueda haber con los desarrolladores del sistema, entre otros factores, hacen necesaria la construcción de **modelos** que, mediante procesos de abstracción apropiados, sirvan para comunicar el mundo físico y social en el que se desenvuelve una organización [Schreiber 1994]. La Figura I.1 muestra un modelo para un sistema de control de equipajes en un aeropuerto. En este modelo se han representado distintos tipos de cintas de transporte para maletas, el flujo de desplazamiento que siguen éstas, dispositivos de escaneo y también el lugar desde serían controladas por los operadores del sistema.

En la historia de la informática ha habido diferentes propuestas de modelado que han ido mejorándose sucesivamente. A principios de los años 70 aparecieron los "modelos de datos", concebidos como una forma de describir la estructura de una base de datos abstrayendo al usuario de detalles de implementación. Estos modelos de datos "clásicos", como el *modelo relacional* [Codd 1970], contaban con una base matemática bien definida para definir relaciones, sus propiedades y algunas operaciones sobre ellas (composición –*join*–, proyección, etc.), pero ofrecían una pobre semántica acerca de cómo interpretar los modelos, y por tanto, se prestaban a ofrecer resultados desconectados del mundo real [Schmid 1975].

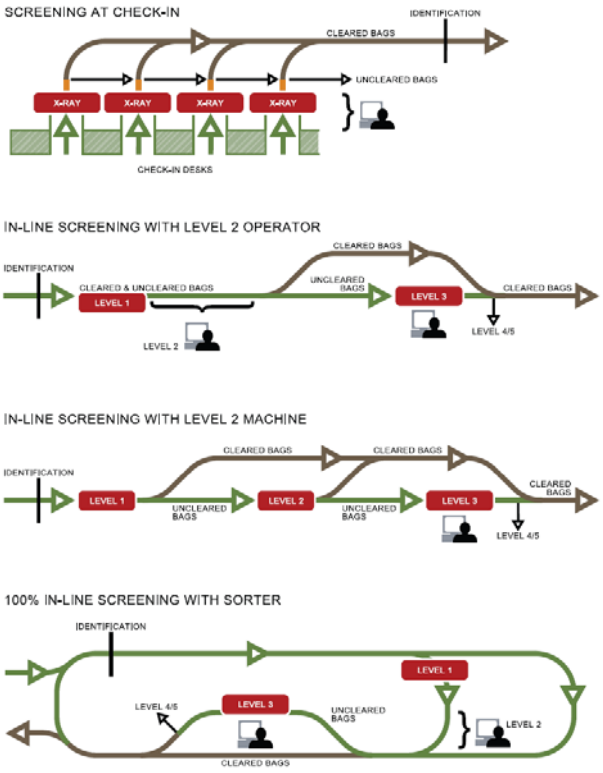


Figura I.1 Modelo de un sistema de control de equipajes (fuente: Van der Lande® Industries)

Esta carencia motivó que posteriormente aparecieran los *modelos de datos semánticos* como el *modelo entidad-relación* [Chen 1976], que combinaban el modelo relacional con redes semánticas, al mismo tiempo que proporcionaba simples técnicas diagramáticas para la representación del conocimiento.

No obstante, esta clase de modelos y las técnicas asociadas se limitaban a estructurar datos en una base de conocimiento y dotarlos de una cierta semántica. La creciente complejidad de los sistemas en general, y particularmente de su software de soporte, hizo coincidir a investigadores procedentes de las áreas de modelado de datos y los lenguajes de programación, en la búsqueda de abstracciones para especificar programas y formas de representación del conocimiento que dotaran a los modelos de más capacidad expresiva que la de las redes semánticas. Con esta intención comenzó a denominarse *modelado conceptual* (*conceptual modelling*), a la actividad que perseguía esta perspectiva más ampliada de modelado y sobre la que se asientan los desarrollos de sistemas modernos. Hoy día, el proceso de modelado conceptual se aplica a diversos ámbitos que van desde procesos de desarrollo software, al modelado del entorno de una organización, así como al modelado de una porción del mundo real con el simple objetivo de facilitar la comunicación y el entendimiento entre personas [Mylopoulos 1992].

1.1 Modelado conceptual

Una definición aceptada del proceso de *modelado conceptual* concibe éste como *"la actividad de describir formalmente algunos aspectos del mundo físico y social a nuestro alrededor, para propósitos de comunicación y entendimiento"* [Mylopoulos 1992], aunque otros autores lo describen más sucintamente como la creación de *"un modelo abstracto de la empresa"* [Bubenko 1980].

Mediante el modelado conceptual se recogen aquellos aspectos relevantes que, bajo un punto de vista determinado, mejor describen una parte de la realidad como, por ejemplo, un aeropuerto, un programa software o una sucursal bancaria. Los modelos obtenidos pueden servir de base sobre la que explicar esa realidad a un nuevo actor en el sistema. En los ejemplos anteriores podrían ser un técnico del sistema de control de equipaje, un desarrollador o un nuevo miembro de la entidad bancaria, respectivamente.

1.2 Modelos y esquemas conceptuales

Por lo que se refiere al producto resultante de la actividad de *modelado conceptual*, existen ciertas diferencias entre lo que los distintos autores definen como *modelo conceptual* y *esquema conceptual*, o incluso, utilizados a veces como sinónimos [Olivé 2007]. Ello puede requerir cierta precisión acerca de lo que aquí se va a significar con el término *modelo conceptual*. Por esta razón, introducimos esta sección para puntualizar algunas diferencias entre distintas concepciones de este término y aclarar la opción por la que nos hemos decantado en este trabajo. No obstante, ambos puntos de vista no son incompatibles y cada uno puede incluir al otro, como veremos a continuación.

Algunos de los autores más notables sobre modelado conceptual llaman *esquema conceptual* a las descripciones utilizadas para modelar un dominio, en analogía con la correspondencia que existe entre los *modelos de datos* y los *esquemas de bases de datos* en el modelo relacional [Mylopoulos 1992][Rolland 1992]. Asimismo, para estos autores un *modelo conceptual* hace referencia a la notación formal, concebida ésta como un conjunto de *"conceptos y mecanismos de abstracción"*, empleada para especificar los esquemas conceptuales [Rolland 1992]. Desde esta perspectiva, el modelo Entidad Relación [Chen 1976] para bases datos, la metodología KAOS (*Knowledge Acquisition in autOmated Specification*), para la adquisición y estructuración de modelos de requisitos [Dardenne 1993], o UML (*Unified Modelling Language*), ideado principalmente para el desarrollo de software [OMG 2007], definen cada uno un modelo conceptual para representar e interpretar una parte de la realidad. Así, un modelo conceptual se concibe como *"una forma de observar dominios"* [Olivé 2007]. En la Figura I.2 aparece representado parcialmente lo que, bajo este punto de vista, sería el modelo conceptual de UML para el modelado de acciones utilizando, a su vez, UML para representarlo.

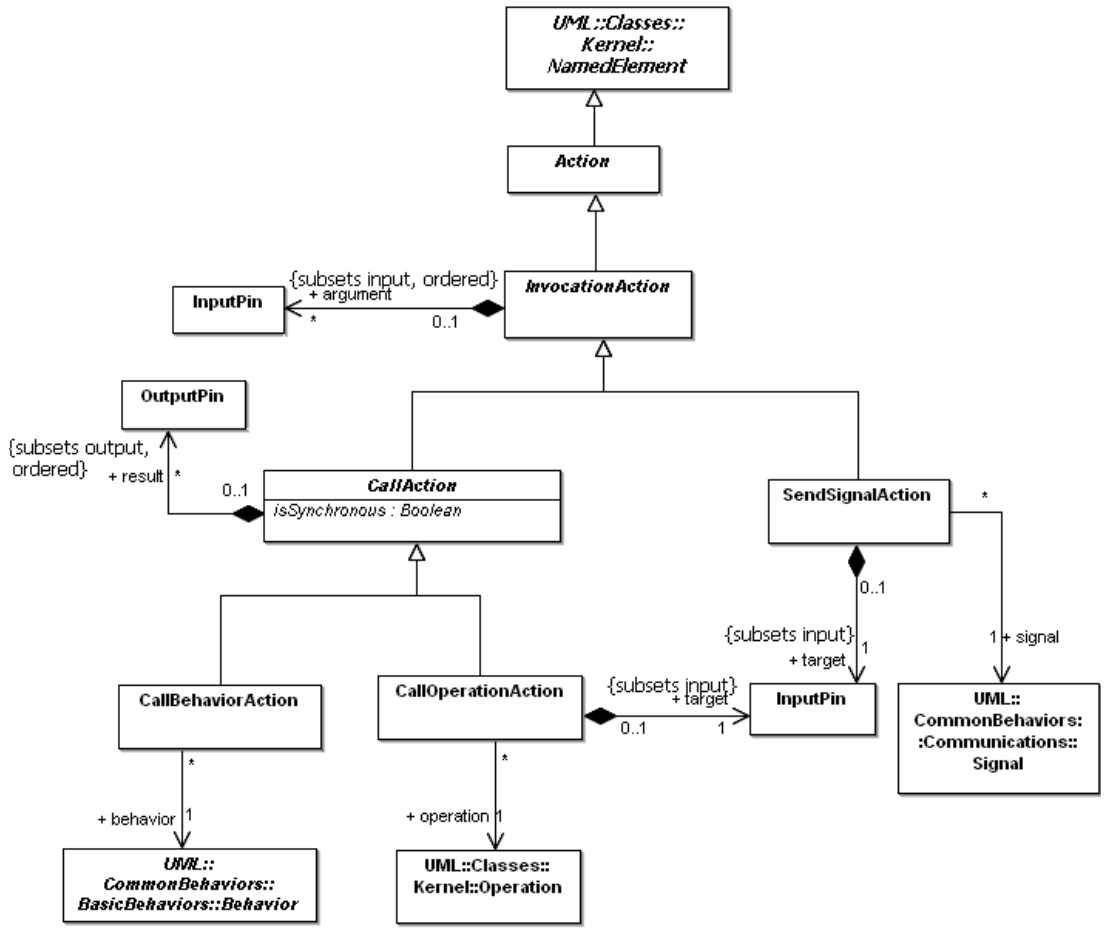


Figura I.2 Fragmento en UML representando el modelo conceptual de UML para acciones

Otros autores destacados, con un enfoque más aplicado al desarrollo de sistemas de información concretos, definen modelo conceptual como "una descripción abstracta de un entorno organizacional (parte de ella es el domino representado y parte es el entorno de uso)" [Wand 1995]. El *entorno de uso* hace referencia al entorno en el que se utiliza el sistema de información a desarrollar. Bajo este punto de vista, un modelo conceptual recoge los objetos y conceptos que son relevantes en un dominio concreto [Boman 1997]. En este caso, un esquema conceptual podría ser más bien un diagrama de clases de UML que, junto al correspondiente diagrama de objetos (también llamada base de información), configuraría un modelo conceptual para un dominio concreto [Bergholtz 2000], y a veces, el propio esquema conceptual es ya considerado un modelo conceptual [Boman 1997].

El primer punto de vista acerca de lo que es un modelo conceptual se corresponde con lo que los segundos autores podrían llamar *meta-modelo conceptual* o simplemente *metamodelo*, y que, según el segundo punto de vista, consistiría en un modelo conceptual o *lenguaje* para expresar otros modelos conceptuales.

Por otra parte, el segundo punto de vista se acerca también a una división clásica entre niveles de modelado que distingue entre tres niveles principales: meta-nivel, nivel de dominios y nivel de instancias (ver Figura I.3) [Letier 2001].

La Tabla I.1 muestra las correspondencias entre los dos puntos de vista y esta división entre niveles de modelado.

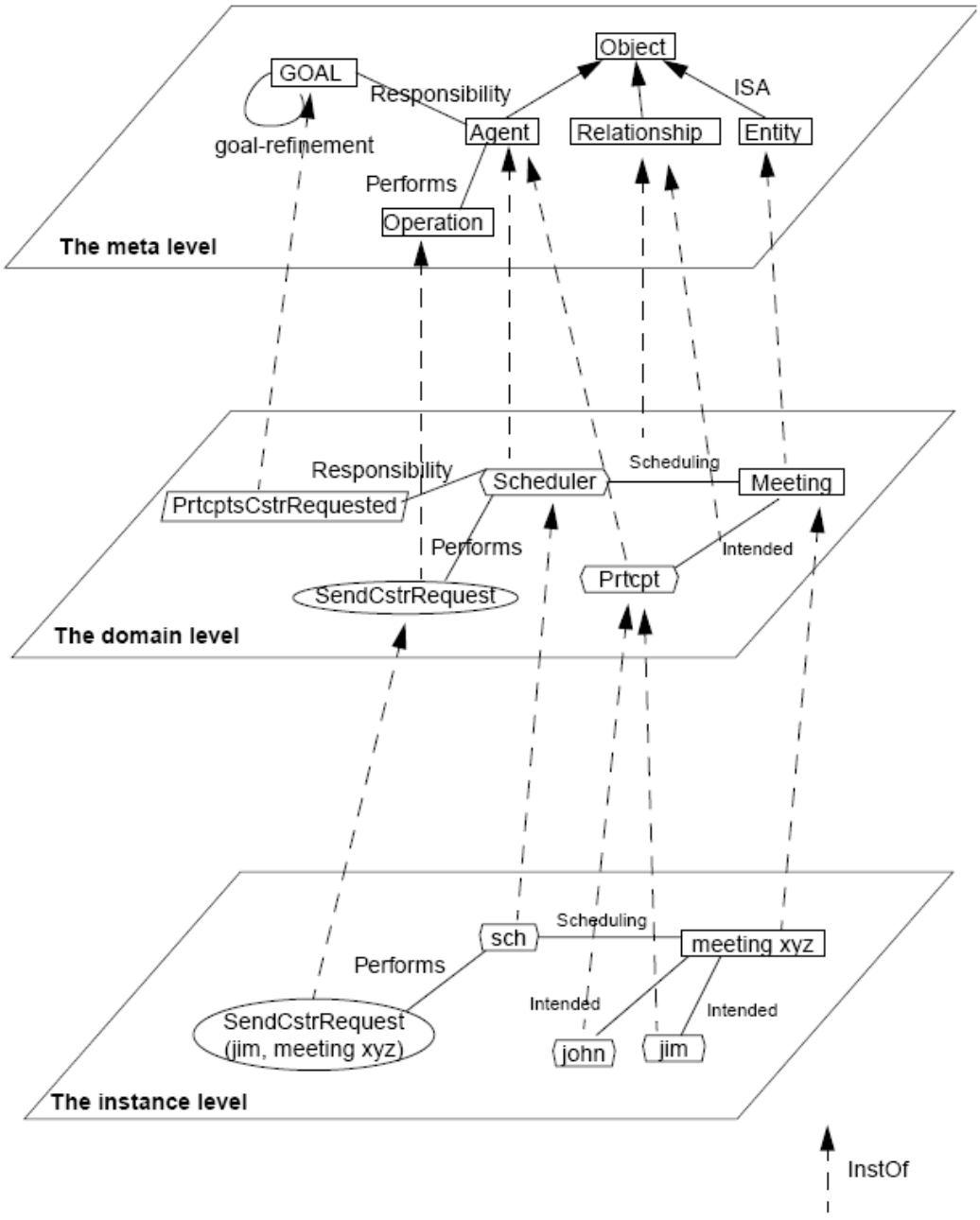


Figura I.3 Niveles de modelado (ejemplo para la metodología KAOS)

Tabla I.1 Correspondencia entre distintos puntos de vista sobre modelado conceptual y niveles de modelado

<table>
<tr><th>Punto de vista 1
(Mylopoulos, Rolland)</th><th colspan="2">Punto de vista 2
(Wand, Moody, Boman)</th><th>Niveles de modelado</th></tr>
<tr><td>Modelo conceptual</td><td colspan="2">Metamodelo conceptual</td><td>Metamodelo</td></tr>
<tr><td rowspan="2">Esquema conceptual</td><td rowspan="2">Modelo conceptual</td><td>Esquema conceptual
(Clases, relaciones)</td><td>Dominio</td></tr>
<tr><td>Base de información
(instancias)</td><td>Instancia</td></tr>
</table>

Esta visión es más habitual en el modelado conceptual de sistemas enfocado al desarrollo total o parcial de dichos sistemas, aunque la finalidad última de un modelo conceptual no es necesariamente el desarrollo de un sistema [Fiadeiro 1992]. Este trabajo también pretende adoptar este enfoque más "práctico" de la actividad de modelado conceptual. Por esta razón, en adelante, con *modelo conceptual* nos referiremos aquí a aquéllos modelos que se utilicen para describir un entorno concreto utilizando términos de la vida real (p.e. cliente, solicita, préstamo, etc.), y no tanto, a los conceptos utilizados para modelar esas entidades del mundo real (p.e. clase, asociación, subclase, etc.) [Moody 2005]. Finalmente, tampoco haremos continuamente una clara distinción entre un modelo conceptual y su representación (o esquema conceptual), ya que, no supone un aspecto primordial para la lectura de este texto y en la bibliografía suelen utilizarse indistintamente [Olivé 2007][Boman 1997].

1.3 Representación de modelos conceptuales

El modelado conceptual de un sistema tiene por objeto fomentar la comunicación y el mutuo entendimiento entre personas acerca de una porción de la realidad. Asimismo, los modelos se utilizan para reflejar el consenso entre las partes implicadas, acerca de cómo entienden y pretenden que funcione en el futuro un sistema en su entorno. En consecuencia, se puede afirmar que los modelos conceptuales están diseñados para ser usados por personas y no por computadoras. Por esta razón, para construir los modelos se tiende a hacer uso de descripciones gráficas y textuales que, aun basadas en la especificación precisa y formal –o semiformal— de algún lenguaje de modelado, tienen como notas características la simplicidad y la eficiencia [Mylopoulos 1992].

Algunos ejemplos de distintas formas de representar modelos conceptuales son un diagrama de clases de UML, una especificación de una clase en Telos [Mylopoulos 1990] o un modelo para representar objetivos en KAOS. La adecuación de las distintas

representaciones se medirá en función de su capacidad para facilitar el común entendimiento entre personas acerca de la realidad modelada.

La Figura I.4 muestra un modelo gráfico en KAOS que representa objetivos propuestos para el servicio de ambulancias de la ciudad de Londres [Letier 2004]. Asimismo, la Figura I.5 muestra la descripción en Telos de una publicación científica enviada a un congreso y su representación mediante una red semántica. Estos ejemplos son meramente ilustrativos para mostrar otras alternativas al omnipresente UML.

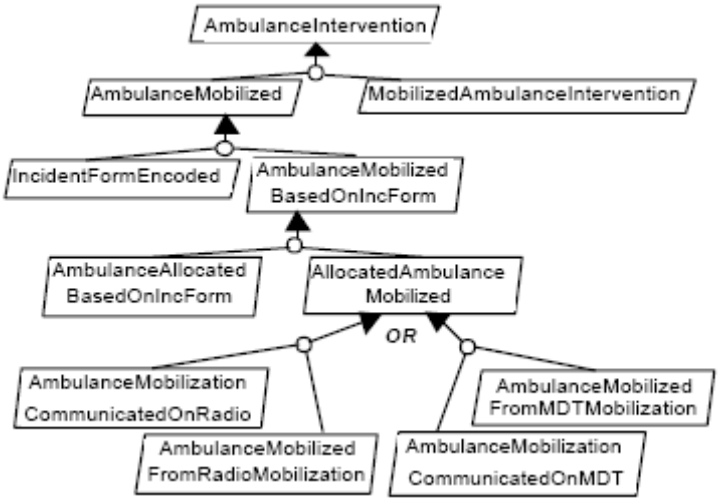


Figura I.4 KAOS: modelo de objetivos para el servicio de ambulancias de Londres

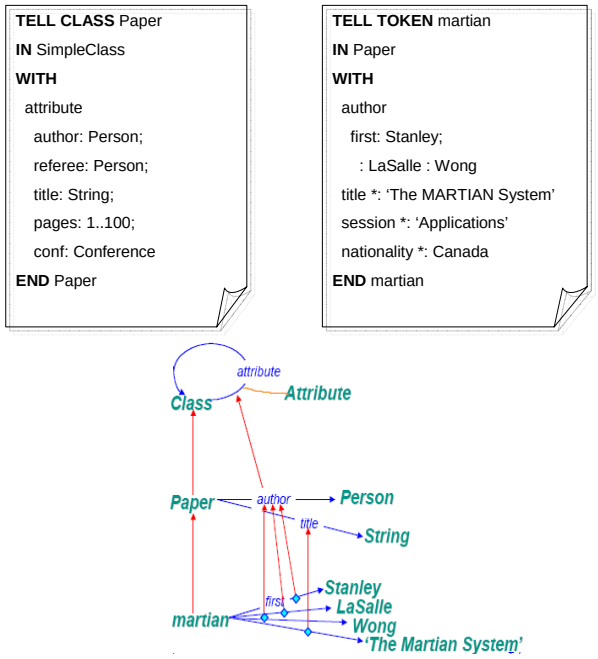


Figura I.5 Telos: modelo conceptual de un artículo para una conferencia

2. SISTEMAS DE INFORMACIÓN Y ORGANIZACIONES

El ámbito de aplicación más extendido de la actividad de modelado conceptual es el diseño y desarrollo de sistemas de información. En este trabajo, vamos a considerar que un sistema de información –SI—, es un sistema basado en computadoras que ayuda a la gestión y uso de la información en una organización o entre varias organizaciones [Boman 1997].

Hoy día, los sistemas de información están presentes en la mayor parte de los entornos donde tiene lugar la actividad humana. Algunos ejemplos de SI podemos encontrarlos en un sistema de reserva de plazas de hotel, un sistema para operar en bolsa con acciones o un sistema para a recaudación de impuestos de un país.

En muchas ocasiones, el SI también mantiene información acerca de la propia empresa, que puede ayudar a gestionar y supervisar el funcionamiento de una organización en relación a sus fines [Boman 1997]. Quizá por esta razón algunos autores identifican el sistema de información con la propia organización que hace uso de él, es decir, abarcando a las personas y usuarios finales con las que interactúa, artefactos, productos que crea, etc. [Liu 2001]. Nosotros preferimos utilizar el término *sistema empresarial* u *organizacional*, para denominar a éste sistema más amplio que incluye personas físicas, artefactos, procesos de negocio, políticas de empresa, etc., y también, uno o varios sistemas de información.

2.1 Colaboración y SI

Otra característica importante de los SI es que están diseñados para ser utilizados por múltiples agentes (humanos o computacionales). Como consecuencia, inmediatamente aparece la necesidad de que estos agentes puedan comunicarse y coordinarse para colaborar de forma efectiva [Ellis 1991]. Esto es, estamos ante sistemas intrínsecamente *sociales*. En un sistema de reservas de avión, una persona puede ser la encargada de introducir nuevas plazas para un vuelo, mientras que otra se encarga de atender al cliente y hacer las oportunas reservas, propiciando que ambas colaboren. Esto hace que los SI sean elementos clave en torno a los que se articulan numerosos procesos colaborativos del sistema organizacional en el que funcionan. Más aún, en los últimos años es habitual que los SI cuenten con algún tipo de herramienta (generalmente software) de apoyo al trabajo colaborativo, que mejore la comunicación y coordinación de sus usuarios. Por todo lo anterior, los SI también han dado en llamarse *Sistemas de Información Cooperativos*.

Aunque en este capítulo se van a tratar aspectos del análisis y diseño de SI con carácter general, el desarrollo posterior de este trabajo se va a centrar en el modelado conceptual de entornos colaborativos, donde el SI de soporte a la colaboración se diseña para ser compartido por una o entre varias organizaciones. En particular, el modelado de sistemas de información se va a enfocar en los siguientes aspectos:

- La estructura de la organización o conjunto de organizaciones.
- El modelado de tareas colaborativas que deben llevarse a cabo.
- La interrelación y coordinación a nivel tecnológico de los distintos componentes del SI para dar soporte a la coordinación humana.

El objetivo es mejorar la interconexión del SI con la lógica de negocio de la empresa [Peppard 2001], y que el SI permita monitorizar y razonar sobre la estructura y forma de organizar el trabajo colaborativo en una empresa.

2.2 Modelado conceptual en el desarrollo de SI

Mediante el modelado conceptual de un determinado entorno (un aeropuerto, una sucursal bancaria, un departamento de la universidad), tratamos de describir una parte de la realidad con el propósito de poder comunicarla y consensuar cómo es o cómo desearíamos que fuera. De esta forma, el modelado conceptual ayuda al proceso de desarrollo del SI de una organización.

El SI resultante permite a sus usuarios gestionar conocimiento de diversa índole acerca de un dominio particular. Asimismo, el modelado conceptual puede describir la forma de organizar el trabajo en una empresa y acceder al SI. Por tanto, el SI también puede gestionar información acerca del dominio en que trabajan sus propios usuarios. Con independencia de que ambos dominios puedan solaparse, o de la información que puedan gestionar ciertos usuarios del SI, algunos autores coinciden en la conveniencia de hacer una distinción acerca de los distintos tipos de conocimiento que desde un punto de vista cognitivo y organizacional (o social), están relacionados con el desarrollo de un SI.

De esta forma, una extendida caracterización distingue entre el conocimiento acerca de 4 *mundos* principales involucrados en el desarrollo de un SI [Jarke 1994]. Cada mundo tiene asociado su propio grupo de *stakeholders*, cuyas necesidades han de integrarse de cara al subsiguiente desarrollo del SI. Estos mundos son (ver Figura I.6):

- El *mundo de la empresa*. Es el *dominio en cuestión*, esto es, el dominio particular acerca del cual el SI mantiene una representación. Por ejemplo, en una sucursal bancaria, el mundo de la empresa se compone de clientes, cuentas corrientes, tipos de interés, transacciones, trabajadores, etc.
- El *mundo de uso*, esto es, el entorno donde se utiliza el SI. Su modelado conceptual comprende la descripción de los agentes, tareas, usuarios, interfaces de usuario para operar con el SI, etc. Continuando con el ejemplo de la sucursal bancaria, dentro de este mundo entrarían el software de gestión de la sucursal, los roles de que pueden desempeñar los empleados al utilizar dicho software (cajero, director, responsable de riesgos, etc.), los impresos para realizar ciertas transacciones,…

- El *mundo del sistema* y que se corresponde con el propio SI. Contiene una representación del dominio (o *mundo de la empresa*) y del modo en que los usuarios del SI accederán a esta información. Esta representación puede ser orientada a objetos, orientada a procesos, orientada a datos o de algún otro tipo. Asimismo, mantiene las correspondencias que conectan el modelado conceptual de un sistema (principalmente mundo de la empresa, pero también el mundo de uso) con su diseño e implementación (*mundo del desarrollo*).
- El *mundo del desarrollo*, donde se describen y planifican procesos, recursos disponibles, equipo de analistas y programadores, que conducirán al desarrollo del SI final.

Figura I.6 Mundos representados en el modelado de SI

La actividad de modelado conceptual para el desarrollo de un SI también puede analizarse en el contexto de la clásica división del proceso de desarrollo de sistemas computacionales que distingue entre las fases de *análisis*, *diseño* e *implementación*. En efecto, durante la fase de análisis, una percepción del mundo real del sistema se abstrae en un modelo conceptual de dicho sistema. Este modelo conceptual comprende la creación de modelos para el mundo de la empresa y el mundo de uso. En la fase de diseño, los modelos conceptuales de la etapa anterior se transforman en un modelo del SI, particularmente se diseñan las interfaces de usuario con el SI a partir de las descripciones del *mundo de uso*. Por último, en la fase de implementación, el modelo del SI se traduce a un modelo de implementación ejecutable, por ejemplo, en términos de clases software (ver Figura I.7).

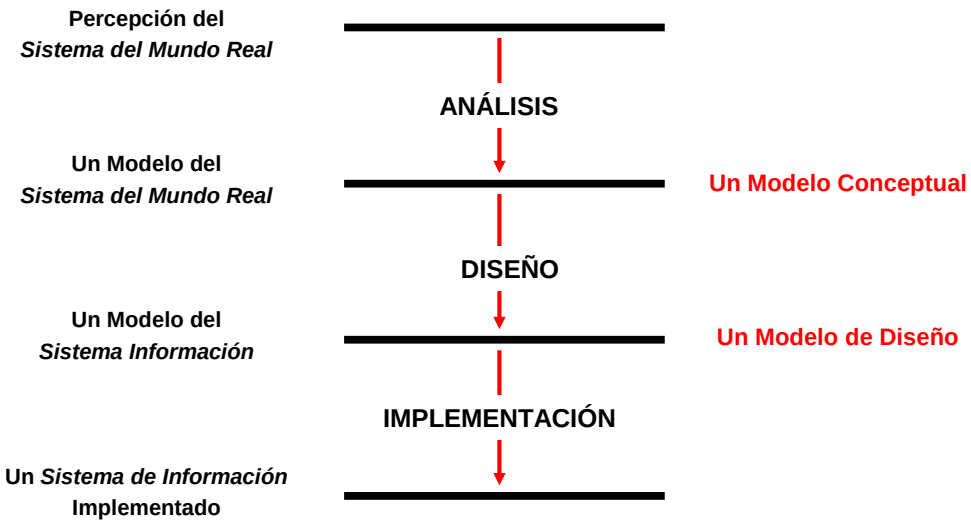


Figura I.7 Modelado conceptual en el desarrollo de un SI

Del análisis anterior se desprende que van a existir fuertes interdependencias entre el conocimiento que concierne a cada uno de los 4 mundos expuestos. En la mayoría de los casos, la integración del SI en la organización o la incorporación de nuevas funcionalidades por parte de éste, cambian la lógica de negocio en la misma y el modelo conceptual que la representa. Al fin y al cabo, la implementación de un SI hace que éste pase de contener una representación de la organización a formar parte de ella. Por ejemplo, la digitalización de ciertos documentos y su inclusión en el SI de una biblioteca, cambia el modo en que sus usuarios acceden a ellos, y también puede llevar a cambiar su localización física por otra donde estén mejor conservados.

Estas relaciones de dependencia son más notables cuando se aborda el desarrollo de un sistema colaborativo (entendido éste como un conjunto estructurado de actores, tareas, artefactos, reglas de negocio, etc.). En este caso el *mundo de uso* y el *mundo del sistema* son en esencia el dominio en cuestión (o *mundo de la empresa*). En efecto, el correcto funcionamiento del *sistema colaborativo* (*mundo de la empresa*) se fundamenta en poder dotar a sus actores (agentes software, personas), de acceso a recursos de información por medio de interfaces adecuadas. Comparando lo que se ha dicho hasta ahora sobre un sistema colaborativo con la definición de *mundo de uso*, pueden verse las coincidencias entre ambos, sobre todo en lo que se refiere a actores e interfaces de usuario. Por otro lado, el entorno colaborativo (de nuevo, *mundo de la empresa* o dominio en cuestión), se modela definiendo la forma compartir, consultar, modificar, recursos de información almacenados en algún SI (*mundo del sistema*), y por tanto, el SI es considerado a priori un elemento del dominio. Así, un proceso de negocio del sistema puede estar organizado, por ejemplo, en torno a la compartición y modificación concurrente de un documento por dos personas; el documento está almacenado en el SI del sistema colaborativo que lo usa; la finalización del documento y su validación dan origen a otras actividades colaborativas, etc. Como puede

verse, en sistemas colaborativos, el *mundo del sistema* es un componente fundamental del *mundo de la empresa*.

En esta sección se han presentado los tipos de conocimiento que mantiene un SI. Este enfoque es útil de cara al desarrollo, ya que considera, además del dominio en cuestión, los ámbitos en el los que se opera con el SI, es decir, el dominio de uso y el dominio de implementación y será el que tengamos presente en el desarrollo posterior de este trabajo. Como ya hemos mencionado, estamos especialmente interesados en enfoques de modelado orientados al diseño e implementación de sistemas computacionales.

Sin embargo, desde el punto de vista de los sistemas organizacionales, donde generalmente el SI es un componente desarrollado a posteriori para ser insertado en la arquitectura de la empresa, el énfasis se hace en la representación que el SI mantiene acerca de un dominio particular (mundo de la empresa). Para ello se manejan principalmente conceptos relativos a actores, procesos (tareas), metas y objetos que conciernen a ese dominio. Estos conceptos dan lugar a diferentes perspectivas de modelado conceptual.

En la siguiente sección vamos a tratar de describir estas perspectivas desde las que se puede analizar y modelar el conocimiento que mantiene un SI acerca de una organización. Asimismo, se describen las interrelaciones que existen entre ellas en el contexto de la lógica de negocio de una organización.

2.3 Conceptos y vistas fundamentales en el modelado conceptual de SI

Como acabamos de ver, un SI mantiene una representación acerca de los diferentes activos (personas, artefactos, procesos, etc.) con que pueda contar una organización. La complejidad de las relaciones entre ellos, hace que a la hora de su diseño, se empleen distintas vistas o perspectivas de modelado recogidas en submodelos del modelo general del sistema [Bajec 2005]. Estas vistas tienden a centrarse en diferentes aspectos de la empresa (actores, procesos, metas, objetos, etc.), y también hacen referencia a la información descrita desde otras perspectivas, fomentando la navegabilidad entre ellas (ver Figura I.8) [Bubenko 2001]. Ejemplos de modelos presentando perspectivas diferentes de la empresa pueden ser los siguientes:

- Un *modelo de actores* recoge las personas, los roles a desempeñar en una empresa y las tareas que tienen encomendadas.
- Un *modelo de tareas* puede describir el desarrollo colaborativo de distintas actividades o procesos que tienen lugar en la empresa.
- Un *modelo de metas* u objetivos se centra en los fines para los que se diseñan ciertos procesos o para el que crean ciertos roles en la empresa.

- Un *modelo de objetos* describe los activos de una organización y sus interdependencias: qué propiedades tienen, quiénes los usan, dónde se crean, etc.

En el capítulo II volveremos sobre este punto para examinar los modelos utilizados en esta tesis para el modelado de sistemas de información colaborativos.

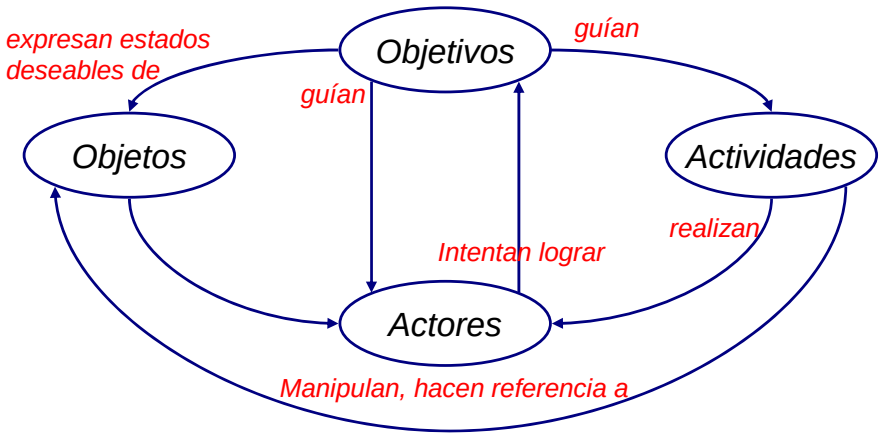


Figura I.8 Vistas de una organización [Boman 1997]

3. DEL ANÁLISIS AL DISEÑO E IMPLEMENTACIÓN DE SI

Las secciones anteriores (2.1 y 2.2) se han centrado en argumentar los beneficios que aporta la actividad de modelado conceptual al desarrollo de SI. Asimismo, se ha tratado de perfilar el objeto del modelado conceptual de un sistema: la descripción formal del dominio del sistema con el fin de fomentar la comunicación y el entendimiento entre personas. Se han enumerado distintos planos o *mundos* relacionados con el desarrollo de un SI, y se han descrito los principales conceptos que han de manejarse para describir adecuadamente un entorno organizacional.

Para la representación de los modelos conceptuales se utiliza un *lenguaje de modelado conceptual* (sección 1.2). Este lenguaje determina el conjunto de constructores (textuales, gráficos o de ambos tipos) y reglas que definen cómo combinarlos para modelar un dominio [Schreiber 1994b]. Ejemplos clásicos de constructores usados en los modelos conceptuales son aquellos que permiten declarar entidades, relaciones, actividades, procesos, etc. La semántica de los constructores condiciona la potencia expresiva de un lenguaje. Asimismo, la potencia expresiva de un lenguaje de modelado determina su capacidad para representar un dominio. Así, un lenguaje demasiado restrictivo, o limitado, en cuanto a sus constructores, puede ser inútil para reflejar conceptos importantes de la realidad que se pretende modelar [Wand 1995].

El estudio pormenorizado de la idoneidad de un lenguaje de modelado para un dominio concreto requiere un análisis que forma parte del campo de la *semiótica* (estudio de la

comunicación entre personas o animales mediante símbolos) y la filosofía del lenguaje, que están más allá del ámbito de esta tesis. No obstante, puesto que los modelos conceptuales son creados para fomentar la comunicación y entre personas, parece claro que la semántica de los lenguajes de modelado ha de estar fundamentada en modelos del conocimiento humano. Por otro lado, los modelos conceptuales han de "materializarse" en modelos ejecutables, por lo general modelos software, con vistas al diseño e implementación de SI.

En lo que sigue introduciremos brevemente los fundamentos que justifican el uso de modelos basados en ontologías para el análisis y diseño de un SI, así como su relación con los modelos orientados a objetos, preferentemente usados en el desarrollo de software. En los capítulos posteriores se profundizará la descripción de estas tecnologías y su aplicación concreta al desarrollo de sistemas de información colaborativos.

3.1 Desarrollo de SI utilizando ontologías

El término *ontología* se utiliza en el área de los *Sistemas de Información Dirigidos por Ontologías*[1] y la Web Semántica, para designar las especificaciones formales acerca de las entidades que existen en un dominio concreto y cómo se relacionan entre ellas [Guarino 1998].

Durante el desarrollo de un SI, los diferentes *stakeholders* hacen uso de modelos conceptuales para comunicar su visión de una parte de la realidad a partir de cómo la perciben. En este sentido, de acuerdo con su definición, las ontologías son un medio útil para representar un dominio [Wand 1995]. En los Capítulos III y IV profundizaremos en el estudio de estos formalismos y su aplicación al caso concreto del desarrollo de sistemas de información colaborativos.

Por otro lado, en el ámbito del diseño y modelado de software existen otras técnicas y lenguajes centrados en la implementación del SI para su ejecución por algún sistema computacional. En estos lenguajes predominan los enfoques orientados a objetos. No obstante, estos lenguajes, como UML [OMG 2007], también son utilizados en el modelado conceptual y en la descripción de la arquitectura de una empresa [Ambler 2003].

A continuación vamos a tratar de analizar las implicaciones que tiene el uso de modelos basados en UML para el desarrollo (análisis, diseño e implementación) de un SI. Tras presentar algunas objeciones que presenta este lenguaje, presentaremos nuestra propuesta acerca del uso de ontologías para completar y mejorar los procesos de desarrollo que hacen uso de modelos especificados en UML.

---

[1] En inglés Ontology-driven Information Systems (ODIS).

3.2 Uso de UML en el desarrollo de SI

Sin duda, en el campo de desarrollo de software, UML se ha convertido en un estándar de referencia, debido en gran medida a que los metaconceptos que utiliza para representar dominios, como *clase* o *atributo*, tienen muchas veces una correspondencia directa con los conceptos que manejan los lenguajes de programación modernos [Lange 2006].

La utilización de UML para el modelado conceptual tiene, además, la ventaja de dotar al desarrollador con un mismo lenguaje y enfoque de modelado (en particular, orientado a objetos) durante las diferentes etapas del desarrollo. De esta forma se facilita la transición del análisis al diseño del sistema, y de la arquitectura de la empresa a la arquitectura del software. Por otra parte, los modelos conceptuales de dominio compartidos entre analistas y usuarios, también pueden servir a los primeros para comunicarse con los diseñadores de software [Evermann 2005b].

Por contra, hay que tener presente que en la fase de análisis del SI, un objeto o instancia de un modelo conceptual representa una entidad del mundo real, mientras que en las fases de diseño e implementación, un objeto de un modelo de software representa una entidad ejecutable que existe sólo dentro de la máquina [Wand 1995]. En consecuencia, la semántica de los constructores usados en los modelos como "atributo" u "objeto" es diferente. En este sentido, no son infrecuentes las confusiones de este tipo que los modelos de dominio en UML causan entre analistas y desarrolladores de software.

Como ejemplo de lo anterior es posible encontrar que, como parte de la arquitectura de la organización, el analista del sistema ha diseñado que los objetos de la clase *Actor* llevan a cabo objetos de la clase *Tarea*, sin embargo, en el ámbito de la lógica de negocio de una empresa, la concepción que existe acerca de las tareas a realizar no encaja con la visión de éstas como "objetos" [Evermann 2005b].

En el lado del desarrollo de software, este tipo de confusiones en ocasiones también lleva a representar en un mismo modelo elementos de la arquitectura de la empresa y componentes de la arquitectura del software. Así, podemos pensar en el caso de un software para la gestión de un hospital. Al modelar la forma en que opera la aplicación, el desarrollador puede especificar mediante un diagrama de clases que los objetos de la clase *Médico* pueden *actualizar*, por medio de diversos métodos, los datos recogidos en objetos de la clase *Historial_de_Paciente* (ver Figura I.9). Sin embargo, mientras que en el SI correspondiente, los (objetos) *historiales de pacientes* pueden tener una representación computacional ejecutable, un *médico* sólo existe en el dominio del sistema.

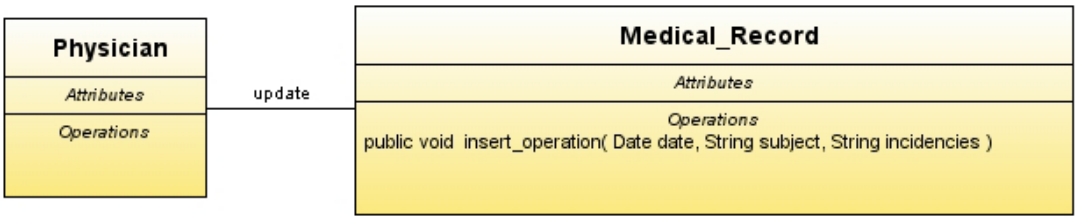


Figura I.9 Diagrama de clases usado en desarrollo de software

Por otra parte, UML carece de una semántica rigurosa –sobre todo en lo que se refiere al tratamiento de la semántica de conjuntos y una teoría de modelos—, que permita utilizar razonadores automáticos sobre los modelos representados con este lenguaje. Asimismo, UML se muestra en muchos casos insuficiente para expresar conceptos sobre pertenencia a conjuntos o es demasiado restrictivo, como la disjunción o el conjunto complementario [OMG 2007c]. La semántica de conjuntos es importante porque es la base sobre la que se construye la jerarquía de clases de un modelo o metamodelo. Los problemas relativos a la semántica de conjuntos pueden abordarse mediante el lenguaje de definición de restricciones OCL (Object Constraint Language) [OMG 2006], pero de nuevo nos encontramos con la falta de una teoría de modelos ("álgebra universal + lógica" [Chang 1990]), y una teoría formal de pruebas para llevar a cabo procesos de inferencia automáticos [OMG 2007c].

Este último aspecto reviste especial importancia de cara a un proceso de desarrollo dirigido por modelos. En efecto, cada vez es más frecuente que dos o más organizaciones traten de integrar su estructura organizativa y sus procesos de negocio para conseguir determinados fines. Un razonador podría detectar la incompatibilidad entre los modelos donde apareciera recogida esta información, siempre y cuando existiera una semántica y un conjunto de axiomas para la relacionar los conceptos manejados en los distintos modelos. Esto redundaría en una mejora de la colaboración a nivel tecnológico entre sistema, y por ende, a nivel humano, ambos focos centrales de este trabajo.

Como consecuencia de lo expuesto anteriormente, se hace necesario realizar una serie de esfuerzos que vayan encaminados a:

- La definición de una semántica bien definida donde se asienten los fundamentos teóricos que respalden el uso de ciertos conceptos usados para representar modelos del software, como "clase" u "objeto", en el modelado conceptual de una organización.
- El establecimiento de las correspondencias adecuadas entre los modelos usados en el análisis para el modelado conceptual de un sistema y los modelos de diseño e implementación (modelos de desarrollo software principalmente).

- Obtener representaciones de los distintos modelos en lenguajes con una semántica formal bien definida que permita llevar a cabo tareas de razonamiento e inferencia automáticos.

El primero de estos puntos ha sido ya abordado recientemente en otros trabajos [Evermann 2003][Wand 1995], en los que se propone el uso de ontologías para describir la semántica de los constructores usados en los modelos, esto es, "clase", "objeto", "atributo", etc. Siguiendo este mismo enfoque, otros trabajos [Green 2007], se han centrado en el análisis de la semántica de lenguajes para el modelado de procesos de negocio como BPML (Business Process Modeling Language) y BPEL4WS (Business Process Execution Language for Web Services), entre otros.

Nuestros esfuerzos se van a centrar en la consecución de los otros dos puntos por medio de lenguajes formales para la descripción de ontologías. El objetivo es tratar de aprovechar las ventajas de ambas tecnologías (lenguajes para describir ontologías y UML), y combinándolas en el proceso de desarrollo.

REFERENCIAS DEL CAPÍTULO

[Ambler 2003] Ambler, S. W., Nalbone, J., Vizdos, M.: "Enterprise Unified Process: Extending the Rational Unified Process". Prentice Hall PTR, 2003

[Bajec 2005] Bajec, M., Krisper, M.: "A methodology and tool support for managing business rules in organisations", Information Systems, Volume 30, Issue 6, September 2005, 423-443

[Bergholtz 2000] Bergholtz, M., Johannesson, P.: "Validating Conceptual Models - Utilising Analysis Patterns as an Instrument for Explanation Generation", NLDB 2000, LNCS, vol. 1959, 2001, pp. 325-339

[Boman 1997] Boman, M., Bubenko, J.A., Johannesson, P., Wangler, B.: Conceptual Modelling, Prentice Hall Series in Computer Science, 1997

[Bubenko 1980] Bubenko, J.A.: "Information Modelling in de Context of Systems Development", Information Processing 1980, proceedings of IFIP Congress, North Holland, 395-411

[Bubenko 2001] Bubenko, J.A., Persson, A., Stirna, J.: "D3 Appendix B: EKD User Guide", Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden, 2001

[Chang 1990] Chang, C. C., Keisler, H. J.: "Model Theory, Studies in Logic and the Foundations of Mathematics", 3rd ed., 1990, Elsevier, ISBN 978-0-444-88054-3

[Chen 1976] Chen, P.: "The Entity-Relationship Model – Toward a Unified View of Data", ACM Transactions on Database Systems 1(1), 9-36, 1976

[Codd 1970] Codd, E.F.: "A relational model of data for large shared data banks", Commun. ACM, vol. 13, no. 6, June 1970, 377-387

[Dardenne 1993] Dardenne, A., van Lamsweerde, A., Fickas, S.: "Goal-directed requirements acquisition", (1993) Science of Computer Programming, 20 (1-2), pp. 3-50

[Ellis 1991] Ellis, C.A., Gibbs, S.J., Rein, G.L.: "Groupware: Some Issues and Experiences", Communications of the ACM, Vol. 34, No. 1 (January 1991) 38-58

[Evermann 2003] Evermann, J.: "Using Design Languages for Conceptual Modelling: The Uml Case". PhD thesis, University of British Columbia, Canada, 2003

[Evermann 2005b] Evermann, J., Wand, Y.: "Ontology based object-oriented domain modelling: fundamental concepts", Requirements Engineering, 10 (2), 2005, 146-160

[Fiadeiro 1992] Fiadeiro, J., Sernadas, C., Maibaum, T., Sernadas, A.: "Describing and Structuring Objects for Conceptual Schema Developments", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 117-138

[Green 2007] Green, P., Rosemann, M., Indulska, M., Manning, C.: "Candidate interoperability standards: An ontological overlap analysis", Data & Knowledge Engineering, Volume 62, Issue 2, August 2007, pp. 274-291

[Guarino 1998] Guarino, N.: "Formal Ontology in Information systems". Proceedings of FOIS '98, (Trento, Italy, June, 1998). IOS Press, Amsterdam, 1998, 3–15

[Hurtado 2002] Hurtado, M.V.: "Un Modelo de Integración Evolutivo entre Sistemas de Información y Sistemas de Ayuda a la Decisión". Tesis Doctoral, Granada 2002

[Jarke 1994] Jarke, M., Pohl, K.: "Establishing visions in context: Towards a model of requirements processes". EMISA Forum 4 (2), pp. 49-62

[Lange 2006] Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: "UML Software Architecture and Design Description". IEEE Software, 23 (2006), 40-46

[Letier 2001] Letier, E.: "Reasoning about Agents in Goal-Oriented Requirements Engineering". PhD thesis, Louvain-la-Neuve, Belgium, 2001

[Letier 2004] Letier, E. and van Lamsweerde, A.: "Reasoning about partial goal satisfaction for requirements and design engineering". In Proceedings of the 12th ACM SIGSOFT Twelfth international Symposium on Foundations of Software Engineering (Newport Beach, CA, USA, October 31 - November 06, 2004). ACM, New York, NY, 53-62

[Liu 2001] Liu, K., Sun, L., Dix, A., Narasipuram, M.: "Norm-based agency for designing collaborative information systems". Information Systems Journal 11 (3), 2001, 229-247

[Mylopoulos 1990] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M.: "Telos: representing knowledge about information systems". ACM Trans. Inf. Syst. 8, 4 (Oct. 1990), 325-362

[Mylopoulos 1992] Mylopoulos, J.: "Conceptual Modelling and Telos" in "Conceptual Modelling, Databases and CASE", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 49-68

[Moody 2005] Moody, D.L.: "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions", Data & Knowledge Engineering, Volume 55, Issue 3, Quality in conceptual modeling - Five examples of the state of art, December 2005, pp. 243-276

[Olivé 2007] Olivé, A.: "Conceptual Modeling of Information Systems", Chapters 1 and 17, Springer-Verlag, 2007

[OMG 2006] OMG, "Object Constraint Language. OMG Available Specification. Version 2.0". (OCL 2.0 2006-06-01). Latest version is available at: http://www.omg.org/docs/formal/06-05-01.pdf

[OMG 2007] OMG, "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09". Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Peppard 2001] Joe Peppard (2001) Bridging the gap between the IS organization and the rest of the business: plotting a route Information Systems Journal 11 (3), 249–270

[Rolland 1992] Rolland, C., Cauvet, C.: "Trends and perspectives in conceptual modelling", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 27-48

[Schmid 1975] Schmid, J. and Swenson, R.: "On the Semantics of the Relational Data Model", Proceedings ACM SIGMOD International Conference on Management of Data, 211-223, 1975

[Schreiber 1994] Schreiber, G., Wielinga, B., de Hoog, R., Akkermans, H., van de Velde, W.: "CommonKADS: A Comprehensive Methodology for KBS Development", IEEE Expert: Intelligent Systems and Their Applications, vol. 09, no. 6, 28-37, 1994

[Schreiber 1994b] Schreiber, G., Wielinga, B., Akkermans, H., Van de Velde, W., Anjewierden, A.: "CML: The CommonKADS Conceptual Modelling Language". A Future for Knowledge Acquisition, 8th European Knowledge Acquisition Workshop. LNCS 867, Springer-Verlag, Berlin, 1994, pp. 1-25

[Wand 1995] Wand, Y., Monarchi, D.E., Parsons, J., Woo, C.C.: "Theoretical foundations for conceptual modelling in information systems development", Decision Support Systems, Volume 15, Issue 4, December 1995, 285-304

Capítulo II

MODELADO CONCEPTUAL DE SISTEMAS COOPERATIVOS

1. INTRODUCCIÓN

Los sistemas basados en computadoras están presentes en numerosos ámbitos de nuestra vida cotidiana, desde el coche o el teléfono móvil, hasta el ordenador portátil que utilizamos para trabajar y del que nos hacemos inseparables. En parte, ello se debe a las posibilidades que brindan los avances en las Tecnologías de la Información y la Comunicación (TIC), que nos permiten acceder y compartir recursos de información localizados en SI ubicados por todo el mundo. De esta forma, estos dispositivos y su software de soporte se han convertido en elementos clave para poder desarrollar nuestro trabajo de forma deslocalizada.

Desde mediados de los años 80 y, fundamentalmente durante la década posterior, el grado de penetración de los ordenadores personales en el ámbito laboral y la mejora de las redes de comunicaciones en general, propició la aparición de nuevas formas de organizar el trabajo en grupos de personas dispersas geográficamente, pero que colaboran entre sí para sacar adelante ciertas tareas.

Por otra parte, la construcción de sistemas de trabajo en grupo es compleja, ya que, se han de considerar diversas cuestiones relativas a la distribución, concurrencia y coordinación de los distintos procesos que tienen lugar en los espacios de trabajo compartidos. También hay que considerar que el objetivo último de estos sistemas es promover la colaboración entre personas, esto es, estamos ante sistemas donde su dimensión social reviste especial importancia. La mera resolución de problemas tecnológicos relativos a la sincronización y coordinación de agentes software y tecnologías hardware en estos sistemas, no siempre ha conseguido sistemas que satisficieran las expectativas de sus usuarios, como de hecho ha quedado demostrado en numerosas

ocasiones [Grudin 1988]. En consecuencia, al margen de cuestiones tecnológicas, se hace necesario analizar el modo de funcionamiento de los grupos y su evolución [Grudin 1994].

2. CSCW Y GROUPWARE: CONCEPTOS Y DEFINICIONES

Como resultado de lo anterior, se despertó en la comunidad científica internacional un interés creciente por analizar las interacciones en el trabajo colaborativo entre personas mediando tecnología basada en computadoras. El resultado fue la aparición de la disciplina científica denominada *Computer Supported Cooperative Work* (en adelante, CSCW), dedicada al estudio de estas interacciones con vistas a la colaboración efectiva de grupos de personas y punto de encuentro entre científicos procedentes de los campos de las Ciencias Sociales (Psicología, Sociología, Psiquiatría, etc.) y de las Ciencias de la Computación [Jacovi 2006].

Por otro lado, el carácter interdisciplinar del CSCW a veces dificulta el consenso acerca del significado de los términos manejados al caracterizar los procesos colaborativos en un grupo (p. ej. *groupware*, cooperación, colaboración, etc.) [Grudin 1994], tal y como veremos a lo largo de esta sección.

Hoy día, el ámbito de estudio del CSCW se ha ampliado más allá del trabajo colaborativo, abarcando otros entornos, como el recreativo o de ocio, en los que las personas hacemos uso de la tecnología para jugar, socializarnos o incluso, competir. Esto ha llevado a cambiar el significado *"work"* (trabajo) de la "*W"* del acrónimo, por el de *"world"* (mundo).

Íntimamente ligado al ámbito del CSCW se encuentra el término *groupware*, entendido como el software y las tecnologías de computadores y comunicaciones utilizadas por un sistema colaborativo, aunque también es habitual que sólo haga referencia al software que da soporte al sistema. Así, en la bibliografía relacionada podemos encontrar, entre otras, las siguientes definiciones:

- *"El uso de tecnologías de la información para ayudar a que las personas trabajen juntas de una forma más efectiva"* [Malone 1990].
- *"El software que acentúa el entorno multiusuario coordinando cosas, así que los usuarios vean a los demás y sin crear conflictos entre ellos"* [Lynch 1990].
- *"Sistemas basados en computadoras que dan soporte a grupos de personas involucradas en una tarea (u objetivo) común proporcionando una interfaz a un entorno compartido"* [Ellis 1991].

El término *groupware* con frecuencia hace referencia a un caso especial de software que proporciona las interfaces y funcionalidades necesarias para que un grupo de personas pueda colaborar. Nosotros adoptaremos la visión del *groupware* como la combinación de

software, hardware, servicios, procesos, etc. de un sistema informático para habilitar espacios compartidos interactivos, ya que, es la más usada por los autores de referencia [Ellis 1991]. Para referirnos únicamente al *software* colaborativo de estos sistemas utilizaremos el término *aplicación groupware*.

2.1 Comunicación, coordinación, colaboración

En la intención de todo sistema *groupware* está proporcionar la funcionalidad necesaria para hacer posible las interacciones *de usuario a usuario* (en inglés, *user-to-user*), a diferencia de las interacciones *usuario-sistema* soportadas por la mayoría de los sistemas software clásicos. En efecto, tanto si se trata de la elaboración de un documento, como consultar una base de datos o una búsqueda en Internet, habitualmente sólo tienen lugar interacciones entre un usuario y un sistema software. Sin embargo, cada vez con mayor frecuencia estas y otras actividades son realizadas por grupos de personas.

Tradicionalmente, se han identificado **tres pilares** altamente relacionados entre sí y sobre los que se asienta un sistema groupware en su propósito de proporcionar la funcionalidad necesaria para trabajar en grupo: *comunicación, colaboración y coordinación* [Ellis 1991].

La *comunicación* es una actividad humana que permite el intercambio de información entre personas. Es posible identificar una serie de elementos que permiten caracterizar este proceso: participantes, información que se transmite y medio utilizado. En un sistema groupware la computadora es el artefacto que actúa como medio para la transmisión de información.

Atendiendo al momento en que se produce, se puede distinguir entre dos tipos de comunicación:

- *Síncrona*, cuando los participantes interaccionan enviando y recibiendo mensajes de información en el mismo período de tiempo (p. ej., en una comunicación por teléfono, en una videoconferencia, etc.).
- *Asíncrona*, cuando los participantes envían sus mensajes en distintos momentos de tiempo (p. ej., comunicación por correo electrónico o postal, fax, etc.).

La *colaboración* hace referencia a la posibilidad de que varias personas puedan trabajar conjuntamente en una misma tarea para la consecución de un determinado fin como, por ejemplo, la prestación de un servicio, la inicialización de un proceso o la elaboración de un documento.

Colaboración y *cooperación* son dos términos muy próximos semántica y sintácticamente, como los verbos de los que se derivan, respectivamente, *colaborar* (del latín, *collaborare*,

"trabajar con") y *cooperar* (también del latín, *cooperari*, "operar/obrar con"), donde *obrar* se define como "*Hacer algo,* trabajar *en ello*". En el ámbito del CSCW, algunos autores consideran sinónimos a ambos términos [Bannon 1989][Schlichter 1998]. Como ya hemos visto, en [Ellis 1991] se identifica la colaboración como una parte integrante de la cooperación y medio para conseguir esta última. Por otro lado, en [Dillenbourg 1995] se distingue la *cooperación* señalando que la tarea principal se divide en subtareas que realiza cada participante de forma independiente y cuyo efecto, una vez completada, se incorpora a la tarea principal. En cambio, cuando se trata de *colaboración*, cada subtarea requiere la implicación conjunta de varios participantes en las misma subtarea y un mayor grado de coordinación para completarla.

Aunque entendemos las diferencias entre ambos que existen desde determinados puntos de vista, los métodos para la especificación de sistemas groupware presentados en esta tesis están diseñados para abordar las concepciones acerca de estos dos términos, y por tanto, los usaremos indistintamente al referirnos a sistemas que hacen posible el trabajo en grupo. Lo mismo será de aplicación a términos derivados como *colaborativo/cooperativo*.

Por último, la *coordinación* es la actividad encaminada a gestionar las dependencias entre actividades realizadas en grupo para la consecución de un objetivo [Malone 1990]. La coordinación se ve como una actividad en sí misma al ser un proceso necesario cuando varias personas participan en una misma tarea. La efectividad de la comunicación y colaboración depende de la coordinación, es decir, de la organización, planificación y sincronización de las actividades del grupo. Por ejemplo, un equipo de programadores sin coordinación entrará en conflicto e incluso repetirá acciones.

La tecnología actual presta un amplio soporte al trabajo distribuido en grupo. En este sentido, la coordinación a nivel tecnológico facilita la coordinación a nivel humano [Schlichter 1998]. De esta forma, los distintos participantes pueden planificar procesos, habilitar o impedir el acceso a recursos por medio de aplicaciones software distribuidas. Asimismo, con frecuencia estas aplicaciones permiten la modificación concurrente del estado de los objetos presentes en un espacio compartido, y que dichos cambios puedan ser advertidos por participantes situados en distintas localizaciones (propiedad conocida como *conciencia de grupo* o *group awareness*). Esto requiere un alto grado de coordinación a nivel tecnológico para que el *groupware* del sistema opere correctamente. La Figura II.1 muestra las relaciones entre la coordinación a nivel tecnológico y a nivel humano.

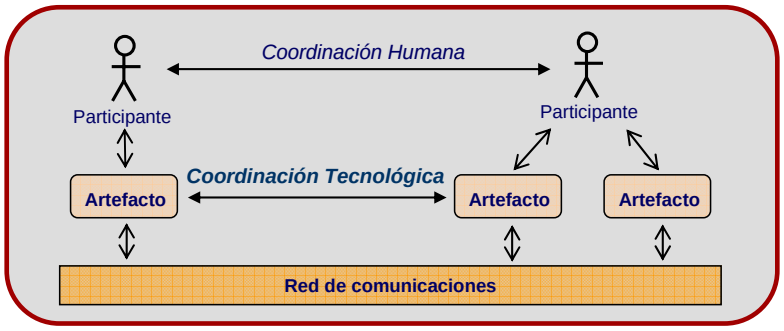


Figura II.1 Distintos niveles de coordinación [Garrido 2003] (adaptado de [Schlichter 1998])

2.2 Perspectivas de estudio en CSCW

El carácter multidisciplinar del CSCW propicia la existencia de distintas perspectivas desde las que estudiar los sistemas colaborativos y que mejoran nuestra comprensión global acerca de su funcionamiento, objeto, éxito o fracaso de implantación, etc.

En [Ellis 1991] se enumeran cinco puntos de vista principales a la hora de abordar el estudio de un sistema CSCW:

- **Sistemas distribuidos.** Habitualmente, los usuarios de un sistema CSCW aparecen distribuidos en el espacio (distinta localización física), y en el tiempo (interactúan con el sistema en distintos o en el mismo momento temporal). Esta perspectiva explora y resalta la descentralización tanto de datos como de control. El análisis de sistemas distribuidos permite obtener propiedades globales del sistema y estudiar cómo preservar consistente el estado global de una aplicación groupware observando y manipulando parámetros locales.

- **Telecomunicaciones.** Esta perspectiva hace hincapié en el intercambio de información entre agentes remotos. Esto afecta principalmente a la mejora de la conectividad y ancho de banda, y al diseño de protocolos de comunicación efectivos para intercambiar muchos tipos de información (texto, voz, vídeo, etc.). Uno de los principales desafíos para esta perspectiva se centra en hacer que las interacciones distribuidas sean tan efectivas como las que se realizan cara a cara entre personas, de tal forma que la interacción remota a través de redes de comunicaciones sea una alternativa *real* para la comunicación entre los distintos participantes.

  Aunque no llegue a reemplazar a la comunicación cara a cara, puede ser preferible en ciertas situaciones para grupos con ciertas dificultades o necesidades. Por ejemplo, al utilizar un computador para la comunicación se pueden localizar y

enviar enlaces a recursos de información a los que los distintos interlocutores pueden ir accediendo sin interrumpir el flujo de interacción.

- **Interacción persona-ordenador (IPO).** Esta perspectiva se centra en la interfaz de usuario que proporciona la computadora. A su vez, la IPO [Dix 1998] es un área multidisciplinar que integra diferentes campos de estudio como la informática, psicología y ergonomía, con la intención de estudiar y desarrollar teorías que puedan aplicarse al análisis de la interacción entre la computadora y el humano [ACM 1992]. En esta área intervienen diseñadores gráficos e industriales, expertos en informática gráfica (tecnologías de visualización, dispositivos de entrada, técnicas de interacción, estilos de diálogo, etc.) e ingeniería del software (metodologías y modelos computacionales, plataformas de desarrollo, etc.), y expertos en ciencias cognitivas (cognición humana, percepción, etc.).

  El groupware invita a los investigadores a abordar los aspectos de relacionados con la IPO en el contexto de interfaces multiusuario o de grupo. Puesto que estas interfaces son sensibles a factores como dinámicas de grupo y estructuras de la organización, es importante que sociólogos, psicólogos, usuarios finales, etc., intervengan en su desarrollo.

- **Inteligencia artificial.** Desde esta perspectiva, se facilita la creación de agentes que tienen como objetivo la mejora de las interacciones entre personas y computadoras. El principal desafío es asegurar que la forma de operar de un determinado groupware realmente mejora la interacción tal como esperan los participantes.

  Para ello, se persigue desarrollar agentes inteligentes que sean flexibles y tengan la capacidad de adaptarse a los procedimientos y formas de trabajar de grupos diferentes. Por ejemplo, dos grupos distintos pueden utilizar tecnologías diferentes para realizar la misma tarea y un mismo grupo, puede usar la tecnología de forma diferente para realizar tareas distintas.

- **Teoría social.** La teoría social o sociología ayuda al diseño de sistemas groupware analizando primero los requisitos del trabajo en grupo en un determinado entorno, y posteriormente, trasladando estos requisitos al sistema basado en computadoras que se va a desarrollar mediante las adaptaciones oportunas. Por otro lado, el análisis sociológico de una organización sirve para guiar y formar a los usuarios en el uso un sistema colaborativo para resolver las tareas que tengan encomendadas. La ausencia de este tipo de análisis ha sido la causa del fracaso de ciertas aplicaciones groupware [Grudin 1988].

2.3 Clasificación de sistemas groupware

En cuanto a la clasificación de sistemas groupware se refiere, existen numerosas propuestas. Como se viene mostrando, la diversidad de puntos de vista y campos de investigación con los que se relaciona el CSCW hace difícil que exista una única forma de describir o clasificar los sistemas groupware.

Entre todas las clasificaciones que se han dado, la más extendida es la conocida como *matriz groupware* o *CSCW*, debida a [DeSanctis 1987] y refinada posteriormente por [Johansen 1988]. Dicha clasificación establece un criterio espacio-temporal para identificar distintos tipos de sistemas groupware. La Figura II.2 muestra una representación de la matriz groupware y algunos ejemplos de distintos tipos de sistemas según los criterios propuestos.

Atendiendo al lugar donde se encuentran, las aplicaciones groupware se pueden concebir para ayudar tanto a un grupo de participantes interaccionando cara a cara como a un grupo distribuido en diferentes localizaciones.

Por otro lado, dependiendo de si el sistema permite interactuar en el mismo o en diferentes momentos de tiempo, se distingue entre sistemas síncronos o asíncronos, respectivamente.

Ejemplos de interacción cara a cara son la tecnología de una sala de reuniones, entornos de conversación y tormentas de ideas (*brain storming*); mientras que un editor en tiempo real de documentos, *chats* y videoconferencias, son ejemplos donde tiene lugar un tipo de interacción síncrona distribuida; una *pizarra* (o pantalla, panel donde visualizar información) física es un ejemplo de sistema que permite la interacción asíncrona; y sistemas de correo electrónico, agendas de grupos, listas de correo y grupos de noticias añaden la componente de distribución a la interacción asíncrona.

Un sistema groupware general podría ser de mayor utilidad si satisface las necesidades de dos o más cuadrantes de la matriz. Por ejemplo, es interesante disponer de la misma funcionalidad base e interfaz en un editor de documentos en tiempo real, tanto cuando se utiliza por un grupo (en el mismo o diferente lugar), como cuando se usa de forma aislada, en la oficina o en casa (diferente momento).

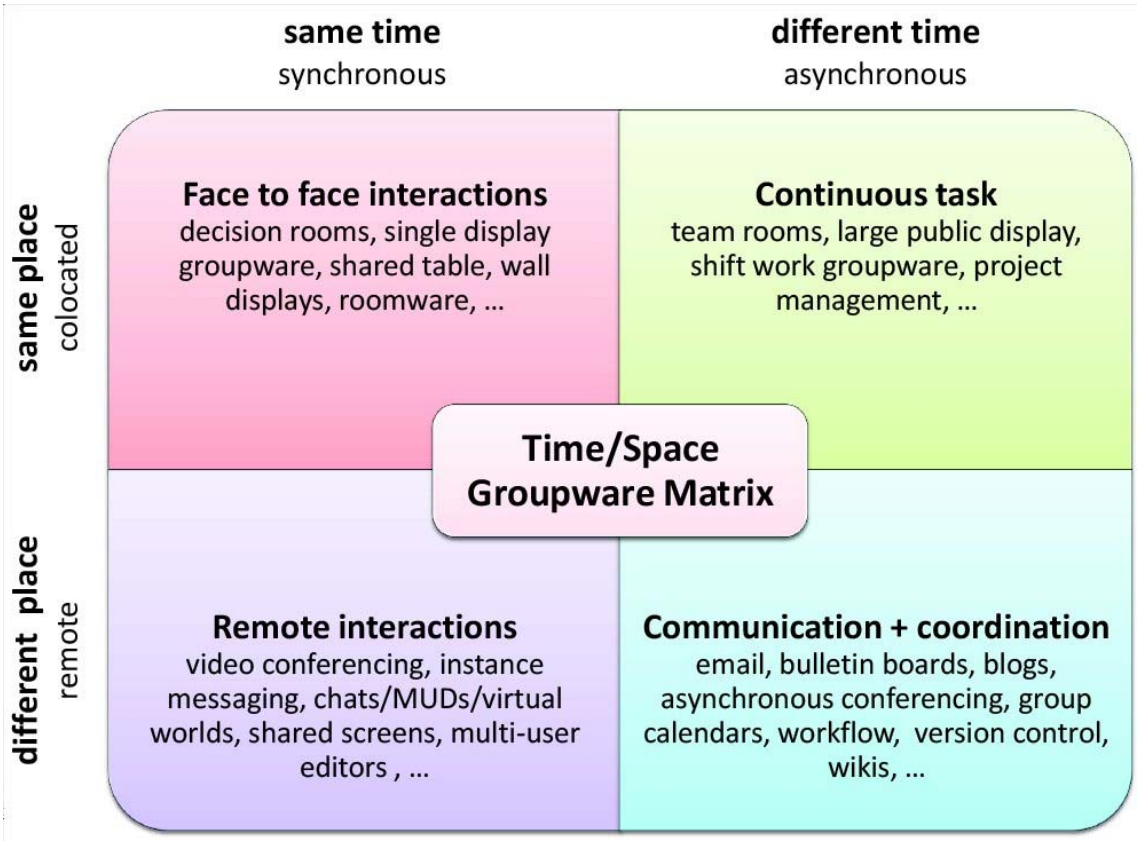


Figura II.2 Matriz Groupware (Fuente: Wikipedia®)

3. METODOLOGÍA AMENITIES

Una vez presentados las nociones y ámbitos relacionados con el mundo del CSCW, pasamos a centrarnos en la forma de analizar y diseñar sistemas groupware por medio de una metodología concreta sobre la que se ha construido el trabajo de esta tesis.

3.1 Introducción

AMENITIES (acrónimo de ***A ME****thodology for a****N****alysis and des****I****gn of coopera****TI****ve syst****EmS***) [Garrido 2003][Garrido 2002] es una metodología basada en modelos de comportamiento y de tareas que permite abordar de manera sistemática el análisis y diseño de sistemas cooperativos, favoreciendo su posterior implementación y despliegue.

En esta metodología, el modelado conceptual del sistema cooperativo se centra en el concepto de *grupo*, cubriendo aspectos relevantes de su comportamiento (dinámicas, evolución, etc.) y estructura (organización, leyes, etc.).

AMENITIES proporciona un marco de trabajo conceptual y metodológico para modelar y diseñar sistemas colaborativos. La propuesta consta de un conjunto específico de modelos y fases a seguir. En particular, aporta los siguientes elementos:

- **Fases generales** para analizar y diseñar sistemas cooperativos.
- **Fases específicas** para la construcción de los diferentes modelos implicados en la metodología.
- Un **marco conceptual de trabajo** para dar soporte al modelado de este tipo de sistemas.
- Una **notación** denominada **COMO-UML** (acrónimo de *COoperative MOdel notation based on UML*) que se basa en el lenguaje UML [OMG 2007]. La notación se utiliza para la construcción del modelo del sistema (denominado modelo cooperativo), considerado el núcleo central de la metodología.
- Una **semántica precisa** de la notación anterior en el dominio del problema (sistemas cooperativos), mediante el formalismo *Redes de Petri Coloreadas* (CPN - *Coloured Petri Nets*) [Jensen 1996].
- Una **herramienta** denominada COMO-TOOL (acrónimo de *COoperative MOdel TOOL*) con diferentes funciones de apoyo al análisis y modelado de estos sistemas.

3.2 Esquema general de la metodología

La Figura II.3 muestra el esquema general de la metodología AMENITIES. En ella se representan los principales modelos implicados y las fases generales.

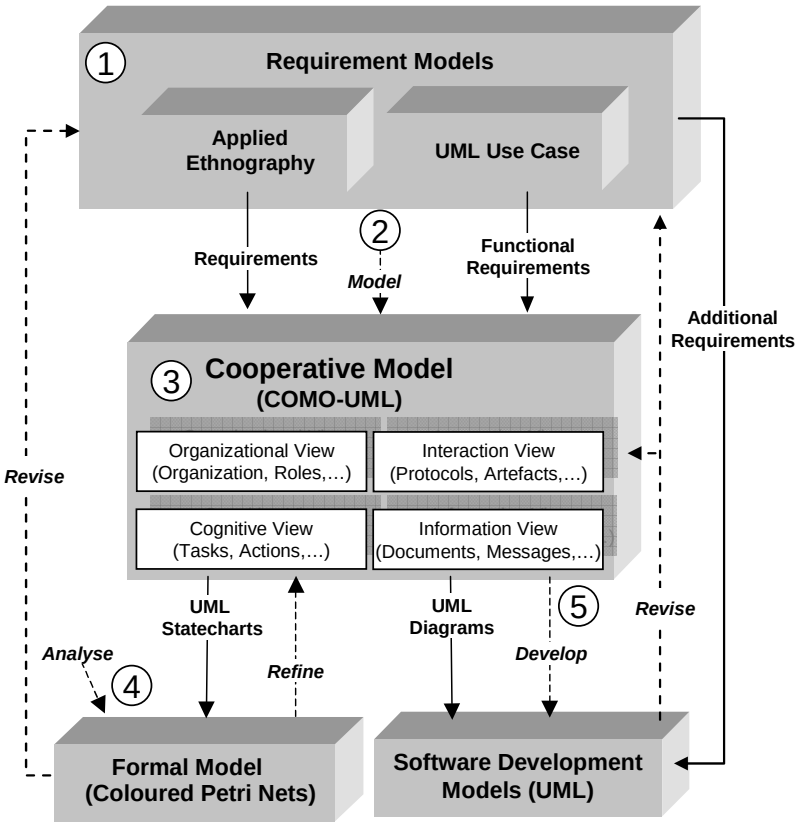


Figura II.3 Esquema general de la metodología AMENITIES [Garrido 2003]

Las fases generales de la metodología son las siguientes:

1. Análisis del sistema y obtención de requisitos.
2. Modelado del sistema cooperativo.
3. Análisis del modelo cooperativo.
4. Diseño del sistema, introduciendo nuevos elementos o cambios en el modelo cooperativo, probablemente como resultado del análisis anterior o de nuevas necesidades.
5. Implementación del sistema software.

Como la mayoría de las metodologías, la aplicación de AMENITIES al desarrollo de un sistema colaborativo sigue un proceso iterativo simple, permitiendo llevar a cabo un refinado del modelo como consecuencia del análisis de éste, así como una revisión de los requisitos de partida o del modelo cooperativo que podrían aportar nueva o diferente información a considerar.

A continuación se describen con más detalle los modelos utilizados y su propósito en el contexto de la metodología.

3.3 Modelos de requisitos

La obtención y especificación de requisitos se lleva a cabo principalmente mediante la técnica de la **etnografía aplicada** y los **diagramas de casos de uso** de UML, aunque otras técnicas bien conocidas (escenarios, *brainstorming*, etc) pueden utilizarse de forma complementaria.

La etnografía ayuda a identificar los requisitos que suelen escapar a la aplicación de otros métodos estructurados más ampliamente utilizados. Esto es aún más evidente en el caso de los sistemas cooperativos donde existe la necesidad de obtener requisitos en base a características y/o peculiaridades sociales, y culturales dentro de los grupos.

El uso de la etnografía tiene como objetivo proporcionar una mejor comprensión de las prácticas de trabajo y del comportamiento de los grupos que las llevan a cabo. Así por ejemplo, nos permite analizar:

- La **estructura de los grupos** en base a ciertos aspectos en muchos casos sutiles, pero de vital importancia (comportamiento del participante dentro y fuera del grupo, responsabilidades directas y asumidas, etc.).
- Cuestiones socioculturales que pueden afectar a la **interacción y comunicación** entre participantes, para la cual utilizan artefactos, tales como, videoconfencia por computadora, editores o espacios de trabajo compartidos, etc.
- **Comportamiento** de los participantes y las posibles influencias de este comportamiento en el resto de participantes que forman parte de un grupo.
- Requisitos funcionales relacionados con las **prácticas de trabajo** y con los **recursos** necesarios para llevar a cabo el trabajo en grupo.

Por otra parte, los casos de uso de UML [Rumbaugh 1999] permiten identificar requisitos funcionales acerca del comportamiento del sistema y, por tanto, representar de forma abstracta los objetivos de los usuarios. Un **caso de uso** es una unidad coherente de funcionalidad expresada como una transacción entre los usuarios y el sistema. Los casos de uso pueden describirse a diferentes niveles de detalle. Por lo tanto, pueden ser factorizados y descritos en términos de otros casos de uso más simples.

En AMENITIES, la aplicación de la etnografía permite reunir informalmente los requisitos del sistema a estudiar y desarrollar. A continuación, a partir de la información obtenida (esquemas, escenarios narrados, grabaciones de vídeo, etc.), los diagramas de casos de uso se utilizan para estructurar y especificar los requisitos funcionales, roles y actores en el sistema.

3.4 Modelo cooperativo

El modelo cooperativo es el núcleo central de AMENITIES. Permite obtener un modelo conceptual global del sistema formado por un conjunto de modelos de comportamiento y tareas, para describir y representar cualquier sistema cooperativo desde el punto de vista de su estructura y funcionamiento. Su propósito es dar una descripción del sistema independiente de su implementación, proporcionando así una mejor comprensión del dominio del problema, dada la complejidad que caracteriza a estos tipos de sistemas.

El método para la construcción del modelo cooperativo es el paso intermedio que permite integrar el análisis social con el diseño del sistema. La notación COMO-UML se utiliza para construir el modelo cooperativo. Se trata de una notación eminentemente gráfica basada en UML que combina partes declarativas y operacionales, y que es propia de la metodología.

El modelo cooperativo se organiza en torno a cuatro vistas principales: *organizacional*, *cognitiva*, *de interacción* y *de información*. Estas vistas representan una partición del sistema centrada en algún aspecto particular [Schreiber 1994b]. El conjunto de conceptos manejados en cada uno de estos modelos está definido en el *marco conceptual de trabajo* mencionado en la sección 3.1. A continuación se describen con más detalle cada una de estas vistas, denominadas también *vistas conceptuales*, y el marco conceptual de trabajo definido en la metodología. Comenzamos por este último.

3.4.1 Marco conceptual de trabajo

Un marco conceptual de trabajo proporciona un conjunto de abstracciones o conceptos comunes a una familia de sistemas relacionados. En AMENITIES se propone un marco conceptual que comprende un conjunto con los conceptos principales y comunes a los sistemas colaborativos (ver Tabla II.1), así como, las principales relaciones entre ellos (ver Figura II.4). Este marco conceptual es usado como *metamodelo* para la especificación de un sistema cooperativo. El conjunto de todos estos elementos forma la base para la concepción abstracta de grupo según la metodología.

Tabla II.1 Definición de conceptos para el marco de trabajo del modelo cooperativo de AMENITIES [Garrido 2003]

Concepto	Definición
Evento	Ocurrencia de algún hecho que tiene una localización tanto en el espacio como en el tiempo.
Acción	Unidad básica de trabajo ejecutable atómicamente.
Artefacto	Dispositivo (hardware y/o aplicación software) utilizado para

	llevar a cabo ciertas acciones.
Objeto de información	Entidad que contiene la información requerida para llevar a cabo acciones, o que se genera como resultado de éstas.
Subactividad	Unidad de trabajo formada por un conjunto de acciones y otras subactividades que permite estructurar tareas.
Tarea	Conjunto de subactividades/acciones cuya realización permite alcanzar objetivos.
Actor	Usuario, programa o entidad que puede desempeñar roles.
Rol	Comportamiento esperado de un actor en base a un conjunto identificable de tareas a realizar.
Capacidad	Habilidad o responsabilidad asociada a un actor que le permite desempeñar roles y llevar a cabo tareas, subactividades o acciones.
Tarea cooperativa	Tarea en la cual para su realización interviene más de un actor, bien desempeñando el mismo rol o distintos.
Organización	Conjunto de roles, y relaciones entre ellos, que se dan en un lugar de trabajo.
Ley	Norma impuesta por una organización que restringe su funcionamiento en base a reglas sociales, culturales, capacidades de los actores, etc.
Grupo	Conjunto de actores desempeñando roles que pertenecen a una misma organización o que participan en la realización de tareas cooperativas.
Protocolo de interacción	Conjunto de reglas de comportamiento que utilizan los miembros de un grupo para llevar a cabo subactividades.

Los **eventos** disparan las **acciones**. Los eventos pueden ser: explícitos, por ejemplo, descrito mediante una acción de envío de una señal de continuar; o implícitos, cuando se termina la realización de una acción se dispara el comienzo de la siguiente. La acción es una unidad de trabajo indivisible que puede agruparse mediante el concepto de **subactividad**.

El **artefacto** podría formar parte de, o ser en sí mismo, un sistema groupware. El artefacto puede considerarse como la unidad tecnológica básica que sirve de soporte al sistema cooperativo. Por ejemplo, se considera artefacto, tanto al sistema groupware correspondiente a un calendario compartido distribuido, como al telepuntero (*telepointer*) que forma parte de él (como mecanismo encargado de proporcionar conciencia de grupo).

El **objeto de información** se entiende como una entidad pasiva utilizada para realizar acciones. En el contexto de un sistema CSCW, podría ser una entidad concreta tal como un documento electrónico (archivo), o una abstracta como un gesto. Este punto de vista es distinto al de los modelos y lenguajes computacionales orientados a objetos (Java, Smalltalk, etc.), donde los objetos no sólo contienen información, sino que también tienen un comportamiento.

La definición de **actor** incluye el término programa para hacer referencia a lo que también, a veces, se denomina agente inteligente, entendiendo éste como una entidad software, y hardware asociado, con actividad propia especializada en emular o comportarse de forma similar al ser humano, como por ejemplo, un robot.

La noción de **tarea** se utiliza para estructurar y describir el trabajo a llevar a cabo por el **grupo** en el sistema cooperativo. Las tareas en este marco conceptual se consideran a un alto nivel de abstracción, puesto que realmente se relacionan, o incluso se identifican, directamente con *objetivos* de los usuarios o del grupo, los cuales suelen formularse como cuestiones generales, sin entrar en detalles acerca de cómo conseguirlos.

El concepto de **rol** describe el comportamiento de los actores y es lo suficientemente general como para especificar el comportamiento humano, además de entidades de computación (p.e. agentes). Es el punto de partida para describir la evolución del comportamiento de los actores en el sistema. El rol puede verse como el estado de un actor. Permite enlazar entre las capacidades de los participantes y el trabajo a llevar a cabo especificado en términos de las tareas.

Las **leyes** y **capacidades** tienen que ver con las políticas que gobiernan el comportamiento del grupo, y por tanto, con el comportamiento social de sus miembros: obligaciones, permisos, prohibiciones, etc. Así, el sistema impone las restricciones (leyes) que rigen su funcionamiento obligando a los elementos que lo forman a comunicarse, coordinarse y colaborar. Los actores adquieren/abandonan habilidades o responsabilidades (capacidades) que en ciertos casos las leyes demandan para llevar a cabo el trabajo. Las capacidades abstraen los posibles estados del comportamiento de los actores en base estas habilidades personales o responsabilidades asumidas, y se adquieren/abandonan mediante la realización de acciones y/o la ocurrencia de ciertos eventos, interviniendo aspectos sociales, tales como, la aceptación de nuevos desafíos por parte de un miembro del grupo, priorización de tareas, etc.

Un **grupo** puede ser estable, es decir, impuesto formalmente por la organización (por ejemplo en base a estructuras jerárquicas, etc.), o esporádico, formado informalmente a iniciativa de los propios participantes (por ejemplo, por deseos o intereses particulares de asociación). La noción de rol, en cualquier caso, nos permite especificar y estructurar organizaciones según requisitos, además de la formación de grupos con objetivos concretos y a corto plazo.

Finalmente, un **protocolo de interacción** es una forma de especificar coordinación mediante la asociación de protocolos sociales y tecnológicos.

La descripción detallada y propósito de cada concepto de modelado es bastante más extensa que la expuesta aquí de forma resumida. Para ampliar conocimientos sobre algún

aspecto particular de los conceptos, consultar [Garrido 2003]. En la Figura II.4 se muestra una representación gráfica con las principales relaciones entre los conceptos definidos anteriormente. La notación utilizada corresponde a la de los diagramas de clases de UML.

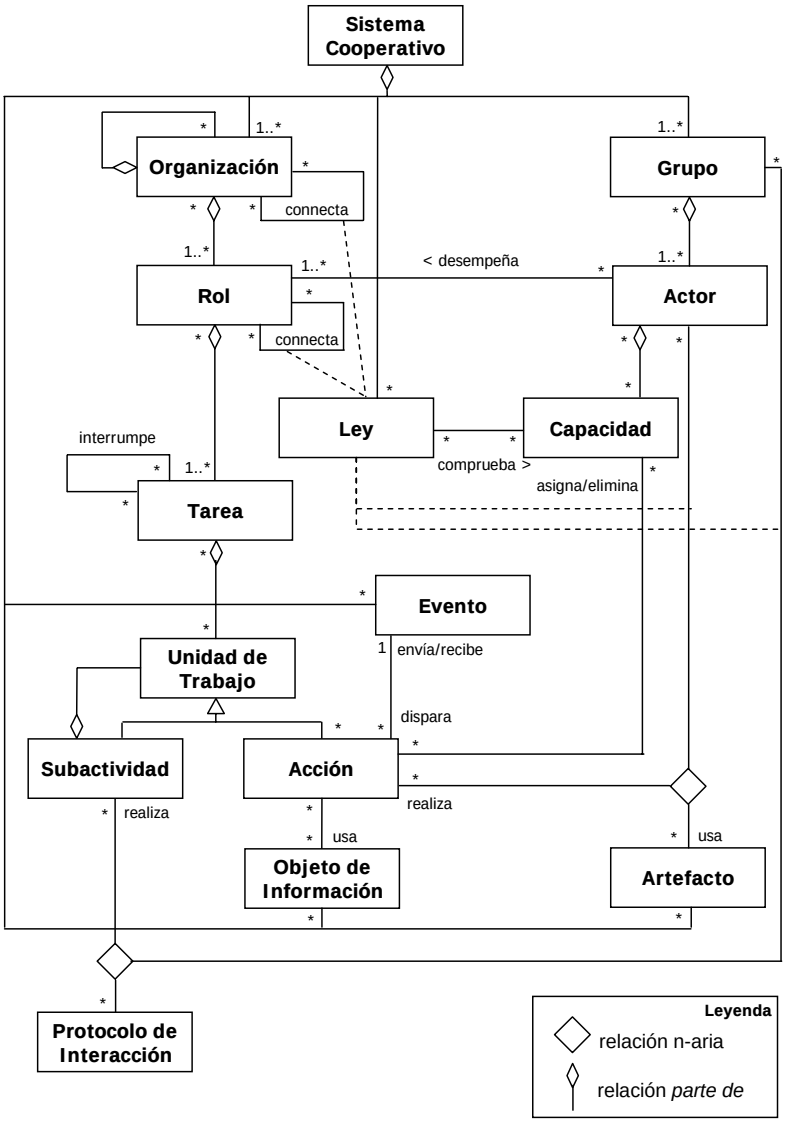


Figura II.4 Principales conceptos y relaciones del marco conceptual de AMENITIES [Garrido 2003]

Como se puede observar, el sistema cooperativo está formado a un primer nivel por:

- Una o más organizaciones, cada una de ellas estructuradas en base a un conjunto de suborganizaciones o roles relacionados.
- Uno o más grupos de actores.
- Leyes que establecen las reglas básicas de comportamiento del grupo dentro de las organizaciones.

- Eventos en el sistema.
- Artefactos disponibles.
- Información.

A un segundo nivel:

- Los roles incluyen tareas que, a su vez, están formadas unidades de trabajo, es decir, por subactividades, y las subactividades por otras subactividades y/o acciones.
- Los grupos están formados por actores, los cuales tienen adquiridas una serie de capacidades.

3.4.2 Vista organizacional

La vista organizacional permite describir cómo se estructuran los grupos en relación a aspectos tales como: grado de acoplamiento de los participantes, sus cargas de trabajo, la evolución del grupo, etc. Investigaciones relacionadas con la teoría de grupos y sistemas CSCW proporcionan dos claras corrientes diferenciadas en la forma de organizar los grupos de trabajo cooperativo:

1. En una corriente, las relaciones de trabajo cooperativo se definen en base a una **organización** *permanente* y *formal* del trabajo mediante estructuras preestablecidas, y bajo una coordinación normalmente centralizada.
2. La otra corriente parte de la concepción de la *diversidad* del trabajo cooperativo, tomando éste como una designación *informal* y neutral de múltiples personas trabajando en **equipo** para fines específicos, normalmente, bajo estructuras de coordinación descentralizadas.

AMENITIES permite especificar estas dos clases de grupos de trabajo las cuales vamos a denominar, respectivamente, **grupos de la organización** y **equipos de trabajo**.

3.4.3 Vista cognitiva

Normalmente, el conocimiento que cada persona tiene para poder realizar una tarea, es un subconjunto del conocimiento total que entre todos poseen los miembros de un grupo. Esta vista muestra el conocimiento de cada miembro del grupo en el contexto de una organización, y que se representa definiendo los roles que puede desempeñar y caracterizando las tareas asociadas a dichos roles.

La definición de roles sirve de enlace entre la especificación del comportamiento general de los grupos de la organización y las actividades individuales y/o colaborativas concretas a realizar por los miembros del grupo.

El concepto clave sobre el que se articula la conexión entre un grupo de la organización y el trabajo a realizar por éste, es el de **tarea**. La estrategia a seguir en esta etapa consiste en identificar y asociar las tareas a roles partiendo de la información, referente a comportamiento social, contenida en los modelos de requisitos (etnografía aplicada y diagramas de casos de uso de UML). También se identifican otros conceptos asociados. En particular, para cada tarea en cada rol se especifican los siguientes:

1. Conjunto de **tareas** que pueden interrumpir su realización. Estas tareas pueden pertenecer al mismo rol o a otros. Por defecto, la realización de una tarea no puede ser interrumpida.
2. **Eventos** que disparan la realización de la tarea.
3. **Leyes** que habilitan la realización de la tarea por miembros del grupo.
4. **Acciones** en las transiciones de entrada y salida a la tarea que pueden llevar a cabo cambios controlados en el estado global del sistema. Por ejemplo, contabilizando el número de participantes que están colaborando actualmente en la realización de una tarea, podemos limitar el número de participantes mediante una ley que tenga en cuenta el estado del sistema en relación al número de participantes que colaboran actualmente en dicha tarea.
5. **Objetos de información** requeridos para llevar a cabo la tarea. Estos objetos proporcionan parte de la vista de la información, ya que también se pueden incluir objetos de información en las etapas siguientes.
6. **Tipo de tarea** dependiendo de si es necesario más de un actor para llevarla a cabo (individual o cooperativa). En caso de ser una tarea cooperativa, hay que especificar la **multiplicidad**, es decir, el número de actores que han de intervenir en la realización de dicha tarea bajo cada uno de los roles implicados.

3.4.4 Vista de interacción

La interacción es un proceso natural en el grupo que tiene lugar, bien mediante la participación en tareas comunes, o bien cuando las acciones en el sistema de un miembro afectan directa o indirectamente al comportamiento de otros miembros o grupos.

Esta vista hace referencia a la interacción explícita entre participantes para realizar una actividad. Un ejemplo de este tipo de interacción es la que tiene lugar en una conversación cara a cara, donde no es posible garantizar que se cumplan las reglas o pasos que gobiernan el propio proceso de interacción, debido a que el comportamiento humano es impredecible. Es decir, se trata de casos en los cuales las tareas están débilmente estructuradas o sin estructurar.

El objetivo de la vista de interacción consiste en representar, mediante el concepto *protocolo de interacción* visto anteriormente, todos o algunos de los siguientes aspectos que se hayan podido identificar en una actividad que requiera interacción directa entre participantes:

- La forma de llevarla a cabo, sus restricciones temporales (síncrona o asíncrona) y el marco de tiempo (largo, medio y corto).
- Los participantes (humanos, agentes, etc.) y sus grados de cooperación.
- Uso de protocolos sociales para alcanzar el objetivo.
- Los requisitos de soporte a la comunicación entre participantes, y a la coordinación (controlada o guiada) de las acciones que deben llevar a cabo. Se tienen en cuenta factores relevantes relacionados con los artefactos utilizados (si los hay) en el proceso de interacción, ya que la interacción ocurre de alguna forma (cara a cara, redes de computadoras, etc.).

3.4.5 Vista de información

La vista de información representa datos tales como documentos y mensajes, cuya principal finalidad es comunicar información de una manera más o menos estructurada. En el modelo cooperativo de AMENITIES, la información es un elemento más del sistema que puede estar implícita (por ejemplo, los mensajes) en actividades o acciones, y si se cree conveniente, también se puede hacer explícita dentro del flujo de información entre éstas.

La vista de información necesita tanto de la vista cognitiva, como de la vista de interacción, puesto que las entidades que representa (documentos, mensajes, etc.) están ligadas directamente a las subactividades/acciones de cualquier tipo que se realizan en el sistema cooperativo. Por ejemplo, un mensaje entre dos usuarios o entre un usuario y la aplicación no tiene sentido, a no ser que esté ligado a algún proceso de colaboración concreta. Adicionalmente, se propone una representación independiente de la información en base a su estructura (agregación y composición) y otras relaciones (generalización/ especialización y asociación).

3.5 Modelo formal

A partir del modelo cooperativo es posible derivar automáticamente una Red de Petri Coloreada (CPN) [Jensen 1996], aplicando esquemas de traducción adecuados con los elementos de la notación COMO-UML [Garrido 2002b]. El resultado es un modelo formal que permite llevar a cabo una simulación del funcionamiento del sistema.

Mediante la ejecución automática o guiada de la red obtenida se pueden validar los modelos construidos y evaluar la usabilidad del sistema en términos de tareas. En concreto, con simulaciones automáticas se pueden obtener medidas del rendimiento, y mediante grafos de ocurrencias o álgebra es posible estudiar propiedades tales como interbloqueos, alcanzabilidad, etc.

3.6 Modelos de desarrollo de software

Para conectar con el desarrollo del software, AMENITIES propone hacer uso de las notaciones disponibles en UML para describir una arquitectura software. Esto es, el punto de encuentro entre los modelos propuestos en la metodología (análisis y diseño), y los modelos software (implementación), va a ser el modelo arquitectónico de la aplicación groupware que da soporte al sistema colaborativo.

En general, una arquitectura del software refleja de alguna forma cómo se organizan y comunican los componentes software de un sistema, pero sin entrar en detalles acerca de la funcionalidad y composición interna de tales componentes [Bass 2003]. En el caso de los sistemas CSCW es especialmente importante obtener la implementación del sistema a partir de un conjunto de subsistemas que se comunican a través de interfaces bien definidas, ya que un sistema cooperativo es inherentemente un sistema distribuido.

Con carácter general, el análisis y modelado de tareas realizado para construir el modelo cooperativo de un sistema CSCW, se puede utilizar para dirigir el desarrollo de software de varias formas (análisis de requisitos, derivar interfaces de usuario, evaluación, etc.). En AMENITIES, el modelo cooperativo (sección 3.4), que sirve como punto de partida para el desarrollo del software, no sólo describe tareas, sino que además abarca e integra junto con éstas otros conceptos tales como: organización, roles, etc., y análogamente a la utilización tradicional de modelos de tareas [Paternò 2000], también permite derivar de forma semiautomática la interfaz de usuario de las aplicaciones groupware.

3.7 Método de modelado

Para construir el modelo cooperativo se sigue un método sistemático dividido en cuatro etapas:

1. Especificación de cada **organización** mediante un conjunto de roles conectados por transiciones.
2. Definición de cada uno de los **roles** en base a conjuntos de tareas.
3. Descripción de las **tareas** mediante conjuntos de subactividades y acciones.

4. Especificación de **protocolos de interacción** para aquellas subactividades que lo requieran.

Los dos conceptos claves en los que se basa el método son **tarea** y **rol**. Asimismo, cada una de las etapas específicas de la que consta el método, maneja otros conceptos (acciones, eventos, etc.), que contribuyen a la especificación de los distintos aspectos del dominio del problema, tales como, comportamiento, interacción, coordinación, etc.

El método mantiene las relaciones semánticas entre las distintas representaciones de cada una de las etapas siguiendo las relaciones de agregación entre los conceptos del marco de trabajo propuesto (véase Figura II.4), y representando dentro de cada nivel el resto de las relaciones entre conceptos. El resultado final es un único modelo conceptual construido y organizado anidadamente.

A continuación se propone un ejemplo de aplicación del método de modelado definido en AMENITIES para la obtención del **modelo cooperativo** del sistema. Asimismo, este ejemplo lo utilizaremos de forma recurrente como caso práctico para ilustrar las distintas propuestas en otros capítulos del presente trabajo. En concreto, se pretende modelar el entorno cooperativo correspondiente a un sistema de **concesión de préstamos hipotecarios** para una sucursal bancaria donde participan tres organizaciones distintas: la *sucursal bancaria* donde el cliente ha solicitado el préstamo, una *notaría* para dar fe pública a los documentos de firma y una *agenciaDeTasación*. Comenzamos con la etapa de especificación de la **organización**. Detallar más.

3.7.1 Especificación de la organización

En esta etapa se definen los **grupos** de las organizaciones que van a cooperar. Aquí el *grupo* se concibe como una composición formal y permanente de participantes impuesta por cada organización, en contraposición al *equipo de trabajo*, que presenta un carácter informal y menos permanente de sus participantes, que a su vez, tienen objetivos de colaboración concretos para llevar a cabo trabajos más específicos [Garrido 2003]. La especificación de los grupos contribuye a conformar la *vista organizacional* del modelo cooperativo (cfr. Figura II.3 y sección 3.4.2). Los equipos de trabajo se definen más adelante, durante la especificación de la *vista cognitiva*, centrada en la descripción de tareas.

Por otro lado, los grupos en las organizaciones se configuran básicamente a través del concepto de rol [Nurcan 1998]. Por tanto, en esta primera etapa, se describe la estructura fundamental de una organización en base a roles y las dinámicas posibles del conjunto de actores (grupo) que desempeñan esos roles, mediante adecuadas conexiones entre ellos. Mediante el concepto de rol, se describe la estructura y dinámicas de una organización independientemente del actor o actores concretos que desempeñen dichos roles.

De este modo, la vista organizacional comprende, por un lado, la identificación de los roles que intervienen en un sistema y sus interrelaciones, y por otro, las leyes que impone la organización para regular el comportamiento de los actores en el sistema en base a cambios de rol u organización (lo que implica de forma implícita un cambio de rol).

Para representar los conceptos presentados anteriormente de forma gráfica, AMENITIES propone la utilización de los denominados *diagramas de organización* de COMO-UML (basados en los diagramas de estados de UML [OMG 2007]).

En el caso particular del sistema de concesión de préstamos podríamos elaborar los siguientes diagramas de organización para la sucursal bancaria, la notaría y la oficina de tasación, tal como se muestra en la Figura II.5.

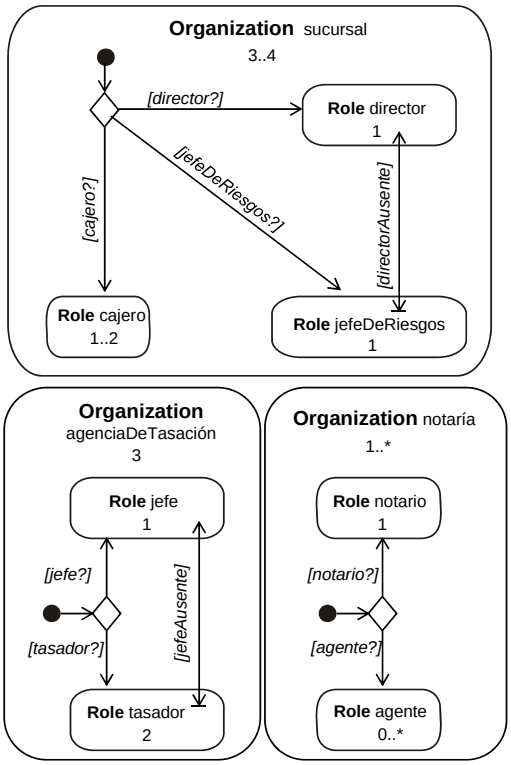


Figura II.5 Diagramas de organización para un sistema de concesión de hipotecas

Como puede apreciarse en la figura anterior, cada rol en cada organización se representa mediante un estado y se corresponde con las cajas redondeadas. Así, en la sucursal bancaria hemos identificado tres roles: cajero, responsable de riesgos y director. Cada uno de estos estados forma parte de otro estado compuesto que representa a la organización a la que pertenecen. En la Figura II.5 pueden observarse tres estados compuestos de este tipo: *sucursal*, *agenciaDeTasación* y *notaría*, correspondientes a cada una de las organizaciones.

Bajo el nombre de la organización se puede especificar el número de actores que forman parte de ella mediante un valor numérico o un rango que denote su posible multiplicidad. En el ejemplo de la Figura II.5, se ha establecido que, para el caso de la notaría, al menos un actor ha de formar parte de la organización (expresado por el rango 1..*), mientras que para la agencia de tasación el número de actores se ha fijado en 3, y para la sucursal en 3 ó 4.

De forma análoga, debajo del identificador de rol aparece un número que indica la multiplicidad o número de actores que deben desempeñar dicho rol. Por ejemplo, para la sucursal bancaria se ha establecido que habrá un actor desempeñando el rol de *director*, un actor desempeñando el rol de *responsable de riesgos* y uno o dos actores como *cajeros*. En la oficina de tasación está previsto que haya dos actores desempeñando el rol *tasador* y un actor el rol *jefe*. Finalmente, en la notaría, un actor desempeñará el rol de *notario* y, opcionalmente, puede haber actores desempeñando el rol de *agente* (expresado con el rango 0..*).

El conjunto de comportamientos posibles para el grupo de actores, se modela en este diagrama conectando roles por medio de transiciones. Cada transición viene regulada por una *ley* (expresada como una guarda, esto es, entre corchetes) (cfr. Figura II.3). Las leyes pueden incluir *capacidades* (expresadas como una cadena de caracteres con el símbolo "*?*" al final), que interrogan las habilidades adquiridas o responsabilidades asumidas por cada actor particular, de tal forma que, para que dicho actor pueda desempeñar el rol de destino de la transición, antes deba estar habilitado para ello. La evaluación de una guarda como verdadera, hace posible disparar su transición asociada, y significa que es posible llevar a cabo el cambio de rol correspondiente. Así, en el ejemplo de la Figura II.5, para determinar el rol que desempeñará inicialmente cada actor se interrogan previamente sus capacidades: *cajero?*, *jefeDeRiesgos?*, *director?*, *notario?*, etc.

Por otro lado, conviene destacar la naturaleza de un tipo especial de transición, llamada **transición aditiva** y denotada mediante un arco con doble flecha y una pequeña barra vertical que especifica el rol de origen. Esta transición permite expresar la posibilidad de que un actor desempeñando el rol de origen pueda interrumpir temporalmente las tareas que tiene encomendadas para atender las tareas propias del rol destino. Todo ello sin necesidad de llevar a cabo un cambio de rol y mientras esté vigente la condición de la guarda, es decir, se satisfaga la ley asociada a la transición. En la Figura II.5 podemos encontrar dos ejemplos de transiciones aditivas, como son la transición que conecta el rol *jefeDeRiesgos* con el rol *director* en la organización de la *sucursal*; y la que conecta el rol *tasador* con el rol *director* en la *agenciaDeTasación*. En el caso de la sucursal, se especifica que un actor desempeñando el rol de *jefeDeRiesgos* puede atribuirse las funciones del *director* por

ausencia de éste último (ley *directorAusente*). Lo mismo puede aplicarse, entre los roles *tasador* y *jefe*, al caso de la agencia de tasación.

Finalmente, en los diagramas de organización, un círculo relleno denota el seudoestado inicial, y un rombo, distintas alternativas de conexión entre roles siempre y cuando se den las condiciones expresadas en las guardas de las transiciones correspondientes.

Por tanto, en cualquier momento, el comportamiento de cada actor viene dado por el rol que desempeña, por las capacidades adquiridas y por las leyes que la organización especifica para permitir desempeñar otros roles. Ello implica que el conjunto de comportamientos posibles que un actor puede realizar está restringido por sus cualidades y otras normas determinadas por la organización. Asimismo, los participantes pueden influir en el comportamiento del resto de miembros del grupo. Por ejemplo, el hecho de que se ausente el *director* de la sucursal posibilita que el actor que desempeña el rol *jefeDeRiesgos*, asuma también el dicho rol.

Tal como se verá en las siguientes etapas, los actores también pueden modificar su comportamiento sin tener que llevar a cabo cambios de rol. Igualmente, será posible que un actor desempeñando un rol interrumpa la realización de un trabajo para pasar a realizar otro nuevo, bien cambiando de rol o sin necesidad de ello.

3.7.2 Definición de roles

Esta etapa, junto con la etapa siguiente, ayuda a conformar la vista cognitiva que proporciona el modelo cooperativo (cfr. Figura II.3 y sección 3.4.3). La definición de roles sirve de enlace entre la especificación del comportamiento de los grupos de la organización y las actividades individuales y/o colaborativas a realizar por los miembros del grupo.

El concepto clave sobre el que se articula la conexión entre un grupo de la organización y el trabajo a realizar por éste, es el de tarea. En esta etapa se identifican las tareas y se asocian a roles a partir de la información referente a comportamiento social, contenida en los modelos de requisitos de AMENITIES (etnografía aplicada y diagramas de casos de uso de UML, entre otros). Adicionalmente, para cada tarea asociada a cada rol se pueden especificar los siguientes elementos:

- Otras **tareas** que pueden interrumpir su ejecución. Estas tareas pueden pertenecer al mismo rol o a otros. Por defecto, la realización de una tarea no puede ser interrumpida.
- **Eventos** que disparan la realización de la tarea.
- **Leyes** que habilitan la ejecución de la tarea por miembros del grupo.

- **Acciones** en las transiciones de entrada y salida a la tarea, que pueden conllevar cambios controlados en el estado global del sistema.
- **Objetos de información** requeridos para llevar a cabo la tarea, u obtenidos como resultado de su realización. Estos objetos también forman parte de la *vista de información*, como se verá más adelante.
- Tipo de la tarea, individual o cooperativa, dependiendo de si son necesarios varios actores para llevarla a cabo.
- En caso de que sea una tarea cooperativa, hay que especificar la multiplicidad, es decir, el número de actores que han de intervenir en la realización de dicha tarea bajo cada uno de los roles implicados.

La notación gráfica del lenguaje COMO-UML permite la especificación de roles por medio de los denominados *diagramas de rol*. En la Figura II.6 se muestran parcialmente ejemplos de estos diagramas para los roles *jefeDeRiesgos*, *tasador*, *cajero* y *notario*.

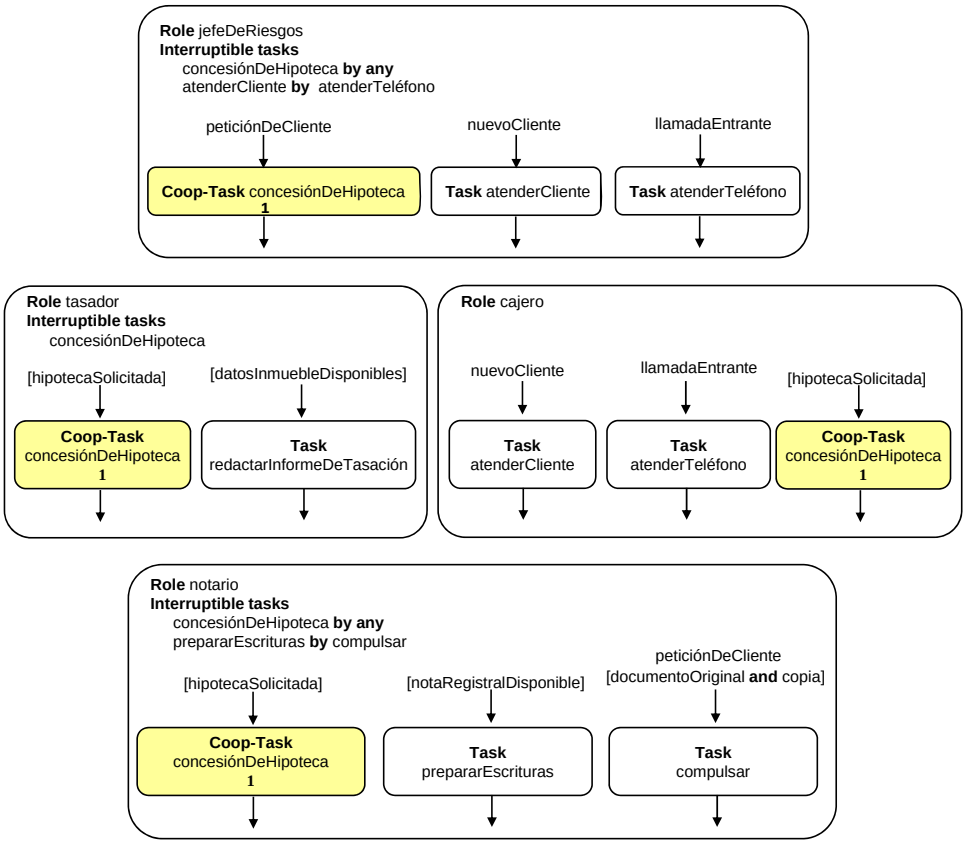


Figura II.6 Diagramas de roles para un sistema de concesión de hipotecas

Como puede apreciarse, existe una tarea cooperativa *concesiónDeHipoteca* en la que participan tres de los roles representados (*jefeDeRiesgos*, *tasador* y *notario*) y que aparece denotada mediante la palabra reservada Coop-Task.

Asimismo, cada rol puede tener encomendadas otras tareas. Tal es el caso también de los roles de la figura: el actor que desempeñe el rol *jefeDeRiesgos*, deberá además de ocuparse, entre otros asuntos, de la atención al cliente (tarea *atenderCliente*) y contestar las llamadas que reciba (tarea *atenderTeléfono*); el rol *tasador* es el encargado de la realización de la tarea *redactarInformeDeTasación*; el *cajero* se encarga básicamente de la atención al cliente, en persona o por teléfono; y la función de *notario*, incluye las tareas de *prepararEscrituras* y *compulsar*.

Mediante las palabras reservadas Interruptible tasks se especifican qué tareas dentro cada rol, pueden ser interrumpidas, y por qué tareas en concreto, pudiendo estas pertenecer a diferentes roles. Por tanto, esta sección dentro de los diagramas de roles permite especificar la importancia relativa de las tareas dentro de la organización. Así, el actor que desempeña el rol *jefeDeRiesgos* puede interrumpir las funciones que esté desarrollando para estudiar la concesión de una hipoteca, para realizar cualquier otra tarea que tenga asignada (veáse el uso de la palabra reservada any, en este caso). Igualmente, la tarea *atenderCliente* de este mismo rol puede verse interrumpida porque se produce una llamada que el *jefeDeRiesgos* deba recibir. Por otro lado, para el caso del rol *cajero*, esto no es así, y tanto la tarea *atenderCliente*, como la tarea *atenderTeléfono*, no pueden ser interrumpidas.

No obstante, el hecho de que una tarea sea interrumpible no significa que pueda ser parada en cualquier momento. Al descomponer una tarea en otras subtareas y acciones más sencillas, pueden especificarse componentes, o secciones que agrupan a varios de ellos, como no interrumpibles. Esto lo veremos más adelante en la sección 3.7.3 al especificar las tareas.

Adicionalmente, para cada tarea puede especificarse el evento que dispara su realización. Si no se especifica dicho evento, el comienzo de una tarea depende del criterio del actor que, por su rol, la tiene encomendada. Por ejemplo, la tarea *compulsar* del *notario* se inicia a petición de un cliente y siempre que se disponga del documento original y la copia que se quiera compulsar (véase la ley *documentoOriginal* and *copia*). En cambio, la tarea *prepararEscrituras* la puede iniciar cuando lo desee, siempre que disponga de la nota registral del inmueble a escriturar (ley *notaRegistralDisponible*).

3.7.3 Descripción de tareas

En esta etapa se especifica con más detalle cómo se deben desarrollar las tareas identificadas en la etapa anterior. De esta forma, se termina de configurar la vista cognitiva del sistema cooperativo (cfr. sección 3.7.2).

Las tareas se definen utilizando diagramas de COMO-UML denominados *diagramas de tareas*, que son adaptaciones y extensiones de los *diagramas de actividades* de UML. En este punto, conviene destacar que la metodología AMENITIES fue ideada tomando como referencia las versiones 1.4 [OMG 2001] y 1.5 [OMG 2003] de UML. Este punto es importante ya que, en la actual versión UML 2.0, los diagramas de actividades han sido rediseñados para dotarlos de una semántica análoga a la de las *redes de Petri*, y por tanto, diferente a la semántica de *máquinas de estados* utilizada en las versiones anteriores del lenguaje, como UML 1.5 y utilizadas como referencia en AMENITIES. Otro cambio importante entre las versiones del lenguaje mencionadas es la utilización del término *actividad* en substitución del término *subactividad*[1]. Por otro lado, cada actividad o acción presente en un diagrama de actividades puede iniciarse porque otras actividades o acciones finalizan su ejecución, porque ciertos objetos y/o datos pasan a estar disponibles, o por la ocurrencia de algún evento externo al flujo de la actividad [OMG 2007]. Por último, las acciones representan unidades de comportamiento atómicas que no pueden ser descompuestas en otras más simples.

En esta etapa se abordan cuestiones de coordinación relativas a procesos y tareas del sistema, y que hacen posible colaborar de forma efectiva. La coordinación entre actividades, que implica la coordinación entre sus participantes, ya que, al definir las actividades también se establecen sus participantes y cuándo intervienen en la misma, los aspectos de coordinación están presentes durante todo el proceso de modelado. Por ejemplo, tal como se ha mostrado anteriormente, la especificación de eventos para iniciar la realización de tareas, también proporciona una forma de coordinación.

Tal y como se ha mostrado anteriormente,, en la etapa de *definición de roles* se especifica si una tarea es individual (la realiza un solo actor) o cooperativa (se requiere más de un actor para llevarla a cabo). Así, la descripción de una tarea individual es normalmente más simple que la de una cooperativa. Por ello, para ilustrar esta etapa utilizando la notación COMO-UML, se ha escogido la especificación de la tarea cooperativa *concesiónDeHipoteca* (ver Figura II.7).

[1] En la descripción del marco conceptual y su posterior representación en la Figura II.4 se ha utilizado el término "subactividad" para preservar el modelo tal y como fue presentado originalmente en la descripción de la metodología AMENITIES. Sin embargo, en adelante utilizaremos el término "actividad" para alinearnos con la nueva versión 2.0 del lenguaje UML.

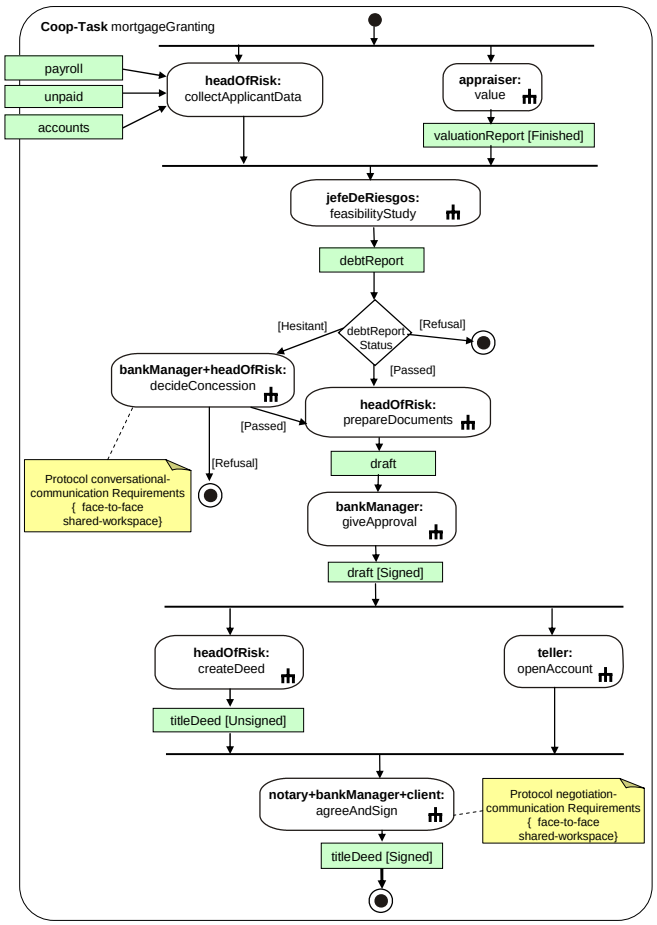


Figura II.7 Diagrama de tareas para *concesiónDeHipoteca*

La Figura II.7 representa una planificación con la secuencia de pasos necesaria para conceder una hipoteca a través de una serie de actividades (no se han representado acciones) que, a su vez, pueden descomponerse en otras actividades y acciones más sencillas. Esto se representa mediante un pequeño tridente simbolizando una jerarquía de descomposición (las acciones no incluyen este símbolo). Por tanto, el diagrama de la figura corresponde al nivel más alto de especificación de la tarea.

Como puede apreciarse, la tarea cooperativa *concesiónDeHipoteca* se inicia con dos tareas en paralelo, *recopilarDatosSolicitante* y *tasar*, y que son realizadas por el *jefeDeRiesgos* y el *tasador*, respectivamente. El proceso concluye con el acuerdo y firma por parte del *notario*, el *director* de la sucursal y el *cliente* (tarea *acordarYFirmar*). No obstante, pueden darse otros cursos alternativos para el desarrollo de la tarea. En efecto, la presencia de una caja de decisión (representada con un rombo) bajo el objeto de información *informeDeDeudas*, define en este caso tres caminos posibles dependiendo de que se cumplan ciertas condiciones sobre dicho informe generado en la actividad *estudioDeViabilidad*:

1. Si el informe arroja ciertas dudas sobre la viabilidad del préstamo, expresado mediante la condición *[dudoso]*, el siguiente paso en la secuencia es la actividad *decidirConcesión*. Tras esta tarea, el informe puede ser a aprobado y en ese caso se pasa a la actividad *prepararDocumentos*. Si es rechazado, se concluye el proceso de concesión.
2. Si el informe es aprobado (condición *[aprobado]*), se pasa directamente a la actividad *prepararDocumentos*.
3. Por último, si el informe indica que se debe rechazar la solicitud (condición *[rechazado]*), se finaliza la realización de la tarea.

Las tareas y acciones pueden tener asociados comentarios (representados mediante una caja cuadrada con la esquina superior derecha doblada, llamada también símbolo de "notas"), para especificar restricciones generales, leyes, requisitos y protocolos de interacción asociados a actividades abstractas.

Este es el caso de la actividad *acordarYFirmar*, cuyo comentario asociado contiene la cláusula non-interruptible, para indicar que su realización no puede ser interrumpida. Recordemos que en la sección anterior de definición de roles, vimos que la tarea cooperativa *concesiónDeHipoteca* para el caso del rol *notario*, en general, podía ser interrumpida por otras tareas. Este comentario, actualiza y restringe el ámbito en el que el *notario* podría interrumpir la tarea *concesiónDeHipoteca*, impidiendo que lo haga durante la firma del préstamo. La misma restricción es aplicable a los roles *director* de la sucursal y *cliente*. De este modo, es posible modificar o añadir diferentes aspectos (restricciones, leyes, etc.) que afectan con carácter individual a ciertas actividades/acciones concretas dentro de una tarea. En este caso, el comentario afecta sólo a una actividad, aunque podría abarcar a más de una actividad y/o acción. Los otros comentarios que aparecen en el diagrama de la Figura II.7, atañen a la especificación de *protocolos de interacción* que trataremos en la próxima sección.

Dentro de la caja redondeada que representa a cada actividad puede especificarse el rol o los roles (que pueden pertenecer a organizaciones diversas) a los que se ha asignado una tarea. La especificación de varios roles, unidos por el operador "+", define un *equipo de trabajo*. El nombre del rol de la organización o del equipo de trabajo responsable de realizar una actividad/acción aparece antes que el nombre de ésta, y separada por el carácter dos puntos. Como ya vimos en la sección 3.7.1., los equipos de trabajo son útiles para describir dinámicas de grupos con fines concretos a corto plazo cuando se requiere especificar colaboración entre organizaciones. Un ejemplo de esto lo podemos encontrar en la actividad *acordarYFirmar*, donde participan el *notario*, el *director* de la sucursal y el *cliente* (ver expresión notario+director+cliente). Con la especificación de equipos de trabajo se completa la *vista organizacional*.

Como comentamos anteriormente, las tareas representadas dentro de un diagrama de actividad pueden contener a su vez secuencias de otras tareas y acciones. En la Figura II.8 se ha representado el resultado de la descomposición de la actividad de tasación de un inmueble (tarea *tasar*) y que forma parte de la tarea cooperativa *concesionDeHipoteca* (cfr. Figura II.7)[2].

En la Figura II.8, puede apreciarse que comienzan a aparecer acciones, esto es, unidades de trabajo que no van a ser descompuestas en otras más simples como, por ejemplo, *asignarMejorTasadorDisponible* o *enviarInformeCliente*. Las acciones y actividades requieren de la especificación de sus participantes, como veíamos para cada una de las tareas de la Figura II.7, pero es posible prescindir de esta especificación si en una actividad de nivel superior donde esté contenida, recoge dicha especificación.

En la especificación de las tareas de las figuras anteriores se incluyen *objetos de información* que forman parte de la *vista de información*. Los objetos de información aparecen representados como cajas rectangulares que contienen el nombre del objeto y, opcionalmente, el estado del mismo especificado entre corchetes. Por ejemplo, en la Figura II.8 se puede observar que, como resultado de la tarea *introducirDatosEnSistemaDeInformación* se genera un *informeDeTasación* (objeto de información) que ha de ser validado (estado *[noValidado]*). Posteriormente, este informe se envía al departamento de control para su validación (acción *enviarInformeDpto.Control*). El departamento de control devolverá dicho informe validado (estado del objeto *[validado]*).

[2] Proceso de tasación tomado y adaptado de TASACIONES y CONSULTORÍA S.A., entidad colaboradora de Caja Navarra. Disponible en http://www3.cajanavarra.es/PortalCAN/es-ES/_Inmobiliario/Tasaciones/_Pestanas/ProcesoTasación/

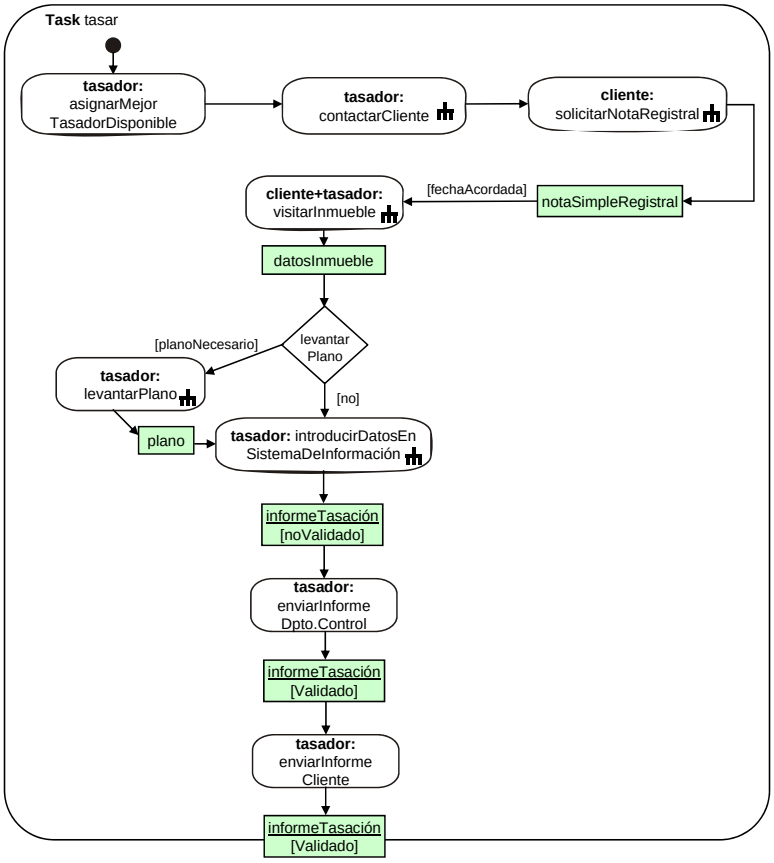


Figura II.8 Diagrama de tareas para la actividad *tasar*

3.7.4 Especificación de los protocolos de interacción

En esta última etapa se aborda la especificación acerca de cómo va a tener lugar la interacción entre participantes durante la realización de una actividad poco o nada estructurada. Hasta ahora, habíamos visto cómo el comportamiento de un actor puede afectar indirectamente a otro, y por extensión, a los grupos a los que pertenecen, al cumplirse ciertas condiciones expresadas a través de leyes. Por ejemplo, anteriormente ya vimos que para la organización *sucursal* de la Figura II.5, un participante desempeñando el rol *jefeDeRiesgos* puede también desempeñar el rol *director*, durante la ausencia del último. Ello se evalúa en la ley *directorAusente* que etiqueta la transición aditiva entre estos dos roles.

La interacción directa entre participantes puede tomar múltiples formas dependiendo fundamentalmente de la naturaleza de la actividad y de los roles que desempeñan los actores implicados. Los protocolos de interacción están concebidos para poder especificar el soporte tecnológico necesario para poder llevar a cabo tareas que por su propia naturaleza, no se pueden estructurar o no es práctico hacerlo. Así, los protocolos de interacción establecen los requisitos/propiedades que caracterizan este tipo de interacción entre participantes recogiendo aspectos tan diversos como protocolos sociales, patrones de

comportamiento, tecnologías, etc. Los protocolos tecnológicos implícitos de los artefactos utilizados en una actividad (por ejemplo, el de una videoconferencia o el del correo electrónico), deben proporcionar el soporte adecuado a las interacciones sociales entre participantes. Con esta etapa se proporciona la *vista de interacción* descrita al explicar el modelo cooperativo (cfr. sección 3.4).

Se pueden especificar y definir los protocolos de interacción que se consideren oportunos. Algunos ejemplos de ellos son los siguientes:

- **Negociación** (*negotiation*): donde los participantes pueden formular preguntas, ideas, sentencias o peticiones, y obtener respuesta o solución de otros participantes, sin orden específico en las intervenciones.
- **Petición-respuesta** (*request-reply*): interacción síncrona donde para cada petición que se formule, a continuación se ha de obtener la correspondiente respuesta, la cual espera el solicitante antes de continuar.
- **Mensajes encolados** (*queued-messages*): similar a *petición-respuesta*, pero permitiendo una interacción asíncrona entre participantes, es decir, no se requiere una respuesta a cada petición.

La especificación de un protocolo de interacción también puede contener información acerca de:

1. Los roles participantes que tienen responsabilidades en la actividad a la que están asociados. La palabra reservada all determina que el protocolo debe ser seguido por todos los participantes implicados en la tarea cooperativa independientemente del rol que desempeñen.
2. Requisitos específicos de comunicación tales como, que exista un espacio de trabajo compartido (shared workspace), que cada participante puede ver al resto, lo cual se corresponde con una interacción cara a cara (face-to-face), etc.
3. Cualquier otro elemento necesario tales como restricciones o leyes.

Por ejemplo, para la actividad *decidirConcesión* en la tarea de la Figura II.7, se ha establecido que el *jefeDeRiesgos* y el *director* de la *sucursal* van a seguir un protocolo conversacional, cara a cara, compartiendo un mismo espacio de trabajo. De este modo, el desarrollador podría escoger, por ejemplo, una herramienta de videoconferencia como soporte tecnológico para la realización de dicha actividad en el sistema colaborativo.

REFERENCIAS DEL CAPÍTULO

[ACM 1992] ACM SIGCHI Curriculum Development Group. "ACM SIGCHI Curricula for Human Computer Interaction". New York: ACM, 1992. ISBN 0-89791-474-0

[Bannon 1989] Bannon, L.J., Schmidt, K.: "CSCW: Four Characters in Search of a Context". In Proc. 1st European Conf. on Comp. Supported Cooperative Work, pages 358-372. Computer Sciences House, Sloug, UK, Sep. 1989

[Bass 2003] Bass, L., Clements, P., Kazman, R.: "Software Architecture in Practice". 2nd edition, Addison-Wesley, 2003

[DeSanctis 1987] DeSanctis, G., Gallupe, B.: "A Foundation for the Study of Group Decision Support Systems". Management Science 33 (5) (1987), pp. 589-609

[Dillenbourg 1995] Dillenbourg, P., et al.: "The Evolution of Research on Collaborative Learning". In: Reimann, P., Spada, H. (eds.) Vol. Learning in humans and machines. Towards an interdisciplinary learning science, London, pp. 189–211 (1995)

[Dix 1998] Dix, A., Finlay, J., Abowd, G., Beale, R.: Human Computer Interaction, second edition. Prentice Hall (1998)

[Ellis 1991] Ellis, C.A., Gibbs, S.J., Rein, G.L.: "Groupware: Some Issues and Experiences", Communications of the ACM, Vol. 34, No. 1 (January 1991) 38-58

[Garrido 2002] Garrido, J.L., Gea, M., Padilla, N., Cañas, J.J., Waern, Y.: "AMENITIES: Modelado de Entornos Cooperativos". In: Aedo, I., Díaz, P., Fernández, C. (eds.):Actas del III Congreso Internacional Interacción Persona-Ordenador 2002 (Interacción'02), Madrid, Spain (Mayo 2002), 97-104

[Garrido 2002b] Garrido, J.L., Gea, M.: "A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling". In: Interactive Systems - Design, Specification and Verification. Revised papers. LNCS 2545. Springer (2002), 16-28

[Garrido 2003] Garrido, J.L.: "AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas". Tesis Doctoral, Granada 2003

[Grudin 1988] Grudin, J.: "Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces". Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work. CSCW '88. ACM Press, 85-93

[Grudin 1994] Grudin, J.: "Computer-supported cooperative work: History and focus." IEEE Computer, 27, 5, 19-26 (1994)

[Jacovi 2006] Jacovi, M., Soroka, V., Gilboa-freedman, G., Ur, S., Shahar, E., Marmasse, N.: "The chasms of CSCW: a citation graph analysis of the CSCW conference". Proceedings of the 2006 ACM Conference on Computer-Supported Cooperative Work, ACM Press, 289-298

[Jensen 1996] Jensen, K.: "Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use". Second Edition, Springer (1996)

[Johansen 1988] Johansen, R.: "Groupware: Computer Support for Business Teams". The Free Press, 1988

[Lynch 1990] Lynch, K.J., Snyder, J.M., Vogel, D.R., McHenry, W.K.: "The Arizona Analyst Information System: Supporting Collaborative Research on international Technological Trends". Proceedings of IFIP WG8.4 Conference on Multi-User Interfaces and Applications, Crete, Greece (1990)

[Malone 1990] Malone, T.W., Crowston, K.: "What is Coordination Theory and How Can It Help Design Cooperative Work Systems". Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'90). ACM Press, New York (1990) 357-370

[Nurcan 1998] Nurcan, S.: "Analysis and Design of Co-operative Work Processes: A Framework". Information and Software Technology, Elsevier, 40(3), (1998) pp.143-156

[OMG 2001] OMG, "Unified Modeling Language Specification", version 1.5, formal/2001-09-67. http://www.omg.org/spec/UML/1.4/

[OMG 2003] OMG, "Unified Modeling Language Specification", version 1.5, formal/2003-03-01. http://www.omg.org/spec/UML/1.5/

[OMG 2007] OMG, "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[Paternò 2000] Paternò, F.: "Model-based Design and Evaluation of Interactive Applications". Springer-Verlag (2000)

[Rumbaugh 1999] Rumbaugh, J., Jacobson, I., Booch, G.: "The Unified Modeling Language - Reference Manual". Addison-Wesley (1999)

[Schlichter 1998] Schlichter, J.H., Koch, M., Bürger, M.: "Workspace Awareness for Distributed Teams". Coordination Technology for Collaborative Applications 1998: 199-218

[Schreiber 1994b] Schreiber, G., Wielinga, B., Akkermans, H., Van de Velde, W., Anjewierden, A.: "CML: The CommonKADS Conceptual Modelling Language". A Future for Knowledge Acquisition, 8th European Knowledge Acquisition Workshop. LNCS 867, Springer-Verlag, Berlin, 1994, pp. 1-25

Capítulo III

ONTOLOGÍAS

1. INTRODUCCIÓN

Sin duda, la investigación en ontologías y su aplicación al modelado conceptual de SI constituyen hoy en día un campo de exploración reconocido que continúa ampliando sus horizontes en distintos ámbitos relacionados con la Informática, tales como: la ingeniería y la representación del conocimiento, modelado e integración de información, diseño de bases de datos, análisis orientado a objetos, diseño de sistemas basados en agentes, la Web Semántica, etc. [Guarino 1998].

Al igual que en los capítulos anteriores, antes de presentar en detalle algunas de las propuestas existentes para el análisis, diseño y/o implementación de ontologías, conviene realizar algunas precisiones terminológicas.

Así, en Filosofía, la *Ontología* (οντος –ser, estar–, y λογος –ciencia, estudio, teoría–), es la rama de la *Metafísica* que estudia *lo que es, en tanto que es y existe*, es decir, qué es, cómo es y cómo es posible. El estudio ontológico acerca del mundo que hace el ser humano parte de lo que éste *percibe* por algún medio. Con este significado, *Ontología* es normalmente escrito con su primera "*O*" en mayúscula y en singular. De esta forma, se distingue de *ontología* (con minúscula), utilizado para referirnos a una concreta, como en la expresión "*la* ontología *de Aristóteles*", y que también admite su formulación en plural: "*hemos diseñado dos* ontologías *para clasificar estos elementos*".

Por otro lado, sobre este último uso del término existe cierto consenso en que puede ser entendido de dos formas [Evermann 2005]:

1. En el ámbito de la Filosofía, una ontología hace referencia a un sistema de categorías que explica qué es y cómo es la realidad desde un punto de vista particular. En este sentido, la ontología tiene un carácter universal y no depende

del lenguaje en que se formaliza: por ejemplo, la ontología de Aristóteles es siempre la misma independientemente del lenguaje utilizado para describirla.

2. Por analogía, en Informática se emplea el término *ontología* para designar a aquellas especificaciones formales acerca de las entidades que existen en un dominio concreto y cómo se relacionan entre ellas, esto es, cómo *son* dichas entidades en ese entorno. En este contexto, las ontologías se crean conforme a las necesidades que existan en un dominio particular, y por tanto, no son universales, sino que pueden cambiarse, adaptarse o personalizarse para propósitos concretos. Un ejemplo simple de ontología sería el de la descripción de una jerarquía de conceptos relacionados únicamente por relaciones de subsunción; ejemplos más complejos serían aquellos donde se utilizaran axiomas adicionales para expresar otras relaciones entre los conceptos y, de esta forma, perfilar las posibles interpretaciones.

Como puede observarse, existen diferencias entre la noción filosófica de ontología, y la usada en Informática. No obstante, ambas comparten el hecho de hacer referencia a *aquello que existe* en un dominio particular de aplicación [Evermann 2005]. Aunque cualquiera de los dos enfoques justifica su aplicación al modelado conceptual de SIs, en este trabajo adoptaremos la segunda concepción del término por ser la más usada y, en última instancia, la más práctica[1].

1.1 Definiciones de *ontología*

Además de tratar de aclarar su sentido, los investigadores en técnicas de representación del conocimiento también han dedicado esfuerzos a concretar una definición del término *ontología* que recogiese adecuadamente su significado [Guarino 1997]. El resultado es el amplio conjunto de definiciones de *ontología* que podemos encontrar en la literatura, algunas incluso contradictorias [Noy 2001], y que han sido sucesivamente revisadas por otros, o por sus mismos autores.

A continuación recogemos las siguientes definiciones por encontrarse entre las más frecuentemente referenciadas en la bibliografía relacionada:

- "*An explicit specification of a conceptualization*" [Gruber 1993].

[1] Para profundizar en otros trabajos que conjugan la concepción filosófica del término con técnicas de modelado conceptual se pueden consultar los trabajos de los autores Evermann y Wand referenciados en la bibliografía de esta tesis. En particular, estos autores toman como referencia la ontología propuesta por Mario Bunge para caracterizar sistemas, tanto físicos como sociales [Bunge 1977][Bunge 1979].

- "*A (AI-) ontology is a theory of what entities can exist in the mind of a knowledgeable agent*" [Wielinga 1993].
- "*An ontology is an explicit, partial account of a conceptualization*" [Guarino 1995].
- "*An ontology for a body of knowledge concerning a particular task or domain describes a taxonomy of concepts for that task or domain that define the semantic interpretation of the knowledge*" [Alberts 1993].
- "*An explicit knowledge-level specification of a conceptualization, i.e. the set of distinctions that are meaningful to an agent. The conceptualization –and therefore the ontology– may be affected by the particular domain and the particular task it is intended for*" [van Heijst 1997][2].
- "*An explicit, partial account of the intended models of a logical language*" [Guarino 1997].
- "*A formal specification of a shared conceptualization*" [Borst 1997].
- "*A logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models*" [Guarino 1998].
- "*Ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge. They provide potential terms for describing our knowledge about the domain*" [Chandrasekaran 1999].
- "*A formal description of entities and their properties, relationships, constraints, behaviours. It provides a common terminology that captures key distinctions and is generic across many domains*" [Gruninger 2000].
- "*A formal explicit description of concepts in a domain of discourse (*classes *–sometimes called* concepts*–), properties of each concept describing various features and attributes of the concept (*slots *–sometimes called* roles *or* properties*–), and restrictions on slots (facets –sometimes called* role restrictions*–)*" [Noy 2001].

No es nuestro propósito argüir a favor o en contra de la validez de cada definición, algo que, por otro lado, está fuera del ámbito de esta tesis. De todas ellas, la más citada con

[2] En este mismo trabajo se proporcionan otras definiciones que destacan distintos aspectos particulares de las ontologías dependiendo del propósito con el que se utilizan.

diferencia[3] es la proporcionada por Tom Gruber: "*una especificación explícita de una conceptualización*" [Gruber 1993]. Sin embargo, al igual que otras, esta definición obliga a aclarar qué se entiende por *conceptualización*, lo que también es motivo de discusión científica. De hecho, la noción de conceptualización utilizada por Gruber –entendida como una estructura <**D**,**R**> donde **D** es un dominio y **R** un conjunto de relaciones relevantes en **D**, y a su vez, tomada de [Genesereth 1987]–, deja de lado el sentido *intensional* (al margen situaciones particulares) del término, algo considerado fundamental en la investigación sobre ontologías[4].

Otros autores también destacan el uso y consideración de las ontologías como *vocabularios* de representación [Chandrasekaran 1999]. En este sentido, una ontología equivale a una gramática de modelado conceptual [Kishore 2004]. En rigor, no es el vocabulario como tal el que convierte al conjunto de términos que contiene en un ontología, sino el conjunto de significados (la *conceptualización*) que pretenden representar [Chandrasekaran 1999], o, dicho de otro modo, su *semántica* [Wand 1995].

En general, podemos decir que las distintas definiciones proveen diferentes, y a la vez complementarios, puntos de vista de lo que es una ontología. Algunos autores proporcionan definiciones de *ontología* al margen de su proceso de construcción o su utilización en aplicaciones concretas. En cambio, otros autores proporcionan definiciones influenciados por su proceso de desarrollo. En cualquier caso, podemos decir que las ontologías hacen referencia a un tipo de conocimiento *consensuado*, concebido para ser compartido y reutilizado por aplicaciones software y grupos de personas [Gómez-Pérez 2004].

Para el propósito de esta tesis hemos adoptado como referencia la última definición de las presentadas [Noy 2001], que a su vez, también está basada en los trabajos de Gruber [Gruber 1993]. De aquí en adelante, todas las alusiones a ontologías estarán formuladas considerando dicha concepción. Según esta definición una ontología consiste en una descripción formal y explícita de conceptos en un dominio de discurso, junto con la definición de relaciones entre ellos. Así, los elementos fundamentales de una ontología incluyen:

1. Clases o conceptos;

---

[3] 3627 citas en Google Scholar™, a 6-10-2008, al artículo donde aparece recogida la definición.

[4] Para profundizar más en detalles acerca de esta cuestión remitimos al lector a los trabajos de Guarino y Giaretta [Guarino 1995][Guarino 1998].

2. Propiedades de dichos conceptos, que describen diversas características o atributos de los mismos, y denominados *slots* –ranuras– (también se pueden encontrar como *roles*[5] o *propiedades*);

3. Restricciones sobre dichos *slots*, denominadas *facetas* –en inglés *facets*–, o *restricciones de roles*.

El conjunto formado por una ontología y el conjunto de instancias individuales de las clases de dicha ontología constituyen una *base de conocimiento*.

Una vez clarificado en qué consiste una ontología, en la siguiente sección vamos a tratar de justificar su uso en el modelado conceptual de Sistemas de Información (SI) y, en particular, en el modelado conceptual de sistemas cooperativos.

2. APLICACIONES

Como hemos visto, existen diferentes puntos de vista desde los que se puede explicar en qué consisten las ontologías o para qué se utilizan. Esta amplia diversidad de definiciones no impide que haya cierto consenso acerca de su efectividad en el desarrollo de SI, o que puedan ser vistas como algo más que una forma elegante de denominar el resultado del análisis y modelado de un dominio mediante alguna de las metodologías más comunes [Guarino 1998]. Al contrario, el análisis ontológico de un dominio clarifica la estructura del conocimiento acerca del mismo. Las ontologías obtenidas como resultado de este análisis permiten contar con un vocabulario preciso para representar el conocimiento. De ahí que el punto de partida para diseñar un SI, y dentro de éste, su sistema de representación del conocimiento junto con el vocabulario para describirlo, debe consistir en llevar a cabo un análisis ontológico suficientemente comprehensivo del dominio que se esté abordando. Por otro lado, un análisis ontológico pobre suele llevar a bases de conocimiento inconsistentes [Chandrasekaran 1999].

Entre los motivos que justifican el uso de ontologías en el modelado de SI podemos destacar de forma resumida los siguientes:

- *Compartir un entendimiento común acerca de la estructura y semántica de la información entre personas o agentes software.* A modo de ejemplo, podemos pensar en diferentes sitios Web que contienen información acerca de diversos sectores industriales de un país y de los servicios que proporcionan las empresas ubicadas en ellos. Si estos sitios Web comparten y hacen pública su información representada mediante una ontología de términos comunes a todos ellos, es posible construir agentes software

[5] A veces el término *rol* se considera sinónimo de relación binaria.

que extraigan y agreguen información procedente de cada uno de esos sitios Web diferentes. La información agregada de esta forma puede ser utilizada por los mismos u otros agentes para responder a consultas desde otras aplicaciones de análisis.

- *Habilitar la reutilización de conocimiento del dominio*. Junto con el anterior, éste probablemente sea uno de los motivos que han hecho resurgir la investigación en ontologías. La reutilización se da sobre todo entre representaciones de conceptos generales que pueden ser utilizadas en modelos para dominios diferentes. Por ejemplo, muchos sistemas tienen que manejar cuestiones relacionadas con la ocurrencia de eventos. Si un grupo de desarrolladores elabora una ontología detallada sobre los mismos (qué lo provoca, qué servicio lo maneja, si desencadena otros eventos, etc.), otros desarrolladores podrán simplemente reutilizar el modelo de eventos de dicha ontología para su propio sistema en otro dominio. La reutilización puede incluir la tarea de integrar más de una ontología o extender alguna ya existente.

- *Hacer explícitas las premisas acerca de un dominio*. A veces estas premisas están fijadas en el código de un programa escrito en un lenguaje de programación concreto, lo que las hace no sólo difíciles de identificar y entender, sino también difíciles de cambiar, especialmente para aquellos sin destrezas de programación. Por otro lado, explicitando las especificaciones acerca de un dominio se facilita su modificación y evolución cuando el conocimiento sobre el dominio cambia. Asimismo, de esta forma se facilita a los usuarios noveles el aprendizaje de los términos del dominio y su significado.

- *Separar el conocimiento del dominio del conocimiento operacional*. Mediante esta distinción es posible separar la descripción de una tarea de los artefactos utilizados para llevarla a cabo. Por ejemplo, una tarea de montaje de componentes en vehículos puede definirse independientemente de la descripción de los componentes en sí. De esta manera, podríamos seguir un mismo proceso de montaje de ciertos componentes para vehículos diferentes.

- *Analizar formalmente el conocimiento del dominio*. Ello sólo es posible cuando se dispone de una especificación declarativa de los términos utilizados para representar el conocimiento. El análisis formal de dichos términos resulta de gran utilidad cuando se persigue reutilizar o extender otras ontologías.

A partir de las razones que acabamos de presentar para desarrollar una ontología, es fácil ver cómo cada una de ellas contribuye, en mayor o menor medida, a facilitar los procesos de colaboración entre personas y entre agentes en un sistema colaborativo. En

efecto, el entendimiento común y compartido, y la reutilización de conocimiento están relacionados con la comunicación de información, pilar fundamental en todo proceso colaborativo. Relacionado con esto, podemos decir que haciendo explícito el conocimiento, lo comunicamos también. Asimismo, la distinción entre conocimiento del dominio y operacional, ayuda a reutilizar no sólo descripciones de objetos en un dominio, sino también los procesos que tienen lugar en él y, de esta forma, se amplían las posibilidades de interoperar a nivel estructural y/o de comportamiento. Por último, la descripción formal del conocimiento disminuye su ambigüedad y facilita su interpretación, lo que repercute en una mejor coordinación entre los agentes cooperantes.

Por otro lado, el desarrollo de una ontología acerca de un dominio suele concebirse como un medio para alcanzar otros objetivos como, por ejemplo, que otros programas puedan utilizar la información que contienen como datos de entrada para posteriormente procesarlos y relacionarlos, y de este modo, producir una salida más elaborada.

La utilidad que se vaya a dar a una ontología condiciona también su grado de precisión o generalidad en la descripción de los conceptos que maneja. A continuación se presentan distintos tipos de ontologías que algunos autores han identificado al desarrollar SI [Noy 2001].

3. TIPOS DE ONTOLOGÍAS

Dependiendo del punto de vista de la gestión del conocimiento que se adopte, se pueden diferenciar múltiples tipos de ontologías. El objetivo de esta sección es presentar algunas de las caracterizaciones de ontologías más frecuentes en la literatura relacionada, así como de los términos utilizados al clasificarlas, y que se relacionan a continuación.

Ontologías finas y gruesas. La caracterización más simple es aquella que diferencia entre ontologías *finas* y *gruesas*. Una ontología *fina* consta de un rico conjunto de axiomas y relaciones conceptuales de relevancia que permiten definir con bastante precisión los conceptos a los que se refiere. Por su parte, una ontología *gruesa* consiste en un conjunto reducido de axiomas formulados en un lenguaje de limitada capacidad expresiva. El objetivo al desarrollar una ontología gruesa es proporcionar unos pocos servicios que puedan ser compartidos por una amplia comunidad de usuarios que ya han aceptado los conceptos y relaciones de la ontología. Las ontologías finas, a veces son consideradas *ontologías de referencia* por su precisión y nivel de detalle, mientras que las ontologías gruesas se conocen también como *ontologías compartibles* [Guarino 1998].

Espectro de ontologías según el nivel de detalle en su especificación. Ampliando la clasificación anterior, Lassila y McGuinness proponen una clasificación de las ontologías en

función de la riqueza de su estructura interna [Lassila 2001]. Bajo esta clasificación, los distintos tipos de ontologías pueden visualizarse dispuestos en un *espectro* o *rango* lineal como el representado en la Figura III.1. Así, las ontologías se clasifican según definan:

- *Vocabularios controlados*, esto es, una lista finita de términos, como puede ser un *catálago*.
- *Glosarios*, es decir, una lista de términos junto con su significado. El significado de los términos se especifica por medio de enunciados en lenguaje natural.
- *Tesauros*, que proveen información adicional acerca de las relaciones entre términos, como las de sinonimia.
- *Jerarquías de términos informales*. Esta categoría se introdujo para hacer referencia a ciertas ontologías que se pueden encontrar en la Web, y en las que las relaciones *es-un* (*is-a* en inglés) no son definidas en sentido estricto. Por ejemplo, en ciertos casos, las instancias de descendientes de una clase más específica, no son necesariamente instancias de una clase más general a dicha clase específica, de tal manera que la transitividad de la relación no siempre se cumpliría.
- *Jerarquías de subclases formales* o *estrictas*. En este caso, si A es una superclase de B, entonces cualquier instancia de B es necesariamente una instancia de A. Las jerarquías de subclases estrictas se utilizan para explotar las propiedades de herencia.
- *Jerarquías de subclases formales y relaciones*. Estas ontologías incluyen también relaciones entre conceptos para describir un dominio concreto.
- *Marcos* (*frames*). En estas ontologías las instancias de una clase pueden contener información acerca de sus propiedades. Estas propiedades serán heredadas por las instancias de las subclases de dicha clase.
- *Restricciones de valor*. En estas ontologías se pueden definir restricciones acerca de qué valores pueden tomar las propiedades de un objeto de una clase.
- *Restricciones lógicas con carácter general*. Estas son las ontologías más expresivas. Con ellas, los ontólogos pueden especificar restricciones entre términos en lógica de primer orden.

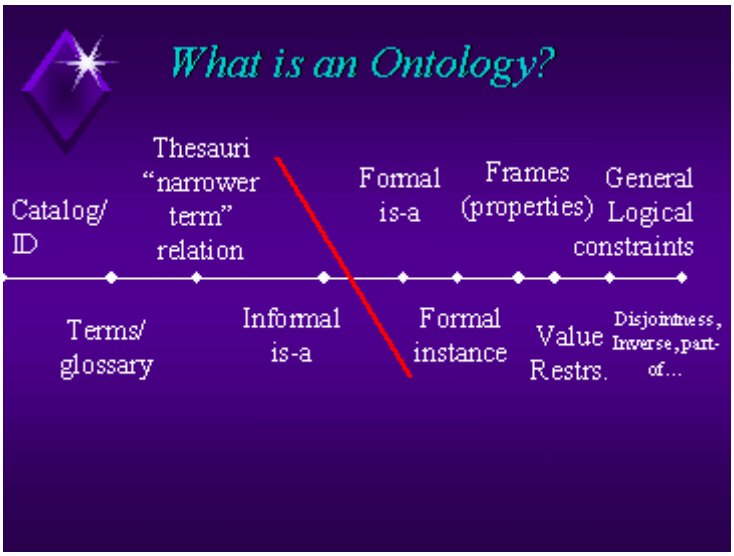


Figura III.1 Espectro de ontologías (fuente [Lassila 2001])

Según el nivel de generalidad y/o el objeto de la ontología. Atendiendo al nivel de generalidad[6], Guarino distingue cuatro tipos de ontologías, representados en la Figura III.2 [Guarino 1998]. Por su parte, Van Heijst junto a otros autores [van Heijst 1997], propone una clasificación muy similar, estableciendo como criterio el *tema objeto* (también *tópico* o simplemente *objeto*) de los conceptos modelados en la ontología. Guarino también identifica este criterio con la clasificación de las ontologías en función de "*su nivel de dependencia de una tarea o punto de vista particular*". Los cuatro tipos diferenciados bajo este criterio son los siguientes:

- *Ontologías de alto nivel.* Describen conceptos generales, como espacio, tiempo, materia, objeto, etc., que normalmente son independientes de un dominio o problema particular.
- *Ontologías de dominio.* Proporcionan un vocabulario genérico para describir conceptos, y relaciones entre ellos, de un dominio específico, como el de la medicina, la automoción, el derecho, etc. Los conceptos de una ontología de dominio normalmente son especializaciones de conceptos que aparecen en ontologías de alto nivel. Lo mismo ocurre con las relaciones entre ellos.
- *Ontologías de tareas.* Describen el vocabulario relativo a una tarea genérica o actividad, como por ejemplo, diagnosticar, planificar, vender, etc. Para ello, al igual que ocurría con las ontologías de dominio, se especializan los términos introducidos en ontologías de alto nivel.

[6] Entendida ésta como la capacidad de algo para ser aplicado a distintos dominios.

- *Ontologías de aplicación*. Describen conceptos que dependen de dominios y tareas particulares, y que son especializaciones de ambos tipos de ontologías relacionadas, vistas en los apartados anteriores. Con frecuencia, estos conceptos corresponden a roles desempeñados por las entidades de un dominio al realizar ciertas tareas.

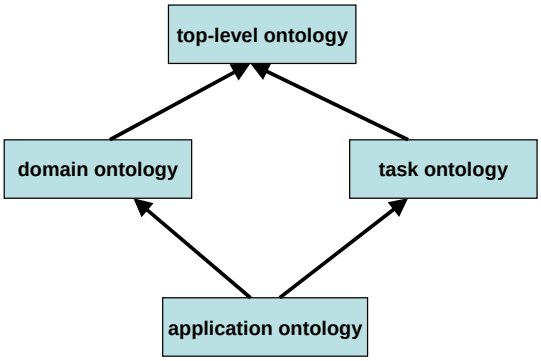


Figura III.2 Tipos de ontologías según su nivel de generalidad. Las flechas representan relaciones de especialización (fuente [Guarino 1998]).

4. PRINCIPIOS Y MÉTODOS DE DESARROLLO DE ONTOLOGÍAS

Una vez revisadas distintas formas de concebir y clasificar las ontologías, vamos a centrarnos en ciertos aspectos prácticos que tienen que ver con el desarrollo, lo que trataremos en esta sección, y las posibles aplicaciones de las ontologías, que abordaremos en la sección siguiente.

Por lo que respecta al desarrollo de una ontología, conviene tener presente que no existe la forma o metodología *correcta* para ello [Noy 2001]. Con carácter general, podemos decir que desarrollar una ontología incluye las siguientes actividades:

- Definir las clases de la ontología.
- Organizar las clases en una jerarquía taxonómica (subclase-superclase).
- Definir las propiedades (o *slots*), así como describir los valores permitidos para dichas propiedades.
- Definir instancias individuales de las clases de la ontología y asignar valores a las propiedades de estas instancias.

Ante cualquier proceso de desarrollo se deben considerar también los siguientes principios fundamentales:

- *No hay un único modo de modelar un dominio*, sino que siempre puede haber distintas alternativas. En la mayoría de los casos, la mejor solución dependerá del sistema

concreto que vaya a hacer uso de la ontología y de las extensiones a la misma que se puedan anticipar.

- *El desarrollo de una ontología es necesariamente un proceso iterativo.* Al igual que en cualquier otro ámbito de modelado, durante el proceso de desarrollo es normal que se propongan cambios que obliguen a rediseñar las propuestas iniciales.

- *Los conceptos de una ontología deben estar próximos a los objetos –físicos o lógicos–, que podamos encontrar en nuestro dominio de interés.* Con frecuencia los conceptos corresponderán a sustantivos, y las relaciones a los verbos que utilicemos para describir una realidad concreta.

Dependiendo del uso al que vayamos a destinar la ontología, trataremos de determinar el diseño que mejor se adapte a la funcionalidad proyectada, que sea más intuitivo, extensible, fácil de mantener, etc. [Noy 2001].

Por otro lado, durante el proceso de construcción de una ontología se entremezclan *aspectos metodológicos* y *aspectos arquitectónicos*. Los aspectos metodológicos siguen un enfoque interdisciplinar centrado en el análisis de la estructura del mundo real con un alto nivel de generalidad, y en la definición de un vocabulario riguroso y claro. Por su parte, los aspectos arquitectónicos colocan a la ontología en el centro del proceso de desarrollo de un SI, dando lugar al desarrollo de *Sistemas de Información Dirigido por Ontologías* (en inglés, *Ontology-driven Information Systems*).

4.1 Pasos en el desarrollo de una ontología

A continuación se va a presentar de forma resumida una sencilla metodología de desarrollo de ontologías, siguiendo en gran medida una propuesta elaborada por las autoras Noy y McGuiness [Noy 2001]. Los pasos propuestos son los siguientes:

1. *Determinar el dominio y el alcance de la ontología.* Básicamente, esta actividad comprende delimitar el dominio que va a abarcar la ontología, para qué se va a usar, para qué cuestiones debe proporcionar una respuesta la ontología y/o quién la utilizará y la mantendrá. Algunos de estos aspectos puede cambiar durante el diseño, pero conocerlos ayudará a acotar la ontología.

2. *Considerar reutilizar otras ontologías existentes.* En algunos casos, la reutilización puede contribuir a ahorrar trabajo a los desarrolladores, ya que, por ejemplo, se parte de decisiones de diseño que ya han sido estudiadas y adoptadas. La reutilización puede llegar a ser un requisito si nuestro sistema debe interactuar con otras aplicaciones que han adoptado alguna ontología particular. En este sentido, los sistemas de representación de conocimiento proveen herramientas que ayudan a exportar/importar otras ontologías.

3. *Enumerar los términos relevantes de la ontología*. Contar con una lista donde aparezcan los términos sobre los que se van a hacer declaraciones en la ontología, sus propiedades, relaciones, etc., contribuye a no olvidarlos y tener que "rescatarlos" en fases posteriores del proceso de desarrollo con un coste mayor. Esta lista de términos y relaciones puede elaborarse sin atender a consideraciones sobre si existen solapamientos entre unos y otros, o la oportunidad de modelar un concepto como una clase o una propiedad, que podrán perfilarse en fases posteriores.

4. *Definir las clases y la jerarquía de clases*. En este proceso se distinguen tres enfoques fundamentales [Uschold 1996]: *top-down* o "*de arriba hacia abajo*", en el que se define el concepto más general y, a partir de éste, todos los demás por especialización del mismo; *bottom-up* o "*de abajo hacia arriba*", al contrario que en el anterior, se comienza por definir los conceptos más específicos para posteriormente agruparlos en clases de conceptos más generales; *combinación*, en el que se combinan los dos enfoques anteriores, de tal forma que se comienza definiendo los conceptos más destacados del dominio para especializarlos y generalizarlos convenientemente.

   Ninguno de los enfoques anteriores se puede calificar como mejor que otro, algo que dependerá también de cada diseñador concreto. En cualquier caso, el proceso comienza con la definición de una serie de clases. Para ello, seleccionaremos preferentemente aquellos términos de la lista creada en el paso anterior que se refieran a objetos que tengan una existencia independiente, antes que términos que se utilicen para describir otros objetos. Las clases se organizarán en una taxonomía jerárquica, de tal forma que, si una clase *A* es subclase de otra clase *B*, entonces cada instancia de *A* es también una instancia de *B*.

5. *Definir las propiedades (también llamadas relaciones o* slots*) de las clases*. Por sí solas, las clases no contienen información suficiente para responder a las cuestiones que se plantearon en el paso 1. Por esta razón, es necesario describir la estructura de los conceptos por medio de *propiedades*. Una vez seleccionadas las clases en el paso anterior, la gran mayoría de los términos restantes de la lista elaborada en el paso 3, corresponderán a propiedades de dichas clases. Todas las subclases de una clase heredan los *slots* de esta última. Por esta razón, conviene asociar cada propiedad a la clase más general que pueda tener dicha propiedad.

6. *Definir facetas y/o restricciones sobre los* slots *o relaciones*. Los *slots* pueden presentar diferentes facetas que permiten definir el tipo al que deben pertenecen los valores de una propiedad, los valores permitidos o el número de dichos valores (conocido también como *cardinalidad del slot*). Por ejemplo, es normal definir una faceta que limite los valores de la propiedad *nombre* de una clase a valores de tipo cadena.

7. *Definir instancias*. La primera iteración del proceso de desarrollo concluye con la definición de instancias de las clases de la jerarquía. La definición de una instancia individual de una clase requiere, en primer lugar, seleccionar una clase, a continuación crear una instancia individual de esa clase, y por último, asignar valores a sus *slots*.

5. LENGUAJES DE IMPLEMENTACIÓN DE ONTOLOGÍAS

La forma de materializar las distintas cuestiones relativas al desarrollo (análisis, diseño e implementación) de una ontología que hemos visto hasta ahora, pasa por su especificación en algún lenguaje concreto de representación del conocimiento.

Como ejemplo de algunos de estos lenguajes de implementación de ontologías que han destacado por su aceptación encontramos:

- **CML** –(*Conceptual Modelling Language*) [Schreiber 1994b]–, es un lenguaje estructurado para la especificación de modelos de conocimiento de la metodología CommonKADS [Wielinga 1993][Schreiber 1994]. Las ontologías en este lenguaje pueden ser vistas como metamodelos que describen la estructura del conocimiento acerca del dominio. Asimismo, CML provee mecanismos para establecer correspondencias (*mappings*) entre distintos niveles de abstracción del conocimiento en una ontología, lo que facilita su construcción por capas, y donde las capas más altas contienen un tipo de conocimiento más abstracto.

  Esta característica es importante ya que, como se ha tratado anteriormente, las ontologías poseen su propia estructura y nivel de generalidad: algunas partes de una ontología pueden estar basadas en una teoría generalmente aceptada, mientras que otras pueden depender de la experiencia en un campo concreto de aplicación. La estructuración del conocimiento en una ontología determina de forma crítica su capacidad para ser compartida y/o reutilizada [Schreiber 1994b]. La Figura III.3 muestra un ejemplo con la definición en CML de una casilla de ajedrez. En él se puede apreciar cómo se ha restringido el valor de las columnas (*rank*) y las filas (*file*) para que tengan un valor comprendido entre 1 y 8.

```
CONCEPT chess-square;
  ATTRIBUTES:
    rank: INTEGER;
    file: INTEGER;

  AXIOMS:
    1 >= rank >= 8;
    1 >= file >= 8;
END CONCEPT chess-square;
```

Figura III.3 Ejemplo de definición de una casilla de ajedrez en CML

- **KIF** –(*Knowledge Interchange Language*) [Genesereth 1998]– es un lenguaje pensado para el intercambio de conocimiento entre agentes de computadoras (principalmente, aunque también es fácilmente comprensible para las personas), cada uno con el mismo o distinto sistema interno de representación de la información.

  KIF cuenta con una semántica declarativa, de tal forma que es posible entender el significado de las expresiones en este lenguaje, sin necesidad de utilizar un intérprete para manipular dichas expresiones. Asimismo, en KIF es posible expresar sentencias arbitrarias de cálculo de predicados, a diferencia también de lenguajes, como Prolog y SQL. Por último, este lenguaje proporciona mecanismos para representar conocimiento acerca del propio conocimiento, lo que permite hacer explícitas las decisiones sobre representación del conocimiento, así como introducir nuevos constructores para la representación del conocimiento sin cambiar el lenguaje. La Figura III.4 muestra un ejemplo de una especificación en este lenguaje que establece que toda persona debe tener una madre.

```
(forall (?x)
   (=> (person ?x)
      (exists (?y)
         (and (person ?y)
            (mother ?x ?y)
         )
      )
   )
)
```

Figura III.4 Ejemplo de especificación en KIF

- **Ontolingua** [Gruber 1993], fue publicado en 1992 por el Knowledge Systems Laboratory de la Universidad de Stanford. Se trata de un sistema de representación del conocimiento concebido para facilitar el diseño y especificación de ontologías mediante un lenguaje basado en KIF, pero con una semántica más lógica y clara.

Ontolingua extiende KIF con una sintaxis adicional para dar soporte a la construcción de módulos de ontologías que puedan ser ensamblados, extendidos o refinados en una nueva ontología. Se han desarrollado también diferentes traductores que permiten representar la especificación de una ontología en otros lenguajes como el propio KIF, el IDL[7] de CORBA, CLIPS, Prolog o Loom [Farquhar 1997].

- **Loom** [Loom 1995], fue concebido inicialmente como un lenguaje y un entorno para la construcción de sistemas inteligentes. Como ocurría con Ontolingua, consiste en un sistema de representación del conocimiento, a la vez que proporciona un lenguaje de representación con una sintaxis basada en KIF.

  El lenguaje de modelado Loom contiene un amplio conjunto de primitivas para la construcción declarativa de la representación de un modelo. Esto hace posible que gran parte de las aplicaciones Loom puedan expresarse también de forma declarativa, lo que a su vez, permite extenderlas, modificarlas y depurarlas más fácilmente. Un factor clave en el éxito del sistema Loom ha radicado en su capacidad para razonar de forma efectiva sobre estructuras de conocimiento declarativas en este lenguaje [MacGregor 1999]. En la Figura III.5 se muestra un ejemplo de una especificación en Loom donde se describe la clase *Persona-con-Dos-Hijos* como subclase de la clase *Persona-con-Hijos*.

```
(defconcept Person)
(defrelation has-child
   :domain Person :range Person)
(defconcept Male)
(defconcept Person-with-Sons
   :is (:and Person (:at-least 1 has-child Male)))
(defconcept Person-with-Two-Sons
   :is (and Person (:exactly 2 has-child Male)))
(tell (Person Fred))
(tell (has-child Fred Sandy))
(tell (Male Sandy))
```

Figura III.5 Ejemplo de especificación en Loom (fuente [MacGregor 1999])

No obstante, de entre todos los ámbitos de aplicación de las ontologías mencionados al comenzar el capítulo, el que más ha contribuido a su desarrollo ha sido el de la Web Semántica [Berners-Lee 2001]. Los lenguajes presentados hasta el momento comparten el

[7] IDL: Interface Definition Language.

hecho de que fueron diseñados previamente a la irrupción de Internet en las empresas y hogares, y con frecuencia se hace referencia a ellos (y a otros lenguajes contemporáneos con éstos), como *lenguajes de ontologías tradicionales* [Gómez-Pérez 2004]. Con objeto de aprovechar las posibilidades que brindan las tecnologías de Internet, a finales de los 90 y principios de la década de 2000, comenzaron a aparecer lenguajes de marcas para representar ontologías (p.e. XOL [Karp 1999], SHOE [Luke 2000], OIL [Fensel 2000], DAML+OIL [W3C DAML 2001], etc.), cuya sintaxis estaba basada en la de los lenguajes ya existentes HTML [W3C HTML 1999] y XML [W3C XML 2000]. Por esta razón, estos lenguajes se denominan también *lenguajes de ontologías basados en web* o *lenguajes de marcado de ontologías* [Gómez-Pérez 2004].

Enmarcados dentro de este último grupo de lenguajes y bajo el auspicio del W3C[8], se han publicado varias *recomendaciones* que contienen la definición de diferentes lenguajes estándares y modelos de datos, que permiten la especificación de la estructura de los contenidos –y su semántica–, en los documentos diseñados para la Web [W3C OWL 2004b]. Estos lenguajes, ordenados de menor a mayor capacidad expresiva, son los siguientes:

- **XML Schema** [W3C XML-S 2004]: es un lenguaje que permite definir la estructura de la información en documentos XML para que sea conforme a un esquema determinado. También extiende XML con la posibilidad de especificar tipos de datos. Sin embargo, la semántica acerca del contenido de la información que se puede expresar con este lenguaje es limitada, y queda restringida al ámbito del formato y tipo de los datos en un documento. Por este motivo, no se suele considerar como un lenguaje de ontologías en sentido propio.

- **RDF** (*Resource Description Framework*) [W3C RDF 2004]: es un modelo de datos para describir objetos (que se denominan *recursos*) y relaciones entre ellos, con una semántica simple. Más concretamente, RDF fue concebido como un marco de trabajo para representar metadatos acerca de recursos Web, tales como, el título, autor y fecha de modificación de una página Web, licencias de uso sobre documentos Web, o el horario en el que pudiera estar disponible un recurso compartido. Los modelos basados en RDF pueden representarse mediante un lenguaje con una sintaxis XML denominado RDF/XML.

- **RDF Schema** (o **RDF-S**) [W3C RDF 2004b]: es un vocabulario para describir propiedades y clases de recursos RDF. Cuenta con vocabulario y semántica para describir jerarquías de generalización de dichas propiedades y clases.

[8] W3C: World Wide Web Consortium, http://www.w3.org

- **OWL** (*Web Ontology Language*) [W3C OWL 2004]: amplia el vocabulario proporcionado por RDF Schema para describir propiedades (relaciones) y clases. Entre otras, podemos citar como ejemplo las siguientes: relaciones entre clases (p.e., disyunción), cardinalidad (p.e., "exactamente uno"), igualdad, tipificación más rica de propiedades, características de las relaciones (p.e., simetría, transitividad,…), y clases enumeradas.

Estos lenguajes comparten la característica común de haber sido diseñados para proporcionar una representación del conocimiento procesable automáticamente por computadoras y que facilite la comunicación y el intercambio de información entre ellas, y no tanto para ser empleadas como lenguaje de comunicación entre humanos.

En la Figura III.6 se muestra una clásica representación gráfica de estos lenguajes conocida como "*Pila de la Web Semántica*" (en inglés, *Semantic Web Stack*). En ella aparecen los lenguajes que acabamos de mencionar dispuestos según su poder expresivo y las especificaciones previas sobre las que se asientan, así como, otros lenguajes y tecnologías relacionados con la Web Semántica.

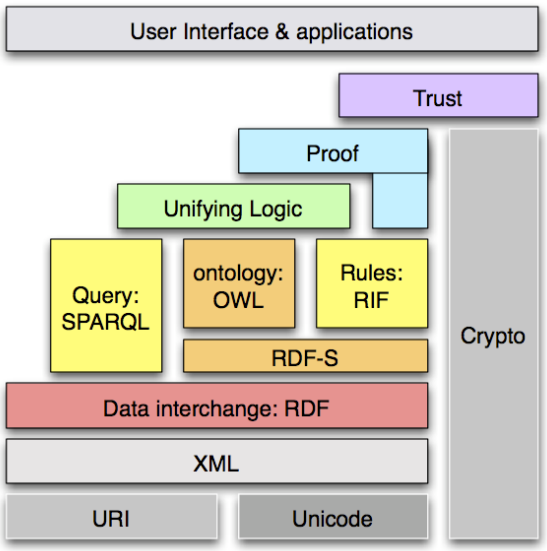


Figura III.6 Pila de tecnologías de la Web Semántica[9]

A continuación nos vamos a detener un poco más en detallar las características del lenguaje OWL, así como los fundamentos en los que se sustenta y que justificarán su elección como lenguaje de implementación de ontologías en el desarrollo posterior de esta tesis.

[9] Fuente: Steve Bratt, http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/Overview.html

5.1 El lenguaje OWL

Con carácter general, además de las ventajas que conlleva el haber sido concebidos específicamente para la Web Semántica (p.e., facilidad para el acceso remoto a datos, estructuración de la información, sintaxis precisa, etc.), los lenguajes propuestos por el W3C cuentan también con el bagaje previo de parte de sus creadores en el desarrollo de lenguajes de ontologías, principalmente el de los diseñadores del lenguaje DAML+OIL [W3C DAML 2001], considerado el precursor de OWL [W3C OWL 2004b]. Entre los científicos que habían participado anteriormente en la creación de otros lenguajes encontramos a: Guus Schreiber (CML para CommonKADS), Richard Fikes (KIF), Dieter Fensel (OIL), Ian Horrocks (OIL, DAML+OIL), Deborah McGuinness (DAML+OIL), etc. A ello se une el respaldo otorgado por el W3C, lo que contribuye a su mayor difusión e implantación en distintos tipos de SI.

Sin embargo, en lo que ser refiere a la especificación de ontologías, OWL sobresale por encima del resto (XML Schema, RDF y RDF Schema) al proveer un **vocabulario más rico** que ninguno y **estándar,** con una **semántica formal**, para expresar conceptos y relaciones entre ellos, y también características de dichas relaciones (simetría, transitividad, unicidad, etc.).

Esto trae consigo que, a diferencia de como ocurre con otros lenguajes basados en XML/Web, se pueda razonar sobre el conocimiento presente en un SI a partir de su representación. En efecto, muchos de los estándares industriales diseñados para la especificación de transacciones en la Web consisten en un conjunto de protocolos que definen el formato de los datos y el orden en que deben aparecer. Estos protocolos están dotados de una semántica operacional que define transacciones del tipo "un *Trabajador* percibe en la *CuentaDestino* una *Pensión* procedente de la *CuentaOrigen*, que es un *Porcentaje* de *IngresosDeducidos* en sus *AñosDeVidaLaboral*". Sin embargo, esta especificación no está diseñada para llevar a cabo tareas de razonamiento aparte del formato de los datos que figuran en ella. Por ejemplo, si un *Trabajador* ha nacido en un país extranjero (por tanto, se trata de un inmigrante), con el que existen convenios en materia de prestaciones por desempleo y jubilaciones, se podría deducir que habría que sumar a sus *AñosDeVidaLaboral*, los trabajados en su país de origen.

Por otro lado, para OWL existen diversas herramientas que permiten razonar sobre especificaciones en este lenguaje. Estas herramientas proporcionan un soporte de razonamiento *genérico*, en el sentido de que no es específico a un dominio particular, como podría ocurrir, por ejemplo, en el caso otros estándares basados en XML sobre los que se discutía en el párrafo anterior. Mientras que la construcción de un razonador requiere un esfuerzo considerable, la creación de una ontología es un proceso mucho más abordable.

De esta forma, los diseñadores de ontologías en OWL pueden beneficiarse de las propiedades formales de este lenguaje [W3C OWL 2004].

Por último, si se desean obtener razonamientos útiles de forma automática sobre el conocimiento representado en una ontología, se hace necesario contar un lenguaje que posea la capacidad de expresar una semántica más rica que la proporcionada por RDF-Schema.

Todo ello ha contribuido a que OWL sea considerado actualmente como un estándar de facto para la descripción de ontologías [Evermann 2008].

5.1.1 Sublenguajes de OWL

La especificación actual del lenguaje OWL definida por el W3C distingue entre los tres sublenguajes siguientes, ordenados de forma creciente según su potencia expresiva:

- **OWL Lite**, concebido para dar soporte a usuarios que principalmente necesiten poder especificar jerarquías de clasificación y restricciones simples. Por ejemplo, existe la posibilidad de definir restricciones de cardinalidad sobre las relaciones, pero restringida a tomar como valores 0 ó 1, únicamente. OWL Lite puede ser un lenguaje útil para representar tesauros o taxonomías reducidas.

- **OWL DL**, capaz de proporcionar la máxima potencia expresiva manteniendo la completitud computacional (se garantiza que todas las conclusiones sean computables), y la decidibilidad (todos los cálculos terminarán en un tiempo finito) de los procesos de deducción automática que pueda llevar a cabo un razonador diseñado para este lenguaje.

  OWL DL contiene todos los constructores propuestos para el lenguaje OWL, pero su uso está sujeto a ciertas restricciones para asegurar la propiedad de decidibilidad que acabamos de mencionar (en otras palabras, que se pueda garantizar que, según el estado de conocimiento actual, pueda existir un proceso de razonamiento decidible para un razonador basado en OWL DL). Así, entre otras limitaciones, se establece que los identificadores de clases, propiedades (también llamadas *slots*, roles o relaciones) e instancias han de ser disjuntos entre sí (esto implica, por ejemplo, que una *clase* no puede ser definida simultáneamente como una *instancia* de otra clase) [W3C OWL 2004c]. No obstante, algunas de estas limitaciones han comenzado a desaparecer parcialmente en las revisiones del lenguaje que se están llevando a cabo actualmente [W3C OWL 2 2008b].

  OWL DL debe su nombre a su semántica formal, basada en Lógica Descriptiva (*Description Logics* [Baader 2003], en adelante, DL), y sobre la que haremos una breve introducción en las secciones siguientes.

- **OWL Full**, pensado para usuarios que deseen la máxima potencia expresiva y la libertad sintáctica de RDF, pero sin garantías computacionales con vistas a procesos de razonamiento automático. Por ejemplo, en OWL Full una clase puede ser tratada simultáneamente como una colección de objetos (o *individuos*) y como un objeto en sí. Asimismo, OWL Full permite aumentar el significado del vocabulario predefinido (OWL o RDF) para este lenguaje. Como contrapartida, aparece el hecho de que es improbable que llegue a existir algún software de razonamiento que soporte todas las características de OWL Full.

El conjunto de constructores de OWL Lite está incluido en el de OWL DL (p.e. en OWL Lite una clase no se puede definir como la unión de otras clases, o como la clase complementaria de otra ya definida, algo que sí es posible en OWL DL). Por su parte, OWL DL y OWL Full usan los mismos constructores, aunque como ya hemos mencionado, en el caso de OWL DL se deben respetar ciertas condiciones al usarlos. A grandes rasgos, OWL DL requiere una clara separación de tipos, lo que significa que una clase no puede tratarse al mismo tiempo como una propiedad o un objeto, algo que sí está permitido en OWL Full.

Por otro lado, OWL en conjunto está diseñado para una máxima compatibilidad con RDF y RDF-Schema. Así, OWL Full puede verse como una extensión de RDF, mientras que OWL Lite y OWL DL pueden verse como extensiones de una vista restringida de RDF. Todo documento OWL (Lite, DL o Full) es un documento RDF, y todo documento RDF es un documento OWL Full.

La propiedad de permitir llevar a cabo procesos de razonamiento automáticos, junto a un vocabulario predefinido con una semántica rica para expresar conceptos y relaciones entre ellos, convierten a OWL DL en el sublenguaje preferido por los investigadores y desarrolladores para el modelado conceptual de un dominio mediante ontologías. En efecto, como ya veíamos en el primer capítulo, un conjunto de constructores rico permite recoger mejor los aspectos relevantes de un dominio [Wand 1995]. Por otro lado, razonar sobre una representación del modelo de dominio puede ayudar a guiar el proceso de modelado, así como a descubrir inconsistencias o conocimiento no declarado explícitamente [Parsia 2005]. Otro valor añadido, extrínseco al lenguaje, pero de considerable importancia [Sure 2005], es el amplio soporte en cuanto a herramientas para manipular las especificaciones en este lenguaje.

Por tanto, parece que OWL DL es la opción más adecuada para obtener un modelo conceptual de alto nivel acerca de un dominio, y a la vez permitir inferir nuevo conocimiento a partir del declarado explícitamente en una base de conocimiento (cfr. sección 1.1) [Baader 2003].

Si a todo lo anterior añadimos su clara orientación hacia la Web, la infraestructura tecnológica a través de la que colabora una gran mayoría de personas hoy día [Bourimi 2007], parece también justificado su aplicación al modelado de sistemas CSCW y los procesos colaborativos que tienen lugar en ellos, así como, para descubrir y gestionar consistentemente el conocimiento del que hacen uso. Además, conviene no olvidar que las recomendaciones para que la construcción (análisis, diseño e implementación) de la propia ontología también sea colaborativa [Farquhar 1997][Xexeo 2005].

Dada su relación con OWL DL, en lo que sigue, trataremos de explicar en qué consisten las **lógicas descriptivas** (DLs), algunos de los distintos tipos que hay de ellas y qué tipo es el que subyace a cada versión de OWL DL publicada hasta ahora.

5.2 Lógicas descriptivas

Con el término *Lógica Descriptiva* (en el pasado, *sistemas terminológicos* o *lenguajes de conceptos*) se hace referencia a un potente formalismo para la representación del conocimiento con una semántica formal bien definida para describir dominios de aplicación (el *mundo*) de forma estructurada, y en el que la prestación de servicios de razonamiento automático desempeña un papel central. Para ello, primero se definen los conceptos relevantes del dominio (su terminología), y posteriormente, estos conceptos son utilizados para especificar propiedades de objetos e individuos que tienen lugar en el dominio (la descripción del mundo).

Por lo que a los procesos de razonamiento se refiere, éstos se encaminan fundamentalmente a comprobar propiedades de los conceptos definidos en una terminología, así como deducir conocimiento implícito a partir de su definición. Algunas de las propiedades clásicas que podemos verificar son:

- *Satisfacibilidad* de un concepto: Un concepto es satisfacible en una terminología $\mathcal{T}$ si la definición de dicho concepto denota un conjunto de individuos que puede no ser vacío.
- Subsunción de conceptos: Un concepto D subsume a otro concepto C, expresado como $C \sqsubseteq D$, si D es considerado más general que C. En otras palabras, la subsunción comprueba si el conjunto denotado por C es un subconjunto del denotado por D.
- Consistencia de una base de conocimiento (conjunto formado por una terminología y las declaraciones acerca de las instancias individuales de los

conceptos que define): Una base de conocimiento es consistente si cada concepto admite la existencia de al menos una instancia individual.

5.2.1 Decidibilidad y complejidad computacional en DL

La semántica de las fórmulas usadas en las lógicas descriptivas se define estableciendo su equivalencia con expresiones en lógica de predicados de primer orden. Más concretamente, la semántica de los distintos tipos de conceptos representados mediante lógicas descriptivas se pueden expresar haciendo uso de diferentes *fragmentos* (o subconjuntos) de la lógica de primer orden para los que, a su vez, se pueden definir diferentes técnicas de razonamiento especializadas y variables en su nivel de complejidad [Baader 2003].

Puesto que la Lógica Descriptiva es un formalismo de representación del conocimiento, y como tal, se supone que dará respuesta a las consultas de un usuario en un tiempo razonable, los investigadores en procedimientos para razonadores en DL centran la mayor parte de su interés en los procesos de decisión; esto es, a diferencia de como ocurre, por ejemplo, con los demostradores de teoremas de lógica de primer orden, los procedimientos para DL deben terminar siempre, tanto para producir respuestas positivas como negativas. La garantía de respuesta en un tiempo finito no siempre implica que se produzca en un tiempo razonable. Por esta razón, la investigación de la complejidad computacional de una DL dada con problemas de inferencia decidibles es también una cuestión importante.

La decidibilidad y complejidad de los problemas de inferencia dependen del poder expresivo de cada DL particular. De nuevo se presenta la cuestión del equilibrio que el diseñador de un sistema debe buscar entre la potencia expresiva de un lenguaje y el coste de gestión del mismo. Por un lado, las DLs muy expresivas es probable que presenten problemas de inferencia de alta complejidad, o incluso sean no decidibles. Por el contrario, una DL demasiado débil (aun con procesos de razonamiento eficientes), puede ser insuficiente para expresar los conceptos más importantes de un dominio de aplicación (cfr. sección 1.1)[Baader 2003][Wand 1995].

A continuación veremos la nomenclatura, la sintaxis y semántica comúnmente utilizada para expresar fórmulas en DL, el tipo de conocimiento que se puede representar con cada una de ellas y su relación con las versiones de OWL DL.

5.2.2 Lenguajes de descripción

En lo que se refiere a la nomenclatura utilizada en DL, las descripciones más básicas corresponden a *conceptos atómicos* y *roles atómicos* (recordemos que los *roles* también se pueden llamar *propiedades* o *relaciones*). A partir de ellos, es posible construir de forma inductiva descripciones complejas mediante los *constructores de conceptos*. Por convención, en

notación abstracta se utilizan las letras *A* y *B* para los conceptos atómicos, la letra *R* para los roles atómicos, y las letras *C* y *D* para las descripciones de conceptos.

Las distintas versiones de DLs se distinguen por los constructores que proporcionan y que determinan su poder expresivo. En este contexto, $\mathcal{AL}$ (de *Attributive Language*) es un lenguaje de descripción que, a su vez, da nombre a una familia de lenguajes ampliamente estudiada en el ámbito de DL. Las descripciones de conceptos en $\mathcal{AL}$ se forman de acuerdo a las siguientes reglas sintácticas:

C, D	$\rightarrow$	$A \mid$	(concepto atómico)
		$\top \mid$	(concepto universal)
		$\bot \mid$	(concepto base)
		$\neg A \mid$	(negación atómica)
		$C \sqcap D \mid$	(intersección)
		$\forall R.C \mid$	(restricción de valores)
		$\exists R.\top \mid$	(cuantificación existencia limitada)

Como puede verse, en $\mathcal{AL}$ el operador de negación ($\neg$) sólo puede aplicarse a conceptos atómicos, y el ámbito del cuantificador existencial ($\exists$) se aplica siempre a todos los conceptos, es decir, no se puede especificar a qué clase deben pertenecer todos los elementos del rango de la propiedad a que se aplica (por eso aparece el concepto universal, $\top$, como su ámbito de aplicación).

Otros lenguajes de la familia $\mathcal{AL}$ tienen mayor capacidad expresiva. Cada lenguaje de esta familia representa una extensión al lenguaje básico $\mathcal{AL}$ y su nombre viene dado añadiendo una letra al final de la cadena $\mathcal{AL}$ indicando qué semántica se puede expresar ($\mathcal{AL}[\mathcal{C}][\mathcal{U}][\mathcal{R}][\ldots]$). A continuación se enumeran algunas de las letras utilizadas para denotar distintas capacidades expresivas:

- $\mathcal{C}$, negación de conceptos compleja, (o arbitraria, extendiendo la negación sobre conceptos atómicos que permitía $\mathcal{AL}$; p.e., para expresar el conjunto complementario de la unión de otros conceptos);
- $\mathcal{U}$, un concepto como resultado de la unión de otros dos (p.e., la clase *Humano* como la unión de *Hombres* y *Mujeres*);
- $\mathcal{R}$, axiomas complejos para definir propiedades (o *relaciones*, *roles*, *slots*; p.e., composición de propiedades, disyunción de propiedades, propiedades reflexivas e irreflexivas, y el complemento de una relación);
- $\mathcal{O}$, clases o conceptos, a partir de conjuntos enumerados (p.e., podría declararse la clase *Día_de_la_semana* como la formada por los elementos *lunes*, *martes*, etc.);
- $\mathcal{I}$, la propiedad inversa de otra dada (p.e., una propiedad *padre* puede especificarse como la inversa de la relación *hijo*);
- $\mathcal{Q}$, restricciones de cardinalidad *cualificadas* sobre propiedades (o relaciones, o roles), esto es, expresiones que permiten especificar la cardinalidad de una propiedad determinada y, además, la clase a la que deben pertenecen los valores que pueda tomar dicha propiedad (p.e., una persona puede *tener* ***dos*** progenitores biológicos, sin embargo, sólo *tendrá* ***una*** madre y ***un*** padre).

De esta forma, el lenguaje $\mathcal{ALCI}$, es un lenguaje de descripción que permite expresar la negación de un concepto cualquiera (por ejemplo, $\neg\forall R.C$), y también definir una propiedad como la inversa de otra.

Normalmente, la semántica de cada uno de los constructores del lenguaje se expresa mediante fórmulas en Lógica de Primer Orden (en adelante, FOL –*First-order Logic*–). Para ello, se define una *interpretación* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, donde $\Delta^{\mathcal{I}}$ es un *dominio de interpretación*, y $\cdot^{\mathcal{I}}$ es la *función de interpretación* que asigna a cada concepto C, un subconjunto $C^{\mathcal{I}}$ de $\Delta^{\mathcal{I}}$, y para cada propiedad R, una relación binaria $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

La Tabla III.1 resume la sintaxis en lenguaje de descripción y la semántica en FOL de algunos de los constructores utilizados en DLs, así como, la letra asociada que se emplea para denotar cierta capacidad expresiva.

No obstante, desde un punto de vista semántico estos lenguajes no siempre son distintos unos de otros. Por ejemplo, la semántica de los constructores hace que se den las equivalencias $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ y $\exists R.C \equiv \neg\forall R.\neg C$ (cuantificación existencial, denotada como $\mathcal{E}$), y por tanto, que el lenguaje $\mathcal{ALC}$ permita expresar los conceptos que se pueden expresar con $\mathcal{ALUE}$. Al a inversa, el lenguaje $\mathcal{ALUE}$ permite expresar conceptos semánticamente equivalentes a los de $\mathcal{ALC}$, trasladando las negaciones a los conceptos atómicos [Euzenat 2001]. En consecuencia, los lenguajes $\mathcal{ALC}$ y $\mathcal{ALUE}$ son semánticamente equivalentes, aunque normalmente se prefiere la forma $\mathcal{ALC}$. Por último, señalar que, con objeto de evitar nombres demasiado largos para los lenguajes de descripción más expresivos, se suele utilizar la abreviatura $\mathcal{S}$ para hacer referencia al lenguaje $\mathcal{ALC}_{\mathcal{R}+}$, esto es, una extensión al lenguaje $\mathcal{ALC}$ para expresar roles transitivos (denotado como $_{\mathcal{R}+}$).

Tabla III.1 Semántica de constructores de la familia de lenguajes $\mathcal{AL}$

Constructor	Sintaxis DL	Semántica en FOL	Letra Asociada
concepto atómico	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\mathcal{ALC}/\mathcal{S}$
concepto universal	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$	
propiedad atómica (o rol, en la jerga DL)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	
intersección	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
unión	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
complemento	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
restricción existencial	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}}$ **e** $y \in C^{\mathcal{I}}\}$	
restricción de valores	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}}$ **implica** $y \in C^{\mathcal{I}}\}$	
jerarquías de propiedades (roles)	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{H}$

acerca de las propiedades (roles)	axiomas de inclusión complejos	$R \sqsubseteq S$ $RS \sqsubseteq S$ $SR \sqsubseteq S$ $S_1 ... S_n \sqsubseteq R$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ $R^{\mathcal{I}} \circ S^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ $S^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ $S_1^{\mathcal{I}} \circ ... \circ S_n^{\mathcal{I}} \subseteq R^{\mathcal{I}}$	$\mathcal{R}$
	aserciones	Sym(R) Tra(R)	Sym(R)$^{\mathcal{I}}$ ***si*** $\langle x,y \rangle \in R^{\mathcal{I}}$ **implica** $\langle y,x \rangle \in R^{\mathcal{I}}$ Tra(R)$^{\mathcal{I}}$ ***si*** $\langle x,y \rangle \in R^{\mathcal{I}}$ y $\langle y,z \rangle \in R^{\mathcal{I}}$ **implica** $\langle x,z \rangle \in R^{\mathcal{I}}$	
		Ref(R) Irr(R) Dis(R,S)	Ref(R)$^{\mathcal{I}}$ ***si*** $\{\langle x,x \rangle \mid x \in \Delta^{\mathcal{I}}\} \subseteq R^{\mathcal{I}}$ Irr(R)$^{\mathcal{I}}$ ***si*** $\{\langle x,x \rangle \mid x \in \Delta^{\mathcal{I}}\} \cap R^{\mathcal{I}} = \varnothing$ Dis(R,S)$^{\mathcal{I}}$ ***si*** $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \varnothing$	
individuos (o nominales)		{o}	$\{o\}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, $\sharp\{o\}^{\mathcal{I}} = 1$	$\mathcal{O}$
propiedad inversa		R^{-}	$\{\langle x,y \rangle \mid \langle y,x \rangle \in R^{\mathcal{I}}\}$	$\mathcal{I}$
restricciones de cardinalidad no cualificadas		$\geqslant nR$ $\leqslant nR$	$\geqslant nR^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x,y \rangle \in R^{\mathcal{I}}\} \geqslant n\}$ $\leqslant nR^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x,y \rangle \in R^{\mathcal{I}}\} \leqslant n\}$	$\mathcal{N}$
restricciones de cardinalidad cualificadas[1]		$\geqslant nR.C$ $\leqslant nR.C$	$\geqslant nR.C^{\mathcal{I}} =$ $\{x \mid \sharp\{y.\langle x,y \rangle \in R^{\mathcal{I}}$ **e** $y \in C^{\mathcal{I}}\} \geqslant n\}$ $\leqslant nR.C^{\mathcal{I}} =$ $\{x \mid \sharp\{y.\langle x,y \rangle \in R^{\mathcal{I}}$ **e** $y \in C^{\mathcal{I}}\} \leqslant n\}$	$\mathcal{Q}$

El símbolo ♯ denota la cardinalidad de un conjunto

[1]Restricciones de cardinalidad cualificadas: además de la cardinalidad, se restringe la clase a la que deben pertenecer los elementos de la propiedad; se "cualifican".

5.2.3 Familias $\mathcal{AL}$ relacionadas con versiones de OWL DL

Como ya hemos mencionado, la semántica del lenguaje OWL DL está basada en Description Logics. En particular, la primera versión de este lenguaje (llamada OWL 1.0) estaba basada en la semántica de la lógica de descripción $\mathcal{SHOIN}(\mathbf{D})$, es decir, una extensión a $\mathcal{ALC}$ con soporte para expresar roles transitivos ($_{\mathcal{R}+}$), jerarquías de roles ($\mathcal{H}$), clases a partir de conceptos enumerados ($\mathcal{O}$), restricciones de cardinalidad ($\mathcal{N}$) y un tipo especial de roles o propiedades que relacionan instancias de clases con valores literales (cadenas que representan valores concretos, como por ejemplo, "true") o pertenecientes a algún de tipo de dato (**D**, *datatype*).

$\mathcal{SHOIN}(\mathbf{D})$, a pesar de estar considerada como una lógica de descripción con una potencia expresiva notable, presenta algunas limitaciones para satisfacer las demandas de los usuarios de OWL. El diseño original de este lenguaje siguió un enfoque algo conservador para asegurar la decidibilidad de los procesos de razonamiento sobre especificaciones en este lenguaje, así como, la existencia de soporte para ellos. A partir de su publicación en febrero de 2004 como estándar del W3C, el uso extensivo del lenguaje OWL (sobre todo en su versión DL), y la celebración de un congreso anual[10] en torno al mismo, ha dado como fruto la identificación de una serie de constructores útiles con los que extender OWL DL a un coste razonable. El desarrollo y avance en paralelo de la teoría de DL también ha proporcionado las bases para llevar a cabo razonamiento con estos constructores que no formaban parte del estándar inicial de OWL DL [W3C OWL 1.1 2007].

El resultado de este proceso está dando lugar a la especificación del lenguaje OWL 2 (anteriormente conocido como OWL 1.1 [W3C OWL 1.1 2007]). Esta nueva versión se encuentra en proceso de revisión, por lo que desde el W3C aún no se ha respaldado ninguna especificación del lenguaje [W3C OWL 2 2008]. OWL 2 extiende OWL DL con nuevos elementos, como:

- constructores adicionales para expresar, por ejemplo, restricciones de cardinalidad cualificadas sobre las propiedades (ver sección 5.2.2), composición de propiedades y características de las propiedades (reflexiva, irreflexiva, simétrica o asimétrica);
- *azúcar sintáctico* para facilitar la expresión de algunos conceptos complejos;

[10] http://www.webont.org/owled/

- extensión del soporte para tipos de datos (p.e., tipos de datos definidos por el usuario a partir de otros tipos de datos primitivos de XML-Schema como *integer* o *string*);
- nuevas anotaciones de ontologías y anotaciones de metamodelado para permitir *punning*[11], es decir, que una entidad de una ontología puedan comportarse como clase en un contexto, y como instancia en otro [Cuenca-Grau 2008].

La semántica subyacente a OWL 2 se corresponde de forma unívoca con la de los constructores de la lógica de descripción $\mathcal{SROIQ}$ [Horrocks 2006], exceptuando las pequeñas extensiones que permite el lenguaje para soportar tipos de datos (**D**), como ya ocurría con la primera versión de OWL, y *punnning*, que no están soportadas por $\mathcal{SROIQ}$ [W3C OWL 2 2008b].

6 INGENIERÍA DE SISTEMAS DIRIGIDO POR ONTOLOGÍAS

En la sección 4 de este capítulo veíamos que el papel central otorgado a las ontologías en el diseño de la arquitectura de un sistema conducía a la aparición de un nuevo tipo de sistemas conocidos como SI *dirigidos por ontologías*. Dentro de éstos aún se puede distinguir entre [Guarino 1998]:

- Sistemas en los que cada ontología se utiliza como un artefacto que guía su proceso de desarrollo estableciendo una clara descripción del dominio durante la fase de modelado conceptual, y en cuyo caso, sería más correcto hablar de *desarrollo de* SI *dirigido por ontologías*.
- SI dirigidos por ontologías en sentido propio, en los que la(s) ontología(s) es un componente más del SI al que se accede y puede ser modificado continuamente en tiempo de ejecución.

Ambos casos entroncan de forma natural con la evolución que han tenido posteriormente las recientes filosofías de desarrollo dirigidas por modelos, donde las ontologías pueden ser consideradas como una clase especial de modelos con una semántica formal basada en Lógica [W3C ODA 2006].

[11] punning: juego de palabras

6.1 OWL en los paradigmas de ingeniería dirigida por modelos (MDE)

En el ámbito del desarrollo de software, la irrupción en los últimos años del enfoque de desarrollo de *Ingeniería dirigida por Modelos* (*Model Driven Engineering*, MDE), ha venido a realzar el papel crucial que desempeña la actividad de modelado conceptual (ver Capítulo I) como medio para conectar y reflejar las demandas de los usuarios en los productos software finalmente desarrollados [Schmidt 2006]. Por otro lado, quienes ponen en entredicho algunos de los postulados defendidos por la filosofía MDE (por ejemplo, buena parte de la comunidad de desarrollo ágil), también coinciden en el valor y beneficios que reporta el uso de modelos conceptuales durante el ciclo de desarrollo del software [Miller 2004].

6.1.1 El enfoque MDA

El máximo exponente de aplicación del paradigma MDE es el enfoque de desarrollo de sistemas conocido como *Arquitectura dirigida por Modelos* (*Model Driven Architecture*, MDA), auspiciado por el Object Management Group[12] (OMG). MDA propone el uso de modelos de dominio donde se recoja la lógica de negocio de una organización, al margen de las plataformas tecnológicas concretas donde eventualmente se implementarán las funcionalidades requeridas. Este enfoque fomenta la autonomía de un modelo de negocio con respecto a las tecnologías empleadas para su implementación, y así, ambos pueden evolucionar de forma más independiente. Asimismo, se favorece la interoperabilidad dentro y a través de distintas plataformas tecnológicas [OMG 2003].

La realización con éxito del enfoque propuesto por la filosofía MDA pasa por establecer adecuadamente las correspondencias entre los modelos de sistema especificados en un nivel de abstracción y los modelos en un nivel de abstracción menor, de manera que se preserven los requisitos iniciales al transformar unos modelos en otros. En este punto, la semántica de los modelos cobra especial importancia. Así, poder contar con una semántica modelo-teórica formal, subyacente a un modelo permite aplicar razonadores automáticos para:

1. Comprobar la consistencia del modelo.
2. Determinar si dos modelos son compatibles (esta característica es importante para seguir un enfoque MDA en la integración de sistemas con vistas a que puedan interoperar entre ellos).

La especificación del estándar MDA adopta el lenguaje UML ([OMG 2007]) como lenguaje de modelado, cuya definición también está a cargo de OMG. Como ya vimos en el

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

capítulo I, UML se ha convertido en un estándar de facto para el modelado de sistemas software [Lange 2006], y su metamodelo viene expresado en términos del propio UML (en concreto, por medio de diagramas de clases). La semántica de los diagramas de clases está basada en la teoría de conjuntos, aunque no es del todo completa, ni formal, y por tanto, tampoco es confiable para procesos de razonamiento automáticos [OMG 2007c]. En particular, la *disyunción* o *intersección* de clases (conjuntos) no tienen una representación directa en UML, y su representación alternativa tampoco recoge adecuadamente la semántica de estos operadores. Por otro lado, en UML no existen constructores para expresar el *conjunto complementario*.

En las secciones anteriores (sección 5.1 y siguientes) acabamos de ver cómo estas limitaciones no están presentes en el lenguaje OWL, que sí cuenta con una semántica formal completa basada en la teoría de conjuntos y la potencia semántica para expresar de manera flexible conceptos como los que acabamos de mencionar: un concepto como la intersección de otros, el complementario, etc. Asimismo, OWL provee de un vocabulario más rico que UML para caracterizar las relaciones (p.e., se puede definir que una relación es *transitiva*), y desde la versión 2.0, también provee constructores para describir relaciones complejas (p.e., se puede especificar una relación como resultado de aplicar la función de composición sobre otras dos). Por otro lado, ciertas clases pueden definirse en función de ciertas propiedades de los individuos (p.e., la clase *Adulto* puede especificarse como aquella que comprende a las instancias de la clase *Persona* cuya edad es superior a un cierto número de años).

A partir de una ontología codificada en OWL, los razonadores pueden encontrar relaciones implícitas entre conceptos, conflictos entre ellos o conocimiento incompleto. Asimismo, puesto que OWL permite describir y trabajar con información incompleta, está especialmente indicado para las actividades de modelado de alto nivel de abstracción en las que aún no se tiene la certeza acerca de las cuáles serán las decisiones de diseño finales [W3C OWL 2 2008c]. Este aspecto se abordará con más detalle en el capítulo V.

Por el contrario, y a diferencia de como ocurre con UML, el principal inconveniente de OWL se encuentra en que la representación del conocimiento se realiza por medio de un lenguaje no visual o diagramático, ya que está orientada al intercambio de información entre computadoras antes que entre humanos. Por tanto, con frecuencia las descripciones en este lenguaje son difíciles de utilizar como herramienta de comunicación con ciertos *stakeholders*, sobre todo con aquellos no familiarizados con lenguajes declarativos y/o de marcas basados en XML [W3C OWL 2004b].

6.1.2 El *Metamodelo de Definición de Ontologías*

Por todo lo anterior, desde el propio OMG se ha advertido la relevancia que, para la plena realización de su propuesta MDA, puede tener el disponer de un substrato formal basado en ontologías para la representación, gestión, interoperabilidad y aplicación de semánticas de negocio, y salvar los inconvenientes del carácter semi-formal y ambiguo que presenta UML [W3C ODA 2006].

Éste ha sido el germen del Metamodelo de Definición de Ontologías (*Ontology Definition Metamodel*, ODM [OMG 2007c]), una especificación que recoge una familia de metamodelos y correspondencias entre ellos, que permiten interoperar con modelos especificados en UML y modelos en otros lenguajes formales como OWL o *Common Logic* (CL, [ISO 2007]), entre otros.

El ODM sienta las bases para, por ejemplo, poder obtener especificaciones en OWL a partir de modelos UML, y a la inversa, modelos UML a partir de ontologías en OWL. De esta forma, se habilita el desarrollo de ontologías tomando como punto de partida modelos de sistema especificados en UML [OMG 2007c]. En última instancia, la especificación del ODM persigue habilitar el modelado de ontologías mediante herramientas basadas en UML.

En consecuencia, un modelo basado en el ODM puede emplearse para:

- El intercambio de conocimiento entre sistemas heterogéneos.
- La representación del conocimiento de forma ontológica y en bases de conocimiento.
- La especificación de expresiones que sirvan de entrada o salida de motores de inferencia para lenguajes con una semántica formal (p.e., OWL) [Berardi 2005].

Aunque el ODM no se centra únicamente en establecer mapeos entre constructores de UML y constructores de OWL, ambos lenguajes y las tecnologías relacionadas ocupan un puesto central en la especificación del mismo. Por otra parte, conviene tener presente que la especificación también establece el carácter *informativo* (por contraposición a *normativo*), de los mapeos propuestos entre los distintos metamodelos. El objeto de esta medida es dejar más libertad a los analistas de un sistema para poder adaptar las correspondencias entre dos metamodelos a necesidades particulares del dominio donde se esté aplicando [OMG 2007c].

Además, el establecimiento de las correspondencias entre los elementos de cada metamodelo no está exento de problemas, sobre todo en aquellos casos donde no existe un vocabulario específico para expresar ciertos conceptos y para los que no es posible encontrar construcciones exactamente equivalentes desde un punto de vista semántico. Por

ejemplo, en OWL se establece que todas las relaciones se definen únicamente entre pares de conceptos, mientras que en UML es posible definir relaciones n-arias [Evermann 2008].

El ODM encierra los fundamentos de un enfoque para el desarrollo y despliegue de sistemas que ha dado en llamarse *Arquitectura Dirigida por Ontologías* (*Ontology Driven Architecture*, ODA). El objetivo de esta nueva filosofía es complementar las metodologías basadas en MDA con las ventajas que presentan los lenguajes formales de representación del conocimiento para la Web Semántica como OWL [W3C ODA 2006].

6.1.3 Ontology Driven Architecture

Como ya se ha introducido en la sección anterior, este enfoque híbrido combina técnicas de representación formal diseñadas para la Web Semántica, junto con prácticas comúnmente aceptadas en la Ingeniería del Software y Sistemas.

Sin embargo, la realización de la filosofía ODA en el desarrollo de los nuevos sistemas está reportando beneficios y explotando nuevas capacidades que van más allá de las inicialmente anticipadas. En efecto, los entornos de computación que promueven los nuevos paradigmas orientados a servicios, tales como, el *Service Oriented Architecture* (SOA), requieren la posibilidad de ofrecer, descubrir y reutilizar servicios de forma declarativa [OASIS 2006]. Esto es posible siempre que los metadatos acerca de los servicios (funcionalidad, incompatibilidades, precondiciones, etc.), sea lo suficientemente fidedigna y precisa.

En este punto es donde se pone de relieve el valor que pueden aportar los lenguajes para la Web Semántica como OWL. Una ontología de metadatos sobre servicios y proveedores de servicios puede ayudar a establecer relaciones entre ellos. A partir de ahí, es posible consultar la ontología para comprobar si dos servicios son compatibles, quién ofrece un servicio o anticipar la activación de terceros en función de ciertas condiciones. Además, se habilita la detección automática de configuraciones de sistema inconsistentes, tanto durante el desarrollo como en tiempo de ejecución [W3C ODA 2006].

La Figura III.7 muestra un ejemplo de un servidor de aplicaciones basado en ODA. En la parte izquierda se han representado posibles fuentes de datos para el sistema: ficheros de configuración de servidores ("web.xml"), ficheros con información de despliegue ("ejb-jar.xml"), etc. Este conjunto de metadatos semánticos y la ontología del sistema se cargan en un motor de inferencia ("*inference engine*"). Los distintos servicios y aplicaciones hacen uso de las capacidades de razonamiento incorporadas al servidor de aplicaciones en tiempo de desarrollo y en tiempo de ejecución.

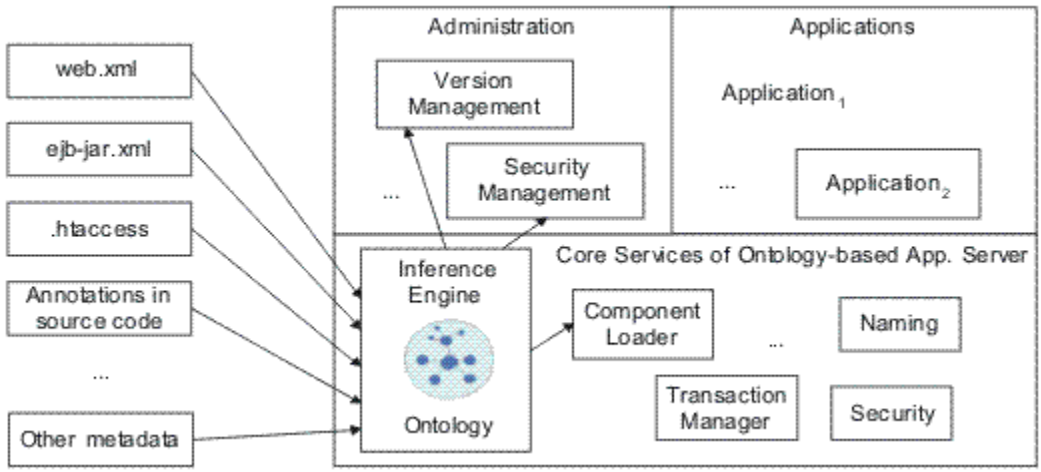


Figura III.7 Arquitectura del sistema de un servidor basado en ontologías. Fuente [W3C ODA 2006]

Como consecuencia, tenemos que el desarrollo de software pasa de ser una actividad basada en el diseño e implementación de nuevas funciones, a una actividad consistente en identificar y reunir funciones ya existentes para lograr una solución y que se conoce como orquestar y coreografiar [Peltz 2003]. Al mismo tiempo, ODA da respuesta a la necesidad de incrementar el nivel de abstracción y automatizar los procesos de Ingeniería del Software derivados de la creciente complejidad de los sistemas en términos de escalabilidad, distribución, interoperabilidad, etc.

Finalmente, como se señalaba al final de la sección 5.1.1, las tecnologías de la Web Semántica aportan una infraestructura fundamental: la Web misma. Si en dicha sección se destacaba el papel de la Web como escenario donde las personas se *encuentran* para colaborar, ahora queremos enfatizar el hecho de que también a nivel tecnológico (ver sección 2.1 y figura II.1 del capítulo II), es el entorno donde agentes basados en ontologías pueden interoperar y colaborar intercambiando información sobre servicios. A ello hay que sumarle el hecho de que buena parte de los sistemas SOA se implementa como servicios Web [OASIS 2006][W3C WSA 2004].

En los siguientes capítulos trataremos de ilustrar la aplicación del lenguaje OWL al desarrollo de sistemas colaborativos siguiendo un enfoque ODA, en el marco de la metodología AMENITIES (ver sección 3 del capítulo II).

REFERENCIAS DEL CAPÍTULO

[Alberts 1993] Alberts, L.K.: "YMIR: an Ontology for Engineering Design". Ph.D. Thesis, University of Twente, Twente, The Netherlands, 1993

[Baader 2003] Baader, F., Calvanese, D., McGuineness, D., Nardi, D., Patel-Schneider, P.: "The Description Logic Handbook". Cambridge University Press, 2003

[Berardi 2005] Berardi, D., Calvanese, D., De Giacomo, G.: "Reasoning on UML class diagrams". Artificial Intelligence 168 (1–2), pp. 70–118 (2005)

[Berners-Lee 2001] Berners-Lee, T., Hendler, J., Lassila, O.: "The Semantic Web," Scientific American, May 2001, pp. 34-43

[Borst 1997] Borst, W.N.: "Construction of Engineering Ontologies for Knowledge Sharing and Reuse". PhD thesis, Universiteit Twente, Enschede, The Netherlands (1997)

[Bourimi 2007] Bourimi, M., Lukosch, S., Kühnel, F.: "Leveraging Visual Tailoring and Synchronous Awareness in Web-Based Collaborative Systems". CRIWG 2007. LNCS 4715, Springer-Verlag, Berlin, 2007, pp. 40-55

[Bunge 1977] Bunge, M.A.: "Ontology I: The Furniture of the World. Volume 3 of Treatise on Basic Philosophy". D. Reidel Publishing Company, Dordrecht, Holland (1977)

[Bunge 1979] Bunge, M.A.: "Ontology II: A World of Systems. Volume 4 of Treatise on Basic Philosophy". D. Reidel Publishing Company, Dordrecht, Holland (1979)

[Chandrasekaran 1999] Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: "What are ontologies, and why do we need them?". Intelligent Systems and Their Applications, IEEE, vol.14, no.1, Jan/Feb 1999, pp. 20-26

[Cuenca-Grau 2008] Cuenca-Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: "OWL 2: The next step for OWL". Web Semantics: Science, Services and Agents on the World Wide Web, Volume 6, Issue 4, Semantic Web Challenge 2006/2007, November 2008, pp. 309-322

[Euzenat 2001] Euzenat, J.: "An infrastructure for formally ensuring interoperability in a heterogeneous semantic web". Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001, pp. 345-360

[Evermann 2005] Evermann, J., Wand, Y.: "Toward Formalizing Domain Modeling Semantics in Language Syntax". IEEE Transactions on Software Engineering, vol. 31, no. 1, January 2005, pp. 21-37

[Evermann 2008] Evermann, J.: "A UML and OWL Description of Bunge's Upper-level Ontology Model". Software and Systems Modeling. In press (doi:10.1007/s10270-008-0082-3)

[Farquhar 1997] Farquhar, A., Fikes, R., Rice, J.: "The Ontolingua Server: a Tool for Collaborative Ontology Construction". International Journal of Human-Computer Studies (IJHCS), 46(6): pp. 707–728, 1997

[Fensel 2000] Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., Klein, M.: "OIL in a nutshell". Proceedings of the European Knowledge Acquisition Conference (EKAW-2000). LNAI 1937, Springer-Verlag, 2000, pp. 1-16

[Genesereth 1987] Genesereth, M.R., Nilsson, N.J.: "Logical Foundations of Artificial Intelligence". San Mateo, CA: Morgan Kaufmann Publishers, 1987

[Genesereth 1998] Genesereth. M.: "Knowledge Interchanged Format (KIF)"; 1998 http://logic.stanford.edu/kif/kif.html

[Gómez-Pérez 2004] Gómez-Pérez, A., Ferández-López, M., Corcho, O.: "Ontological Engineering: with Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web". Springer-Verlag, London, 2004

[Gruber 1993] Gruber, T. R.: "A translation approach to portable ontology specifications". Knowledge Acquisition 5, 1993, pp. 199-220

[Gruninger 2000] Gruninger, M., Atefi, K., Fox, M.S.: "Ontologies to support process integration in enterprise engineering". Kluwer Academic Publishers 2001, Computational and Mathematical Organization Theory 6, 2000, pp. 381–394

[Guarino 1995] Guarino, N., Giaretta, P.: "Ontologies and Knowledge Bases: Towards a Terminological Clarification". In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam, 1995, pp. 25-32

[Guarino 1997] Guarino, N.: "Understanding, Building, and Using Ontologies: A Commentary to 'Using Explicit Ontologies in KBS Development', by van Heijst, Schreiber, and Wielinga". International Journal of Human and Computer Studies 46, 1997, pp. 293-310

[Guarino 1998] Guarino, N.: "Formal Ontology in Information systems". Proceedings of FOIS '98, (Trento, Italy, June, 1998). IOS Press, Amsterdam, 1998, pp. 3-15

[van Heijst 1997] van Heijst, G., Schreiber, A.T., Wielinga, B.J.: "Using explicit ontologies in KBS development". International Journal of Human and Computer Studies 46, 1997, 183-292

[Horrocks 2006] Horrocks, I., Kutz, O., Sattler, U.: "The even more irresistible SROIQ". In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2006, pp. 57-67

[ISO 2007] ISO/IEC FDIS 24707:2007(E) Information technology – Common Logic (Common Logic) – A framework for a family of logic-based languages. Available at http://cl.tamu.edu/

[Karp 1999] Karp, P.D., Chaudhri, V.K., Thomere, J.: "XOL: An XML-based ontology exchange language". Version 0.4, 1999. http://www.ai.sri.com/~pkarp/xol/xol.html

[Kishore 2004] Kishore, R., Zhang, H., Ramesh, R.: "A Helix-Spindle Model for Ontological Engineering". Communications of ACM, February 2004/Vol. 47, No. 2, pp. 69-75

[Lange 2006] Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: "UML Software Architecture and Design Description". IEEE Software, 23 (2006), pp. 40-46

[Lassila 2001] Lassila, O., McGuinness, D.: "The role of frame-based representation on the semantic web". Technical Report KSL-01-02. Knowledge System Laboratory, Stanford University, Stanford, CA, 2001

[Loom 1995] Tutorial for Loom version 2.1., http://www.isi.edu/isd/LOOM/documentation/tutorial2.1.html, May 1995

[Luke 2000] Luke, S., Heflin, J.D.: "SHOE 1.01. Proposed Specification". Technical Report. Parallel Understanding Systems Group. Department of Computer Science. University of Maryland. http://www.cs.umd.edu/projects/plus/SHOE/spec.html

[MacGregor 1999] MacGregor, R.: "Retrospective on LOOM". Technical Report. Information Sciences Institute. University of Southern California. http://www.isi.edu/isd/LOOM/papers/macgregor/Loom_Retrospective.html

[Miller 2004] Miller, G., Ambler, S., Cook, S., Mellor, S., Frank, K., Kern, J.: "Model driven architecture: the realities, a year later". In Companion To the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (Vancouver, BC, CANADA, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, pp. 138-140

[Noy 2001] Noy, N.F., McGuinness, D.L.: "Ontology development 101: A guide to creating your first ontology". Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001

[OASIS 2006] OASIS, "Reference Model for Service Oriented Architecture 1.0", Committee Specification 1, 2 August 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[OMG 2003] OMG, "MDA Guide Version 1.0.1". OMG Document Number: omg/03-06-01. Available at: http://www.omg.org/docs/omg/03-06-01.pdf

[OMG 2007] OMG, "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09. Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Parsia 2005] Parsia, B., Sirin, E., Kalyanpur, A.: "Debugging OWL ontologies". In Proceedings of the 14th international Conference on World Wide Web (Chiba, Japan, May 10 - 14, 2005). WWW '05. ACM, New York, NY, 633-640

[Peltz 2003] Peltz C.: "Web Services Orchestration and Choreography". IEEE Computer 36 (2003), pp. 46–52

[Schmidt 2006] Schmidt, D.C.: "Guest Editor's Introduction: Model-Driven Engineering". IEEE Computer, 39 (2006), pp. 25-31

[Schreiber 1994] Schreiber, G., Wielinga, B., de Hoog, R., Akkermans, H., Van de Velde, W.: "CommonKADS: A Comprehensive Methodology for KBS Development," IEEE Expert: Intelligent Systems and Their Applications, vol. 09, no. 6, pp. 28-37, Dec., 1994

[Schreiber 1994b] Schreiber, G., Wielinga, B., Akkermans, H., Van de Velde, W., Anjewierden, A.: "CML: The CommonKADS Conceptual Modelling Language". A Future for Knowledge Acquisition, 8th European Knowledge Acquisition Workshop. LNCS 867, Springer-Verlag, Berlin, 1994, pp. 1-25

[Sure 2005] Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., Oberle, D.: "The SWRC ontology - semantic web for research communities". In: Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Covilha, Portugal, 2005, pp. 218-231

[Uschold 1996] Uschold, M., Gruninger, M.: "Ontologies: Principles, Methods and Applications". Knowledge Engineering Review 11 (2), 1996, pp. 93-155

[Wand 1995] Wand, Y., Monarchi, D.E., Parsons, J., Woo, C.C.: "Theoretical foundations for conceptual modelling in information systems development", Decision Support Systems, Volume 15, Issue 4, December 1995, pp. 285-304

[Wielinga 1993] Wielinga, B.J., Schreiber, A.T.: "Reusable and sharable knowledge bases: a European perspective". In Proceedings of Proceedings of First International

Conference on Building and Sharing of Very Large - Scaled Knowledge Bases. Tokyo, Japan, 1993, pp. 103-115

[W3C DAML 2001] W3C, (Editores) Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A. (Editores): "DAML+OIL (March 2001) Reference Description". W3C Note, 18 December 2001. http://www.w3.org/TR/daml+oil-reference

[W3C HTML 1999] Raggett, D., le Hors, A., Jacobs, I.: "HTML 4.01 Specification". W3C Recommendation, 24 December 1999, http://www.w3.org/TR/html401/

[W3C ODA 2006] W3C, (Editores) Tetlon, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., Kendall, E.: "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering". W3C Editors' Draft of 11 February 2006, http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[W3C OWL 2004] W3C, (Editores) Smith, M.K., Welty, C., McGuinness, D.L.: "OWL Web Ontology Language: Guide". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/

[W3C OWL 2004b] W3C, (Editores) McGuinness, D.L., van Harmelen, F. (Editores): "OWL Web Ontology Language: Overview". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-features/

[W3C OWL 2004c] W3C, (Editores) Dean, M., Schreiber, G. (Editores): "OWL Web Ontology Language Reference". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-ref/

[W3C OWL 1.1 2007] W3C, (Editores) Patel-Schneider, P.F., Horrocks, I.: "OWL 1.1 Web Ontology Language: Overview". Editor's Draft of 23 May 2007, http://webont.org/owl/1.1/overview.html

[W3C OWL 2 2008] W3C, (Editores) Motik, B., Patel-Schneider, P.F., Parsia, B.: "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax". W3C Working Draft, 02 December 2008. http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/

[W3C OWL 2 2008b] W3C, (Editores) Motik, B., Patel-Schneider, P.F., Cuenca-Grau, B.: "OWL 2 Web Ontology Language: Direct Semantics". http://www.w3.org/TR/2008/WD-owl2-semantics-20081202/

[W3C OWL 2 2008c] W3C, (Editores) Parsia, B., Patel-Schneider, P.F.: "OWL 2 Web Ontology Language: Primer". W3C Working Draft, 11 April 2008. http://www.w3.org/TR/owl2-primer/

[W3C RDF 2004] W3C, (Editores) Manola, F., Miller, E. (Editores): "RDF Primer". W3C Recommendation, 10 February 2004. http://www.w3.org/TR/rdf-primer/

[W3C RDF 2004b] W3C, (Editores) Brickley, D., Guha, R.V. (Editores): "RDF Vocabulary Description Language 1.0: RDF Schema". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-schema/

[W3C WSA 2004] W3C, (Editores) Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, I.M., Ferris, C., Orchard, D.: "Web Services Architecture". W3C Working Group Note, 11 February 2004. http://www.w3.org/TR/ws-arch/

[W3C XML 2000] W3C, (Editores) Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: "Extensible Markup Language (XML) 1.0". W3C Recommendation, 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[W3C XML-S 2004] W3C, (Editores) Fallside, D.C., Walmsley, P. (Editores): "XML Schema Part 0: Primer". W3C Recommendation, 28 October 2004, http://www.w3.org/TR/xmlschema-0/

[Xexeo 2005] Xexeo, G., Vivacqua, A., De Souza, J.M., Braga, B., D'Almeida Jr., J.N., Almentero, B.K., Castilho, R., Miranda, B.: "COE: A collaborative ontology editor based on a peer-to-peer framework". (2005) Advanced Engineering Informatics, 19 (2), pp. 113-12

Capítulo IV

ANÁLISIS Y DISEÑO DE SISTEMAS COLABORATIVOS GUIADO POR ONTOLOGÍAS

1. INTRODUCCIÓN

Como resultado de la actividad de modelado conceptual (cfr. capítulo I), se puede afirmar que en un SI se contiene una representación del mundo tal y como este es percibido por sus desarrolladores (analistas, diseñadores y programadores) [Wand 1995]. Bajo esta premisa y tomando en cuenta las definiciones de ontología expresadas en el capítulo anterior, una ontología también constituye un modelo conceptual de un dominio.

Este punto de vista tampoco difiere del adoptado por la filosofía ODA (cfr. sección 6.1.3 en el capítulo III), donde las ontologías se consideran "*modelos conceptuales* explícitos *con semántica formal basada en lógica*" [W3C ODA 2006].

El método que presentamos a continuación aborda el desarrollo de sistemas colaborativos comenzando por la especificación de su arquitectura del sistema, y que a su vez, está en una descripción ontológica del entorno de trabajo en grupo. El objetivo es establecer los fundamentos que conduzcan a una propuesta para el *desarrollo de sistemas colaborativos guiado por ontologías* [Guarino 1998].

Para ello, se propone la formalización mediante ontologías de los modelos del marco de trabajo propuesto en AMENITIES (sección 3 del capítulo II). El lenguaje escogido para la implementación de las ontologías es OWL.

2. REPRESENTACIÓN ONTOLÓGICA DE SISTEMAS COLABORATIVOS

Al igual que ocurre con otras propuestas, la especificación de la metodología AMENITIES está organizada fundamentalmente en torno a modelos y vistas de

comportamiento que permiten describir el trabajo en grupo dentro de una organización o entre varias organizaciones [Garrido 2005].

Para la descripción de dichos modelos se hace uso de un vocabulario específico definido previamente, donde se recogen los principales conceptos que se suelen manejar al describir un proceso colaborativo (p.e., *rol*, *tarea*, *grupo*, etc.), así como, las relaciones entre ellos. En AMENITIES estos conceptos se describen en el *marco conceptual de trabajo* proporcionado por la metodología, y representado gráficamente de forma resumida y parcial por medio de un diagrama de clases de UML (cfr. sección 3.4.1 y Figura II.4 en capítulo II).

Si recordamos, esta concepción de marco conceptual se asemeja bastante a la definición de ontología que ya indicamos que usaríamos en este texto (cfr. sección 1.1 del capítulo III): "*una descripción* formal *de conceptos – y relaciones entre ellos –, en un dominio de discurso*" [Noy 2001]. La única diferencia la encontramos en el carácter formal que posee una ontología.

En AMENITIES los conceptos para describir un sistema colaborativo están descritos de manera semi-formal en UML y en lenguaje natural. Por tanto, el primer paso para obtener una representación ontológica de los modelos de comportamiento de la metodología consistirá en describir formalmente el vocabulario a utilizar mediante un lenguaje con una semántica modelo-teórica bien definida, como es el caso de OWL [Gruber 1993].

2.1 Formalización del marco conceptual de AMENITIES en OWL 2

Conviene tener presente el carácter general con el que fue diseñado el marco conceptual de AMENITIES, a fin de poder representar un amplio espectro de sistemas colaborativos. Por esta razón, la ontología resultante de su formalización constituirá una *ontología de dominio* [Guarino 1998], es decir, tendrá también un carácter general y las clases o conceptos especificados en ella podrán reutilizarse para describir cómo se organiza el trabajo colaborativo en distintos entornos. Asimismo, esta ontología de dominio puede verse como un metamodelo que guía el proceso de desarrollo de sistemas colaborativos de acuerdo con la filosofía de AMENITIES.

La declaración de instancias de estos conceptos que hagan referencia a entidades del mundo real en un entorno colaborativo concreto, así como, las especializaciones de esta ontología de dominio y/o la definición de nuevos conceptos (p.e. tipos de roles o actividades) para dominios específicos (p.e. una universidad, un aeropuerto, una oficina, etc.), se realizará posteriormente por medio de *ontologías de aplicación*.

Ambos tipos de ontologías, *de dominio* y *de aplicación*, capturarán la estructura del sistema El modelo del sistema colaborativo se complementará con la especificación de su

comportamiento mediante la descripción de flujos de actividades y acciones. Para ello, habrá que definir previamente en el nivel de la ontología de dominio un conjunto de clases y relaciones que nos permita representar la secuencia de pasos por las que pueda atravesar una tarea, es decir, la estructura conceptual de un flujo de trabajo: actividades, acciones, orden en que tienen lugar, etc. La forma en que se organizan flujos de actividades para tareas concretas en ámbitos específicos se describirá en ontologías de aplicación.

De esta forma, se sientan las bases para un proceso de desarrollo de ontologías modular donde la ontología global del sistema colaborativo que se está modelando, se construye a partir de fragmentos divididos en función del nivel de especificidad de la información contenida en ellos. En la sección 3.1 se tratarán con más detalle los niveles de diseño en las ontologías creadas y su justificación de cara a la evolución de dichas ontologías.

Por tanto, para la creación de esta primera ontología de dominio seguiremos el proceso de desarrollo propuesto por Noy y McGuinness [Noy 2001], que introdujimos en el capítulo III (sección 4.1), extendiéndolo en el paso 6 para incluir la definición de jerarquías de propiedades, y omitiendo transitoriamente el último paso de definición de instancias (que retomaremos al crear las ontologías de aplicación). Asimismo, veremos cómo su implementación en el lenguaje OWL 2 requerirá puntualmente el uso de ciertos *patrones de diseño* [Gamma 1995] para ontologías [Gangemi 2007].

1º. Determinar el dominio y alcance de la ontología. En nuestro caso, este paso está ya determinado, puesto que AMENITIES es una metodología para el análisis y diseño de *sistemas cooperativos*, éste será el dominio de la ontología.

2º. Considerar reutilizar otras ontologías existentes. En este caso, nuestro objetivo es implementar un marco conceptual concreto y representarlo lo más fielmente posible, de tal forma que se preserve la coherencia con otros modelos de sistema (cfr. sección 3 en capítulo II). No obstante, este proceso de formalización también sirve para revisar la definición de algunos conceptos y adaptarla a nuevas necesidades, y de esta forma, alinearnos con el principio de diseño de ontologías que caracteriza este proceso como esencialmente iterativo.

Por otro lado, el ámbito de los sistemas colaborativos es bastante genérico y la mayor parte de los términos del marco conceptual de AMENITIES (rol, tarea, grupo, etc.), son comunes a otras propuestas de diseño.

En el momento de comenzar la definición de nuestra ontología, la propuesta más conocida de modelado de organizaciones era la presentada dentro del proyecto *Toronto Virtual Enterprise* (TOVE) [Fox 1998]. Posteriormente han aparecido otras ontologías para

describir sistemas colaborativos u organizaciones como, por ejemplo, las presentadas en [Penichet 2008] y [Plisson 2007].

Todas ellas presentan enfoques interesantes, y en muchos casos, comparten con AMENITIES la forma de relacionar gran parte de los conceptos, lo que facilita la reutilización a nivel conceptual (es decir, la inclusión en la ontología para AMENITIES de un concepto definido en alguna de las otras propuestas no llevaría a inconsistencias).

Sin embargo, en los casos de TOVE y la ontología de [Penichet 2008], la principal dificultad para reutilizarlas la encontramos en el lenguaje de implementación de la ontología. TOVE hace uso del lenguaje KIF, basado en lógica de primer orden y, en consecuencia, no decidible para procesos de razonamiento automático. La ontología de [Penichet 2008] está descrita en su mayor parte en lenguaje natural y también hace uso de UML 2.0 para representar los conceptos gráficamente. Por tanto, carece de una semántica formal completa (cfr. sección 6.1.1 en capítulo III) que permita su procesamiento automático para ser descubierta y reutilizada, así como, llevar a cabo tareas de razonamiento. La ontología de [Plisson 2007] está implementada en OWL, lo que es una ventaja a la hora de importar definiciones de conceptos, aunque se centra en el modelado de la estructura de organizaciones, sobre todo de carácter empresarial, y no tanto en el modelado de la colaboración como ocurre en AMENITIES.

Por último, una carencia importante común a todas estas propuestas la encontramos en la ausencia de soporte para modelar cómo se van a desarrollar las actividades y acciones que conforman una tarea, es decir, el orden en el que habrán de realizarse cada una de ellas. Sobre este punto incidiremos más adelante en la sección 2.3 al modelar los flujos de trabajo mediante clases y relaciones de una ontología.

Todos esto motivos han hecho que realmente no hayamos reutilizado en sentido estricto nada que no estuviera ya definido en el marco conceptual de AMENITIES. No obstante, de cara a posibles extensiones retomaremos las ontologías citadas para volver a estudiar la definición de algunos conceptos.

3º. Enumerar los términos relevantes de la ontología. La especificación del marco conceptual relaciona una serie de términos comúnmente usados a la hora de describir un sistema colaborativo, tales como, *Evento*, *Acción*, *Artefacto*, *Objeto de Información*, *Actividad*, *Tarea*, *Actor*, *Rol*, *Capacidad*, *Tarea Cooperativa*, *Organización*, *Ley*, *Grupo* y *Protocolo de Interacción*.

Cada uno de estos términos representa un concepto para el que la metodología define cómo ha de interpretarse en cada modelo. En la definición de cada concepto se descubren otros nuevos (p.e. *Unidad de Trabajo*), y también otros términos que se utilizan para expresar relaciones entre dichos conceptos como, por ejemplo, que los actores *desempeñan* roles, los eventos *disparan* acciones, los grupos *se componen* de actores, etc.

4º. Definir las clases y la jerarquía de clases. Al contar con un marco conceptual ya definido, esta etapa y la siguiente se simplifican considerablemente. En efecto, para determinar las clases de la ontología seleccionaremos los términos de la etapa anterior para los que la metodología define un concepto: Evento, Acción, etc. Asimismo, la representación del marco conceptual en un diagrama de clases de UML (cfr. Figura II.4 en capítulo II), aunque no recoge dicho marco de forma completa, constituye un buen punto de partida que ayuda al proceso de creación de la ontología que nos ocupa, ya que, las clases de UML pueden traducirse directamente a clases de la ontología [Hurtado 2002] [Evermann 2008]. En la sección 2.2 se tratarán los fundamentos lógicos de esta y otras correspondencias entre elementos del lenguaje UML y OWL. En la Figura IV.1 se muestra un fragmento de la definición de algunas de estas clases en OWL 2 utilizando la sintaxis RDF/XML.

```
<owl:Class rdf:about="#Task"/>
<owl:Class rdf:about="#Action"/>
<owl:Class rdf:about="#Activity"/>
<owl:Class rdf:about="#Artefact"/>
<owl:Class rdf:about="#Capability"/>
<owl:Class rdf:about="#Event"/>
<owl:Class rdf:about="#Group"/>
<owl:Class rdf:about="#Role"/>
```

Figura IV.1 Declaración de clases en OWL 2 en sintaxis RDF/XML

Por lo que respecta a la jerarquía de clases, ésta no es muy profunda, ya que, el conjunto de clases no es muy extenso (menos de una veintena de conceptos), y por otro lado, la idea es definir clases generales que provean un marco de referencia para el modelado de sistemas colaborativos. Además, como ya se ha señalado al comienzo de esta sección, estas clases podrán ser especializadas para ámbitos concretos en ontologías de aplicación.

A pesar de todo, a través de su definición es posible identificar algunas jerarquías de clases. Por ejemplo, las similitudes de comportamiento que presentan Actividades y Acciones a la hora de modelar una tarea hacen que, en ciertos casos, puedan abstraerse bajo el concepto de *Unidad de Trabajo*. Así, la nueva clase o concepto se obtiene siguiendo un enfoque *bottom-up* (cfr. sección 4.1 del capítulo III y [Uschold 1996]). En realidad, no existirán instancias de la clase *Unidad de Trabajo*, sino que esta más bien se introduce como un modo de simplificar y agrupar la descripción de ciertas relaciones de las Actividades y las Acciones con otras clases del marco conceptual. Para ello, se hace uso del patrón de diseño *axioma de cobertura*, que hace equivalente la superclase *Unidad de Trabajo* a la unión de las subclases *Actividad* y *Acción*, es decir, en lógica descriptiva sería:

$$Work\_Unit \equiv Activity \sqcup Action$$

La Figura IV.2 (a) muestra la especificación en lógica descriptiva del axioma de cobertura, mientras que la Figura IV.2 (b) muestra la especificación en OWL 2 de la clase *Unidad de Trabajo*.

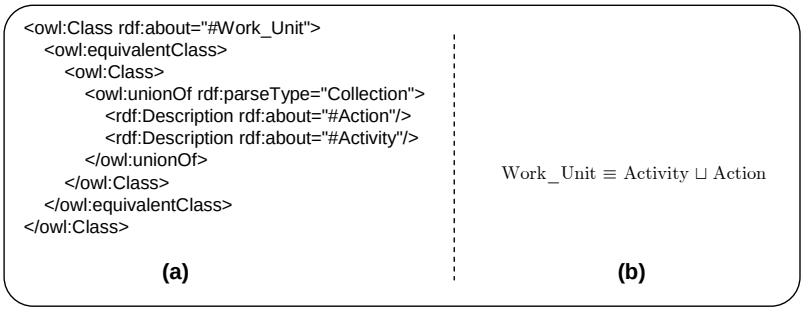


Figura IV.2 Ejemplo de especificación de axioma de cobertura: (a) en OWL 2; (b) en Lógica Descriptiva

Como ejemplo de clase diseñada siguiendo el enfoque complementario *top-down*, encontramos la clase *Equipo de Trabajo*, utilizada para hacer referencia a un tipo especial de grupo que se crea temporalmente para realizar una actividad específica.

5º. Definir las propiedades (también llamadas relaciones o *slots*) de las clases. Por medio de las propiedades se estructuran los conceptos, lo que a su vez, configura la arquitectura de un sistema colaborativo desde la perspectiva de AMENITIES.

En la lista de términos obtenida en el tercer paso ya aparecían bastante delimitados, por un lado, aquellos que correspondían a conceptos y, por otro, los utilizados para relacionarlos o describir sus propiedades. Tras haber escogido de dicha lista los términos que representarán conceptos (paso 4º), la mayor parte de los términos que queden representarán propiedades.

Al igual que ocurría en el paso anterior, el diagrama de clases de UML en el que se sintetiza el marco conceptual de AMENITIES, puede utilizarse para identificar fácilmente las relaciones entre conceptos. En este caso, serán las asociaciones del diagrama de clases las que se traducirán a propiedades de la ontología.

Así, encontramos que un sistema cooperativo *se compone* fundamentalmente de grupos, organizaciones, leyes, artefactos, eventos y objetos de información. Los actores *realizan* acciones *usando* artefactos, los grupos *se componen* de actores, los roles *se componen* de tareas, que a su vez, *se componen* de unidades de trabajo (es decir, acciones y actividades), ciertas tareas pueden *interrumpir* a otras, etc.

Por otro lado, hemos tratado de seguir la pauta consistente en asociar propiedades y relaciones a la clase más general para la que se pueda definir. Por ejemplo, tanto la realización de acciones como de actividades puede requerir *hacer uso de* y/o *producir* objetos de información. En este caso, bastará con asociar las relaciones *hacer uso de* y *producir* a la clase más general Unidad de Trabajo, y que serán heredadas por instancias de las clases Actividad y Acción.

Otro aspecto que merece especial atención en este paso es la representación en OWL 2 de relaciones n-arias, esto es, relaciones que involucran a tres o más elementos a la vez. En la definición del marco conceptual que nos ocupa se pueden identificar varias de estas relaciones. Por ejemplo, los *grupos* (elemento 1) realizan *actividades* (elemento 2) siguiendo ciertos *protocolos de interacción* (elemento 3); el desempeño de un *rol* (elemento 1), puede llevar al desempeño de otro rol (elemento 2) cuando se da la condición expresada en una ley (elemento 3) asociada…

Sin embargo, hasta ahora las versiones del lenguaje OWL no ofrecen soporte nativo para representar relaciones entre más de dos elementos. La forma de resolver este problema se verá con más detalle en la sección 2.1.1 mediante un patrón de diseño concreto.

6º. Definir facetas y/o restricciones sobre los *slots* o relaciones. En el paso anterior hemos visto cómo la relación *se compone* asocia diversas clases: un grupo y los actores que lo integran, las tareas incluidas en un rol, etc. Al mismo tiempo, parece claro que un grupo *no* se compone de tareas y que no tiene mucho sentido hablar de grupos integrados por un solo actor.

Por tanto, en esta fase se pueden establecer ciertas restricciones sobre el rango y el dominio de los *slots*, así como, la cardinalidad que deban tener algunos de ellos. Por ejemplo, los grupos estarán compuestos por actores. Así, el rango de la relación *se compone* para la instancias de la clase Grupo (dominio de la relación) se restringe para que únicamente tome valores de las clase Actor. Esto puede recogerse mediante una *restricción universal*, es decir, una restricción que limita el rango de una propiedad a valores de una única clase (en este caso la clase Actor). Además, un grupo *se compone* al menos de dos actores, para lo que puede hacerse uso de una *restricción de cardinalidad*. En la Figura IV.3 se muestra la especificación en OWL de las restricciones que acabamos de mencionar.

```
<owl:Class rdf:about="#Group">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#has_part"/>
         <owl:allValuesFrom rdf:resource="#Actor"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#has_part"/>
         <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Figura IV.3 Especificación de restricciones sobre la relación *se compone* para la clase Grupo

Aún pueden darse situaciones más específicas, como el caso de la relación *se compone* cuando se trata de la clase *Sistema Colaborativo*. En efecto, según AMENITIES un sistema colaborativo *se compone* en primera instancia de artefactos, eventos, grupos, objetos de información, leyes y organizaciones (cfr. sección 3.4.1 del capítulo II). Sin embargo, mientras que la especificación de algunos de estos elementos puede ser accesoria en ciertas situaciones (por ejemplo, los eventos o los objetos de información pueden dejarse al margen cuando no se quiere entrar en muchos detalles a la hora de describir un sistema), parece claro que siempre deba especificarse al menos un grupo y una organización. Puesto que OWL 2 permite formular expresiones de la lógica $\mathcal{SROIQ}$, las restricciones anteriores sobre la relación *se compone* pueden recogerse mediante *restricciones de cardinalidad cualificadas*, esto es, restricciones en las que, además del número de instancias con que se relaciona cada instancia de una clase (cardinalidad), se puede especificar la clase a la que deben pertenecer los elementos con los que se relaciona (cfr. capítulo III, sección 5.2.2). Así, puede establecerse que un Sistema Colaborativo *se compone* al menos de un Grupo y una Organización. La Figura IV.4 muestra la forma de reflejar esta situación utilizando la sintaxis funcional para OWL 2. Señalar que hemos utilizado la nueva versión del lenguaje puesto que, la primera versión de OWL no permitía especificar el tipo de los elementos de una relación en una restricción de cardinalidad [W3C OWL 2 2008].

```
// Class: http://localhost/amenitiesOWL/amenities.owl#Collaborative_System

SubClassOf(Collaborative_System ObjectAllValuesFrom(has_part ObjectUnionOf(Artefact
                                                                            Event
                                                                            Group
                                                                            Information_Object
                                                                            Law
                                                                            Organization)))
SubClassOf(Collaborative_System ObjectMinCardinality(1 has_part Organization))
SubClassOf(Collaborative_System ObjectMinCardinality(1 has_part Group))
```

Figura IV.4 Restricciones sobre la relación *se compone* para la clase Sistema Colaborativo

Aunque no se aborda en el método propuesto por Noy y McGuinness [Noy 2001], otra forma de restringir una relación es definiéndola como una especialización de otra relación [W3C OWL 2004]. Tal es el caso de las transiciones aditivas entre roles. Estas transiciones describen un tipo especial de cambio en las funciones que desempeña un actor según el cual, por un lado, adquiere las capacidades necesarias para realizar las funciones propias del nuevo rol (como en una transición normal entre roles), y por otro lado, preserva las capacidades propias del rol que venía desempeñando [Garrido 2003]. Ambos tipos de transiciones se recogen de forma sintética en el diagrama de clases de la Figura II.4 mediante la asociación *conecta*. Para la representación en OWL de ambos tipos de transiciones hemos definido la relación *conecta* y la relación *conecta_aditiva* como un subtipo especial de la primera. Para ello, utilizamos el constructor *subPropertyOf* que proporciona el lenguaje OWL. En la Figura IV.5 se puede ver cómo quedaría la especificación de las dos propiedades:

```
// Object property: http://localhost/amenitiesOWL/amenities.owl#connects

ObjectPropertyDomain(connects ObjectUnionOf(Role Organization))


// Object property: http://localhost/amenitiesOWL/amenities.owl#connects_additive

Declaration(ObjectProperty(connects_additive))
SubObjectPropertyOf(connects_additive connects)
ObjectPropertyDomain(connects_additive Role)
```

Figura IV.5 Ejemplo de jerarquía de propiedades entre la relación *conecta* y *conecta_aditiva*

7º. Definir instancias. Las instancias de las clases definidas en el marco conceptual que hemos implementado hasta ahora corresponderán a entornos colaborativos concretos: una oficina, un aeropuerto, una sucursal bancaria, etc. Como ya hemos comentado, este paso se tratará al diseñar las ontologías de aplicación para casos concretos en la sección 2.4 de este capítulo. De nuevo, las instancias de clases de modelos UML se harán corresponder con instancias individuales de conceptos de la ontología en OWL.

2.1.1 Algunas consideraciones de implementación con el lenguaje OWL

En el capítulo I, sección 3, veíamos cómo la capacidad expresiva de los constructores de un lenguaje de modelado determina su idoneidad para representar entidades en un dominio. Por otro lado, no existen la ontología ni el lenguaje de modelado perfectos [Noy 2001][Wand 1995].

Los constructores del lenguaje OWL abarcan un amplio espectro de operadores de la teoría de conjuntos, facilitando la representación de ciertos conceptos complejos (la intersección de dos conceptos, el conjunto complementario, etc.). Sin embargo, algunas

relaciones entre conceptos que se pueden diseñar a nivel conceptual, luego no cuentan con una implementación directa, o las restricciones que impone el lenguaje para preservar las propiedades de decibilidad en el razonamiento limitan el uso de ciertos constructores [W3C SWBP 2006].

Tal es el caso de la representación de relaciones n-arias (relaciones que involucran más de dos instancias individuales[1]), o la especificación de restricciones de cardinalidad sobre propiedades transitivas. Al observar la descripción de los conceptos utilizados en AMENITIES encontramos varias situaciones en las que es natural recurrir a este tipo de construcciones. A continuación mostramos cómo las hemos modelado.

Relaciones n-arias. En AMENITIES, las transiciones entre roles, que permiten que un actor pase a desempeñar funciones distintas de las asignadas inicialmente, vienen reguladas por leyes. Por tanto, una forma de modelar estas situaciones sería utilizar una asociación n-aria (con *n* igual a tres), que relacionase simultáneamente los dos roles que intervienen en la transición junto con la ley que regula el cambio. Lo mismo cabría aplicar en el caso de las transiciones aditivas.

Algo similar ocurre al modelar la realización de una actividad por un grupo siguiendo ciertos protocolos de interacción. En la misma línea, nos encontramos con que los actores realizan acciones por medio de artefactos.

UML permite capturar este tipo de situaciones mediante asociaciones n-arias o mediante clases asociación. En cambio, en OWL no disponemos de un constructor específico para ellas, sino que para modelarlas hemos de recurrir a un patrón de diseño denominado *reificación* y que consiste en crear una nueva clase y *n* nuevas relaciones funcionales, una por cada participante en la asociación. Las instancias de la nueva clase representarán instancias (tuplas) de la relación n-aria que se está modelando [W3C SWBP 2006].

Conviene destacar que, aunque este enfoque no sea semánticamente equivalente a una relación n-aria en un diagrama de clases de UML, se puede considerar *correcto*, en el sentido de que preserva la consistencia de la definición de los conceptos y el razonamiento que podríamos llevar a cabo en dicho diagrama de clases [Berardi 2005][Calvanese 2001].

Por último, las nuevas clases (o conceptos) *reificadas* las vamos a declarar como subclases de otra clase más general *Reified_Relation*, con el propósito de que, por un lado, puedan ser identificadas como conceptos que no se corresponden con entidades del dominio, y por otro lado, que posteriormente puedan ser procesadas y mostradas de forma

[1] Las instancias que forman parte de una relación se denominan también *argumentos* de la relación

diferente por herramientas gráficas de visualización y edición de ontologías. La Figura IV.6 muestra la especificación en OWL de las clases y conceptos definidos para modelar una transición entre dos roles y que viene regulada por una ley.

```
// Class: http://localhost/amenitiesOWL/amenities.owl#Role_Connection_Relation

SubClassOf(Role_Connection_Relation Reified_Relation)
SubClassOf(Role_Connection_Relation ObjectAllValuesFrom(role_value Role))


// Object property: http://localhost/amenitiesOWL/amenities.owl#connects

ObjectPropertyDomain(connects ObjectUnionOf(Role Organization))


// Object property: http://localhost/amenitiesOWL/amenities.owl#connection_law

ObjectPropertyDomain(connection_law Role_Connection_Relation)
ObjectPropertyRange(connection_law Law)


// Object property: http://localhost/amenitiesOWL/amenities.owl#role_value

ObjectPropertyDomain(role_value Role_Connection_Relation)
ObjectPropertyRange(role_value Role)
```

Figura IV.6 Especificación en OWL de la relación n-aria conecta mediante una clase reificada

En la Figura IV.6 puede apreciarse cómo, siguiendo el patrón que acabamos de describir, se ha definido una nueva clase *Role_Connection_Relation* (cuyas instancias representarán transiciones entre roles), y tres relaciones o propiedades (una por cada argumento de la relación):

- *conecta*, que relaciona el rol origen con la nueva clase reificada *Role_Connection_Relation*.
- *ley_de_conexión*, que enlaza instancias de la clase *Role_Connection_Relation* con la ley asociada a cada transición entre roles.
- *valor_del_rol* que enlaza instancias de la clase *Role_Connection_Relation* con el rol destino de cada transición.

En la Figura IV.7 se representa gráficamente la transición aditiva entre el rol *jefeDeRiesgos* y el rol *director* de una sucursal bancaria cuando el segundo se encuentra ausente (ley). La figura se ha obtenido a partir de la especificación en OWL de la información contenida en el *diagrama de organización* de la Figura II.5 de esta memoria. Para su elaboración hemos utilizado el *plug-in Jambalaya*[2] para la herramienta de edición de ontologías *Protégé* [Knublauch 2004]. Las cajas con el borde rojo corresponden a clases de la ontología y las cajas moradas a instancias de dichas clases.

[2] Versión utilizada: 2.3.5, Build: 3, 11/04/2006 14:45. The Jambalaya Project. Más información en: http://www.thechiselgroup.org/jambalaya

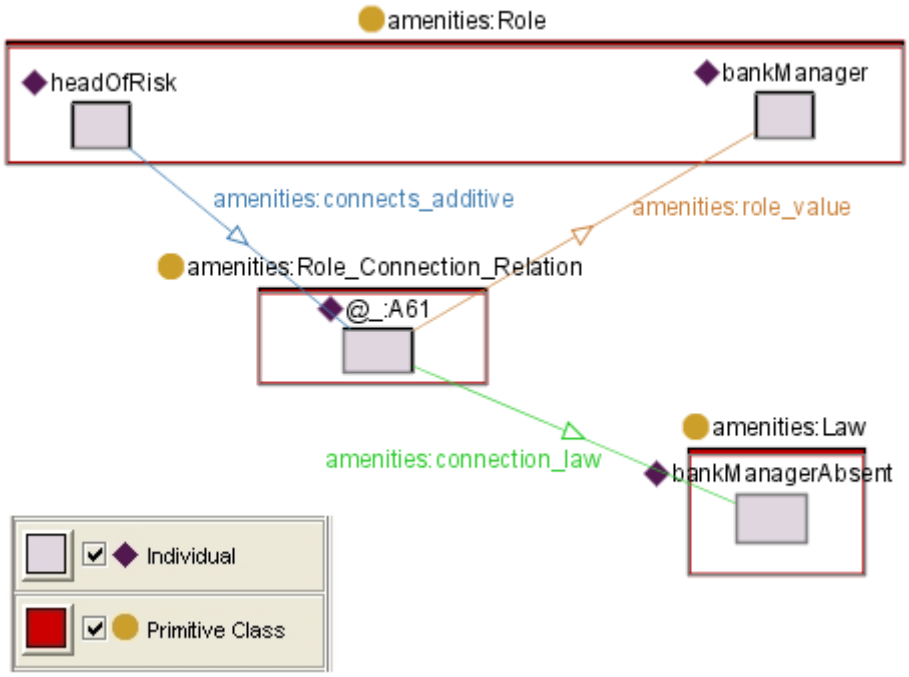


Figura IV.7 Representación gráfica de clases e instancias de la relación n-aria *conecta_aditiva*

Restricciones de cardinalidad sobre propiedades transitivas. Durante la etapa de especificación de restricciones sobre las propiedades, establecimos que los elementos de la clase Sistema Colaborativo debían estar compuestos, al menos, de un grupo y una organización. Es decir, estábamos definiendo restricciones de cardinalidad sobre la relación de composición para el caso de la clase Sistema Colaborativo.

Por otro lado, esta relación de composición (que nosotros modelamos en OWL mediante la propiedad *se compone*) es inherentemente transitiva. Así, de un sistema colaborativo compuesto por un grupo de actores, puede decirse que está compuesto por dichos actores. Lo mismo cabe aplicarse para la relación de composición entre un sistema colaborativo, sus organizaciones y los roles que forman parte de dichas organizaciones.

Sin embargo, mantener la decidibilidad de los procesos de razonamiento automático obliga a no poder declarar restricciones de cardinalidad sobre una propiedad transitiva o cualesquiera de sus *superpropiedades*[3], ni tampoco sobre su propiedad inversa[4] o las superpropiedades de esta última [W3C OWL 2004c].

La solución a este problema pasa por declarar como una nueva relación que será una superpropiedad (o *super-relación*) de aquélla sobre la que se han definido restricciones de

[3] Propiedad de la que hereda una propiedad o relación.

[4] Por ejemplo, una propiedad inversa de *se compone* podría ser la propiedad *es_parte_de*, y que también tiene carácter transitivo.

cardinalidad. Esta nueva superpropiedad será la que se declare transitiva [Drummond 2006].

Por ejemplo, para el caso de la relación *se compone*, podemos declarar una nueva propiedad *comprende*, como superpropiedad de la primera y transitiva. Así, el sistema colaborativo de concesión de préstamos que presentamos en el capítulo II, sección 3.7, vemos que *se compone* de tres organizaciones: una sucursal, una notaría y una agenciaDeTasación. En consecuencia, según las definiciones que acabamos de realizar, automáticamente el sistema de concesión *comprende* también la sucursal, la notaría y la agencia de tasación. De la misma forma, la sucursal *se compone* de los roles cajero, jefeDeRiesgos y director, y a su vez, *comprende* dichos roles. Como la relación *comprende* es transitiva, el sistema colaborativo de concesión de préstamos que *comprende* la sucursal bancaria, también *comprende* los roles cajero, jefeDeRiesgos y director. La Figura IV.8 representa el proceso de razonamiento seguido.

Todas las instancias de la relación *comprende* pueden ser inferidas automáticamente por un razonador a partir de la declaración de instancias de la relación *se compone*.

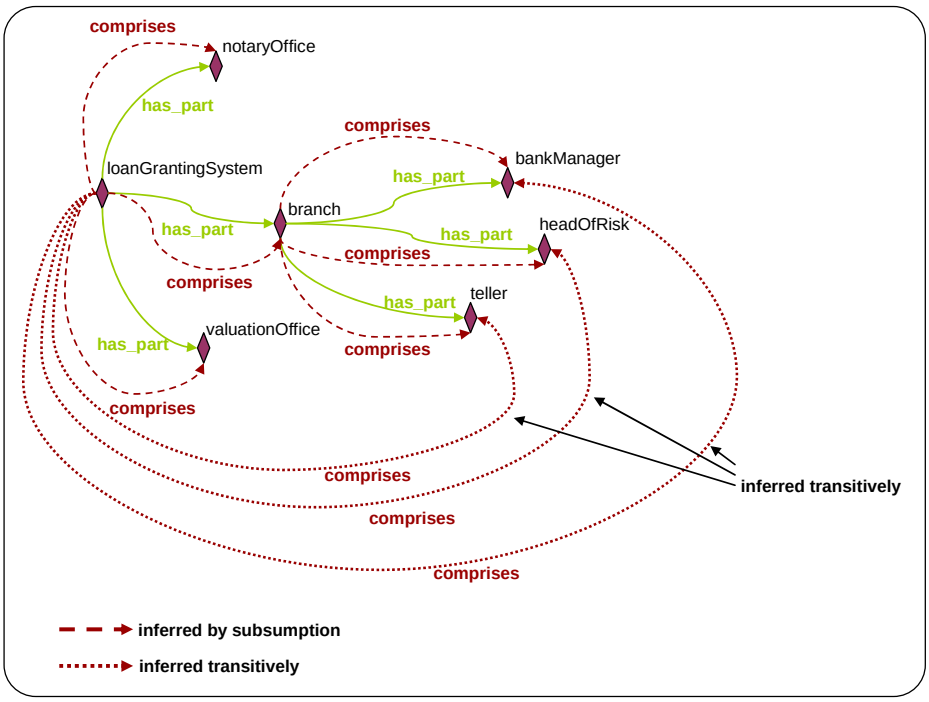


Figura IV.8 Restricciones de cardinalidad sobre propiedades transitivas

OWL no es un lenguaje de Programación Orientada a Objetos (POO). Finalmente, aunque en buena parte del proceso de creación de la ontología anterior hayamos tomado como referencia un diagrama de clases de UML, conviene tener presente algunas

distinciones para que las equivalencias establecidas tácitamente ofrezcan ciertas garantías de consistencia.

En efecto, OWL y UML comparten el hecho de realizar un modelado centrado en objetos, lo que hace que las clases y asociaciones de ambos lenguajes compartan gran parte de su significado. Sin embargo, en OWL conviene tener presente que [W3C SWBP 2006b]:

- Las propiedades o relaciones existen independientemente de clases específicas.
- Las instancias de las clases pueden tener múltiples tipos y cambiar su tipo como resultado de un proceso de clasificación.
- La definición de una clase puede cambiar dinámicamente como resultado de un proceso de inferencia.

Adicionalmente, OWL es un lenguaje estrictamente declarativo y lógico. En consecuencia, carece de los aspectos operacionales de la POO, como los métodos. Asimismo, el razonamiento en OWL es estrictamente lógico y no se pueden definir excepciones o sobrecarga de operadores [W3C OWL 2 2008c].

2.1.2 Representación gráfica del marco conceptual a partir de su especificación en OWL

En la Figura IV.9 se muestra una representación gráfica de la especificación resultante hasta el momento (algunas relaciones y clases se han eliminado por motivos de legibilidad). Su correlación con el diagrama de clases de la Figura II.4 del capítulo II es inmediata:

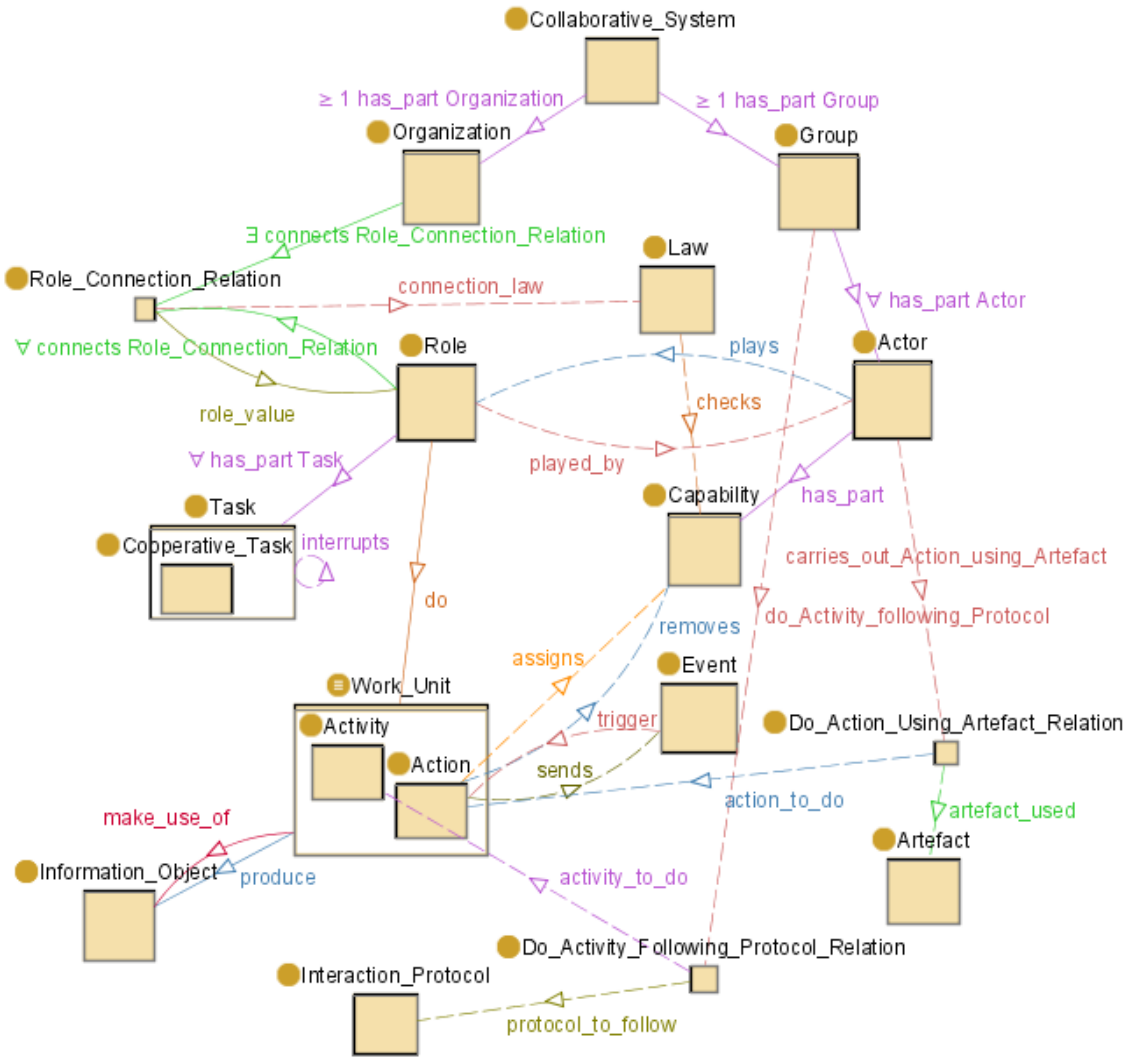


Figura IV.9 Representación gráfica del marco conceptual de AMENITIES a partir de su especificación en OWL

2.2 Correspondencia entre modelos UML y OWL

Acabamos de ver cómo, un diagrama de clases de UML puede servir de guía en la creación una ontología OWL respetando ciertas diferencias conceptuales acerca de cómo se conciben las clases y los atributos en ambos lenguajes. Asimismo, ambos modelos se pueden traducir de uno a otro siguiendo un método consistente básicamente en codificar las clases del diagrama como clases de la ontología, y las asociaciones como propiedades. Los atributos de las clases también se traducirían a propiedades de las clases en OWL y las instancias de las clases (objetos) se corresponderían con las instancias de los conceptos de la ontología [Berardi 2005][Evermann 2008]. Una correlación similar se utiliza en otros lenguajes de modelado orientado a objetos y OWL [Jarrar 2007][Keet 2007].

Aparte del carácter intuitivo del esquema de traducción presentado, podemos argüir los siguientes fundamentos lógicos que garantizan su consistencia y corrección de cara a la preservación de la semántica de los modelos y los procesos de razonamiento que podamos realizar en base a ellos:

- Es posible obtener una formalización en Lógica de Primer Orden (FOL), tanto de un diagrama de clases de UML (y sus diagramas de objetos asociados), como de una base de conocimiento en OWL, utilizando el mismo alfabeto, de tal forma que las interpretaciones $\mathcal{I}$ y $\mathcal{J}$ que se definan[5] respectivamente para dichas formalizaciones sean *compatibles* (comparables). A partir de ahí, es posible demostrar que toda clase de un diagrama de clases es consistente (es decir, satisface su interpretación $\mathcal{I}$ expresada en FOL), siempre y cuando (*si y sólo si*), el correspondiente concepto de la ontología es consistente con respecto a la base de conocimiento OWL en la que está definido (es decir, la interpretación $\mathcal{J}$ para tal concepto es distinta del conjunto vacío o satisfacible) [Berardi 2005].

- Es cierto que no es posible capturar íntegramente la semántica de los diagramas de clases de UML en OWL (tampoco en $\mathcal{SROIQ}$, la lógica descriptiva en la que está basado), ya que no existen constructores para expresar relaciones n-arias, ni tampoco restricciones de identificación o dependencias funcionales [Calvanese 2001]. Sin embargo, como consecuencia de lo declarado en el apartado anterior, la correlación descrita más arriba puede considerarse "*correcta en el sentido de que preserva la consistencia de clases, y de ahí, esencialmente todos los servicios de razonamiento sobre diagramas de clases de UML*", como consistencia, subsunción y equivalencia de clases para los que las restricciones de identificación o las dependencias funcionales no son necesarias [Berardi 2005]. Además, aunque hay que reconocer que continúan utilizándose, las referencias explícitas a las restricciones de identificación y las dependencias funcionales han desaparecido del metamodelo en la última revisión UML [OMG 2007]. Actualmente, se encuentran bajo estudio diversas alternativas para tratar de representar restricciones de identificación precisamente sobre relaciones n-arias utilizando un conjunto de constructores decidible [Calvanese 2007][Calvanese 2008][Parsia 2008].

---

[5] Remitimos al lector a la sección 5.2.2 y la tabla III.1 del capítulo III. Allí se proporciona un ejemplo de formalización en FOL para un conjunto de operadores de lógicas descriptivas (en las que está basado el lenguaje OWL), definiendo una interpretación $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

- Por último, con respecto al modelado de relaciones n-arias (para las que se define una nueva clase y *n* nuevas relaciones funcionales), una variante particular de una propiedad, denominada *propiedad del modelo de árbol*, de la lógica descriptiva $\mathcal{SROIQ}$ garantiza la unicidad de la representación propuesta, es decir, no existirán dos o más instancias del concepto *reificado* por cada tupla que represente una instancia de la asociación n-aria [Motik 2008][Horrocks 2001].

Aparte de su factibilidad, en el capítulo anterior abordamos los beneficios que reporta el uso combinado y la correspondencia entre los modelos de ambos lenguajes, en tanto que permite razonar sobre especificaciones en UML y habilita la implementación de enfoques de desarrollo dirigidos por ontologías (cfr. sección 6.1 del capítulo III).

2.3 Especificación ontológica de procesos colaborativos

Sin duda, el modelado y especificación de tareas ocupa un papel central en el diseño de un sistema colaborativo. De hecho, la mayoría de las ontologías relacionadas que se pueden encontrar en la bibliografía introducen conceptos como Tarea, Acción, etc., para describir la *estructura* de sistemas de este tipo [Fox 1998][Plisson 2007].

Por otro lado, para modelar íntegramente un sistema colaborativo es necesario, además de su estructura, describir los procesos que tendrán lugar en él, esto es, el *comportamiento* del sistema. Para ello, las tareas y acciones han de poder disponerse ordenadas en *secuencias* que representen el flujo de trabajo a seguir. Este es el propósito de notaciones como los diagramas de actividades de UML y con este fin son utilizados en AMENITIES.

Sin embargo, los lenguajes de representación del conocimiento utilizados para especificar ontologías suelen centrarse en describir únicamente la estructura de los conceptos para un dominio, mientras que la descripción del comportamiento queda fuera del diseño ontológico, o se realiza separadamente mediante algún tipo de lenguaje procedural [Loom 1995]. Como consecuencia, la información comienza a dispersarse por diferentes modelos, dificultando su gestión, y la tarea de modelado se hace más proclive a errores [Lange 2006]. Asimismo, se desaprovecha la capacidad que proporcionan las ontologías para razonar acerca la estructura del comportamiento, esto es, la forma en que se organizan y disponen las tareas en un flujo de trabajo.

En muchos casos, estas limitaciones vienen impuestas por el propio lenguaje de representación de ontologías, que no proporcionan los constructores necesarios (vocabulario) para representar explícitamente conceptos estructurados como listas, secuencias, conjuntos, etc. En definitiva, volvemos a toparnos con el problema de que la capacidad expresiva de un lenguaje de modelado condiciona su adecuación para representar

las entidades de un dominio (en este caso, los procesos que tienen lugar en él) [Wand 1995].

A continuación abordamos el problema de la representación de secuencias en OWL y su aplicación a la representación de diagramas de actividades de UML.

2.3.1 Representación de secuencias de actividades en OWL

En OWL todas las entidades de una ontología se describen en términos de clases, propiedades, instancias de clases y relaciones entre dichas instancias. Como acabamos de ver, estos elementos bastan para describir la estructura de los conceptos de un sistema colaborativo, y que puede estar representada en un diagrama de clases de UML.

Sin embargo, OWL no provee constructores nativos para modelar listas ordenadas de instancias de conceptos. Este tipo de construcciones pueden ser útiles si queremos modelar secuencias de actividades. De esta forma, podríamos expresar, por ejemplo, que una tarea *t1* consta de una lista de tres actividades –*a1*, *a2* y *a3*–, para indicar que ése es el orden en que deben realizarse dichas actividades para completar *t1*. En este sentido, en OWL es posible utilizar constructores de listas heredados del lenguaje RDF (*rdf:List*, *rdf:first, rdf:rest, rdf:nil*), para implementar secuencias de elementos ordenados, pero su uso fuera de otras construcciones (por ejemplo, combinado con *owl:unionOf*, utilizado para representar un concepto como la unión de una lista de conceptos), inmediatamente convierte la ontología en OWL Full, con la consiguiente pérdida de garantías computacionales para el razonamiento [W3C OWL 2004c]. Por tanto, si queremos explotar las capacidades de un razonador sobre la descripción de un sistema colaborativo en el lenguaje OWL, la descripción de secuencias habrá de realizarse por medio de soluciones alternativas.

Otras propuestas, como la del lenguaje OWL-S [W3C OWL-S 2004] o la ontología SWRC[6] [Sure 2005], se han ocupado de traducir los constructores de RDF para listas a conceptos y propiedades en OWL, pero no proveen mecanismos adicionales para inferir propiedades sobre el orden entre actividades [Drummond 2006].

No obstante, este esquema de representación mediante listas es bastante limitado y sólo permite describir el desarrollo de tareas sencillas. Basta pensar en la imposibilidad de especificar la realización de actividades concurrentemente, ciclos o estructuras condicionales.

En las secciones siguientes se propone un conjunto de clases y relaciones para solucionar el problema de la representación de secuencias de actividades en el lenguaje OWL.

---

[6] SWRC: Semantic Web for Research Communities

2.3.2 Representación de diagramas de actividades de UML en OWL

El metamodelo de los diagramas de actividades (en adelante DA) de UML, al igual que todo el metamodelo del lenguaje, está especificado reflexivamente en términos de conceptos del propio UML (fundamentalmente clases, paquetes, asociaciones, generalizaciones y dependencias) [Seidewitz 2003], y utilizando como notación los diagramas de clases que propone este lenguaje [OMG 2007].

Típicamente, en un diagrama de clases se representan clases y asociaciones entre ellas. Con la excepción de las relaciones n-arias (también de las restricciones de identificación y las dependencias funcionales, pero no están presentes como tales en el metamodelo de UML [OMG 2007]), estos elementos pueden mapearse fácilmente a clases y propiedades en una ontología en OWL conservando la misma semántica, como ya hemos visto para el caso del marco conceptual de AMENITIES.

Además, puesto que AMENITIES propone el uso de los DA de UML para modelar el desarrollo de tareas y actividades, parece razonable establecer una correspondencia entre las clases y asociaciones del metamodelo de los DA, a un conjunto de clases y propiedades OWL que permita representar el desarrollo de una tarea o actividad en ontologías creadas para este lenguaje.

En este caso, dicha correspondencia no tendría por qué comportar ninguna pérdida semántica ya que, el metamodelo de UML para los DA no hace uso de relaciones n-arias para su descripción. Por tanto, cada instanciación $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ que sea un modelo (de la formalización en FOL) de un DA de UML, sería también un modelo de la correspondiente representación en OWL según el esquema de traducción propuesto, y viceversa.

No obstante, en este proceso de traducción a OWL, queremos aprovechar algo de la experiencia y crítica posterior que se ha hecho de los DA de UML para mejorar la forma de representar flujos de tareas, particularmente las que se refieren a su desconexión del modelo de organización de un sistema colaborativo [Russell 2006]. Por otro lado, tampoco tendría mucho sentido codificar sin más el metamodelo de los DA en OWL, al margen de que esto habilitara para potenciales procesos de inferencia sobre la estructura de clases resultante. Por último, en la especificación del metamodelo de UML para los DA, la jerarquía de clases de actividades y acciones aparece sorprendentemente mezclada con las clases utilizadas para su representación gráfica en dichos diagramas, tal y como veremos más adelante [Noguera 2009].

Por estas razones, hemos pensado en distanciarnos ligeramente del metamodelo de UML y proponer un conjunto de clases con el que poder mantener una correspondencia uno a uno (aunque sin preservar la misma estructura conceptual). Esto permite convertir unos modelos en otros (los basados en UML a OWL, y al contrario, los basados en OWL a

UML), pudiéndose recuperar siempre los modelos de partida, ya que, cada elemento de un modelo de partida se corresponderá únicamente con un elemento en el modelo de destino.

A continuación se presenta el conjunto de clases y propiedades diseñado para representar los DA en ontologías OWL.

Diseño y selección del conjunto de clases para representar secuencias de actividades en OWL. La solución que se presenta comienza por conectar la ontología que acabamos de definir para el marco conceptual de AMENITIES con otro conjunto extra de conceptos y propiedades (elementos básicos de construcción de ontologías en OWL), que permitan definir secuencias que recojan el orden en que se deban desarrollar las actividades y acciones que forman parte de una tarea. Además, este conjunto también podrá servir como metamodelo para validar y probar descripciones de tareas colaborativas.

A nivel conceptual, el esquema propuesto consiste en considerar cada *tarea* (o *tarea cooperativa* según el marco conceptual de la Figura IV.9), como *descrita_en* una *secuencia* de *pasos* (*steps*). Las *actividades* también pueden describirse por medio de *secuencias* de *pasos*.

Un *paso* es una mera abstracción para conectar las diferentes etapas por las que atraviesa la *secuencia* de *unidades de trabajo* de las que se compone una *tarea* o *actividad*. Los *pasos* se clasifican en *control_flow_steps, work_unit_steps* (*activity_steps* y *action_steps*) e *information objects steps* (la jerarquía completa de las clases utilizadas puede observarse en la parte izquierda de la Figura IV.11). Estas subclases de pasos se utilizan únicamente para agrupar bajo una misma denominación la naturaleza de aquello que se va a *realizar*, *usar* o *producir* en cada *paso*, a saber:

- una reordenación del flujo de control en una tarea (p.e., su inicio, finalización, división, convergencia con otro, etc.) Para ello, se han definido consecuentemente las clases *First_step*, *Final_step*, *Sync_Step*, etc. A su vez, los *pasos de sincronización* (*sync_steps*) se subdividen en *fork_steps* y *join_steps*, y se corresponden con las barras horizontales gruesas utilizadas en los diagramas de actividades para sincronizar o iniciar un conjunto de actividades que se ejecutan en paralelo;
- una *unidad de trabajo*, es decir, una *actividad* o una *acción*, y que ya habían sido definidas en la ontología para el marco conceptual de AMENITIES;
- un *objeto de información* como resultado del cumplimiento de una actividad o acción;

La Figura IV.10 muestra la especificación en OWL de las clases y relaciones a las que acabamos de hacer referencia.

Con carácter general, cada *secuencia* consta de al menos un *first_step* (paso inicial), seguido de otra serie de pasos adicionales, y concluye en un *final_step*. Exceptuando las instancias de la clase *Final_Step*, las instancias de las clases-*step* pueden ir *seguidas_por* una o más instancias

de clases-*step*. Al igual que en los DA de UML, existe una *guarda* asociada a cada instancia de la relación *seguido_por*. Esta *guarda* es una condición a evaluar en el momento en que el flujo de control alcanza su *step* asociado. Si se evalúa como *VERDADERA*, se interpretará como que el flujo de control puede continuar hacia el siguiente *step* conectado a través de la relación *seguido_por* (*followed_by* en la implementación de la Figura IV.10).

```
// Object property: http://localhost/amenitiesOWL_11/amenities.owl#described_in
ObjectPropertyDomain(described_in ObjectUnionOf(Task Activity))
ObjectPropertyRange(described_in Sequence)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Sequence
SubClassOf(Sequence ObjectMinCardinality(1 has_part))
SubClassOf(Sequence ObjectAllValuesFrom(has_part Step))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Step
SubClassOf(Step ObjectAllValuesFrom(part_of Sequence))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Control_Flow_Step
SubClassOf(Control_Flow_Step Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Work_Unit_Step
SubClassOf(Work_Unit_Step Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Activity_Step
SubClassOf(Activity_Step ObjectAllValuesFrom(performs Activity))
SubClassOf(Activity_Step Work_Unit_Step)
SubClassOf(Activity_Step ObjectExactCardinality(1 performs))
SubClassOf(Activity_Step ObjectAllValuesFrom(is_a Work_Unit_Step))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Information_Object_Step
SubClassOf(Information_Object_Step Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#First_Step
SubClassOf(First_Step ObjectAllValuesFrom(is_a Control_Flow_Step))
SubClassOf(First_Step ObjectMinCardinality(1 followed_by))
SubClassOf(First_Step ObjectExactCardinality(0 come_after))
SubClassOf(First_Step Control_Flow_Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Final_Step
SubClassOf(Final_Step Control_Flow_Step)
SubClassOf(Final_Step ObjectMinCardinality(1 come_after))
SubClassOf(Final_Step ObjectExactCardinality(0 followed_by))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Join_Step
SubClassOf(Join_Step Sync_Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Fork_Step
SubClassOf(Fork_Step Sync_Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Work_Unit
EquivalentClasses(Work_Unit ObjectUnionOf(Activity Action))
SubClassOf(Work_Unit ObjectAllValuesFrom(part_of ObjectUnionOf(Task Activity)))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Activity
SubClassOf(Activity Work_Unit)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Role
SubClassOf(Role ObjectMinCardinality(1 has_part))
SubClassOf(Role ObjectAllValuesFrom(has_part Task))
SubClassOf(Role Collaborative_Entity)
SubClassOf(Role ObjectMinCardinality(1 played_by))
SubClassOf(Role ObjectMinCardinality(1 part_of Organization))
SubClassOf(Role ObjectAllValuesFrom(part_of Organization))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Decision_Step
SubClassOf(Decision_Step ObjectAllValuesFrom(is_a Control_Flow_Step))
SubClassOf(Decision_Step Control_Flow_Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Followed_by_Relation
SubClassOf(Followed_by_Relation Reified_Relation)
EquivalentClasses(Followed_by_Relation ObjectUnionOf(Object_Followed_by_Relation
Control_Followed_by_Relation))

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Action
SubClassOf(Action Work_Unit)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Merge_Step
SubClassOf(Merge_Step ObjectAllValuesFrom(is_a Control_Flow_Step))
SubClassOf(Merge_Step Control_Flow_Step)

// Class: http://localhost/amenitiesOWL_11/amenities.owl#Event
SubClassOf(Event ObjectAllValuesFrom(part_of Collaborative_System))
SubClassOf(Event Collaborative_Entity)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#produce
ObjectPropertyDomain(produce Work_Unit)
ObjectPropertyRange(produce Information_Object)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#trigger
InverseObjectProperties(receives trigger)
InverseFunctionalObjectProperty(trigger)
ObjectPropertyDomain(trigger Event)
ObjectPropertyRange(trigger Action)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#receives
InverseObjectProperties(receives trigger)
ObjectPropertyDomain(receives Action)
ObjectPropertyRange(receives Event)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#sends
ObjectPropertyDomain(sends Action)
ObjectPropertyRange(sends Event)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#evaluates
ObjectPropertyDomain(evaluates Followed_by_Relation)
ObjectPropertyRange(evaluates Guard)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#followed_by
InverseObjectProperties(come_after followed_by)
ObjectPropertyDomain(followed_by Step)
ObjectPropertyRange(followed_by Followed_by_Relation)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#following_step
ObjectPropertyDomain(following_step Followed_by_Relation)
ObjectPropertyRange(following_step Step)

// Object property: http://localhost/amenitiesOWL_11/amenities.owl#precede
TransitiveObjectProperty(precede)
ObjectPropertyDomain(precede Step)
ObjectPropertyRange(precede Step)

// Sub property chain axiom
SubObjectPropertyOf(SubObjectPropertyChain(following_step followed_by) precede)
```

Figura IV.10 Especificación en OWL de clases y relaciones para representar secuencias de unidades de trabajo

Sin embargo, este enfoque define implícitamente una relación terciaria (o 3-aria) para representar un flujo de trabajo entre dos *steps*, cuyos tres participantes serían el *step* de origen, el de destino y la guarda a comprobar. Consecuentemente, la relación de flujo entre dos *steps* ha de ser reificada en una nueva clase que hemos denominado *Followed_by_Relation*. Así, la relación *followed_by* conecta el *step* "origen" con la clase reificada *Followed_by_Relation*, y las instancias de esta clase conectarán con los *steps* "destino" y la guarda a evaluar en cada caso, por medio, respectivamente, de dos nuevas relaciones funcionales: *following_step* e *evaluates*. Para preservar la propiedad de modelo de árbol (ver

secciones 2.1.1 y 2.2), la relación *followed_by* se ha declarado en OWL como *InverseFunctionalProperty*, es decir,

No obstante, este esquema encaja mejor con el metamodelo de UML 2.0 para los DA, donde los arcos que representan el flujo entre dos nodos de actividad también se modelan por medio de una clase, denominada *ActivityEdge*, en lugar de hacerlo mediante una relación n-aria o una clase asociación. Por tanto, la conexión entre los dos metamodelos (el de UML y el que se propone en esta tesis en OWL), se facilita y se hace más directa ya que, la clase *ActivityEdge* se puede mapear directamente a la clase *Followed_by_Relation*, tal y como veremos en la siguiente sección.

Conexión con la ontología del marco conceptual de AMENITIES. El conjunto de clases y relaciones definido en la sección anterior no es más que una extensión, aunque necesaria, a la ya presentada ontología del marco conceptual para modelar las entidades del dominio en un entorno colaborativo (cfr. Figura IV.9). De ahí que también pueda verse como un esqueleto (p.e., *Sequence*, *First_Step*, *Activity_Step*, *Final_Step*, *Decision_Step*, *followed_by*, etc.) concebido con el propósito de poder especificar procesos colaborativos en la ontología subyacente al modelo de sistema colaborativo, así como instanciarse para representar flujos de trabajo concretos. Este enfoque también permite enfatizar la separación a nivel conceptual entre actividades y acciones, y su posterior disposición en el desarrollo de una tarea.

La relación *descrita_en* enlaza las nuevas clases para describir secuencias con las clases para describir un dominio colaborativo conectando una tarea o actividad con la secuencia de pasos establecidos para llevarla a cabo. En realidad, ambos conjuntos de clases se complementan uno a otro y producen un metamodelo de sistema colaborativo consistente, que ayuda a mejorar la descripción tanto de vistas estructurales, como de comportamiento de un sistema colaborativo.

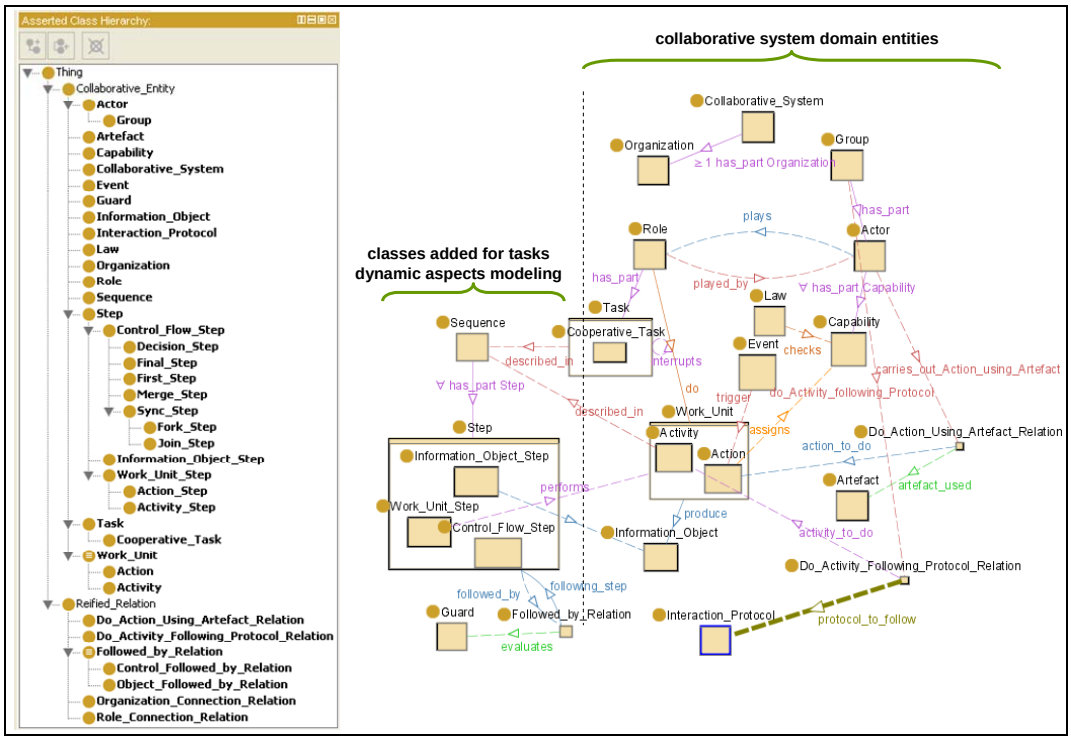


Figura IV.11 Ontología para representar sistemas colaborativos basados en AMENITIES

En la parte izquierda de la Figura IV.11 se muestra el conjunto de clases que resulta, y a la derecha una representación gráfica del mismo a partir de su especificación en OWL. No se han mostrado las subclases de *Control_Flow_Step*, *Work_Unit_Step* e *Information_Object_Step* para no complicar el diagrama. Por la misma razón, se han suprimido algunas otras relaciones de la ontología, como por ejemplo algunas relaciones inversas de las mostradas en la figura.

Correspondencia entre clases del metamodelo de UML para los DA y el conjunto de clases en OWL propuesto. Con objeto de poder representar los DA de UML 2.0 en ontologías en OWL para sistemas colaborativos, así como poder transformar unos modelos en otros pudiendo recuperar siempre las especificaciones de partida, se define formalmente el mapeo de la Tabla IV.1. En ella se representan el elemento del metamodelo de los DA, su símbolo específico en el diagrama si lo hubiera, la clase correspondiente en OWL, y por último, su semántica formal en Description Logics según la función de interpretación definida en la sección 5.2.2 del capítulo III. Las clases que acabamos de añadir a la ontología permiten cubrir el llamado *nivel intermedio* en la especificación *UML Superstructure*: "*el nivel intermedio soporta [...] el modelado de flujos concurrentes de control y de datos, y decisiones*" [OMG 2007].

Por otro lado, la revisión del metamodelo de UML para los DA ha permitido descubrir detalles de su diseño de difícil justificación. Por ejemplo, en los DA las instancias de la clase

Actividad (*Activity*) se representan por medio de instancias de la clase *ActivityNode*. De esta forma se separa una entidad de modelado (que en este caso sería una actividad concreta) de su representación gráfica y orden que ocupa dentro de un flujo de trabajo. Esto permite reutilizar dicha actividad y emplazarla en otros flujos de trabajo mediante otras instancias de la clase *ActiviyNode*. Este mismo enfoque se ha seguido en nuestra representación en OWL de los DA por medio de los conceptos *Actividad* y *Step*.

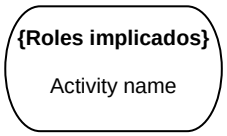
Sin embargo, el metamodelo de los DA de UML define la clase *Acción* (*Action*), que a nivel de modelado conceptual es una abstracción del mismo tipo que *Actividad*, como subclase de *ActivityNode*, un elemento gráfico, y como consecuencia, confunde elementos de distinta naturaleza en la misma jerarquía de clases. Nosotros hemos preferido mantener separadas las jerarquías de actividades y acciones de la de los *steps* [Noguera 2009].

Además, un simple mapeo entre ambos metamodelos reduciría la posibilidad de habilitar mecanismos de razonamiento automáticos basados en Description Logics a partir de la definición de ciertas restricciones y características de las relaciones. Este punto se abordará en el siguiente capítulo.

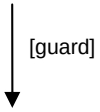
Por último, conviene destacar que los DA fueron introducidos en AMENITIES con una pequeña variación según la cual, en lugar de utilizar *swimlanes* para delimitar particiones en el diagrama en función de los roles a los que se asigna cada actividad, dichos roles se escriben separados por comas encima del nombre de la actividad. Esto es así debido a que la naturaleza colaborativa de los procesos modelados conlleva que con frecuencia se asigne varios para la realización de las distintas actividades y sea difícil delimitar particiones sin que se solapen unas con otras. La nueva versión 2.0 de UML incorpora una notación similar permitiendo representar las particiones entre llaves y separadas por comas dentro de los *ActivityNode*, justo encima del nombre de la actividad correspondiente [OMG 2007].

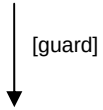
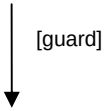
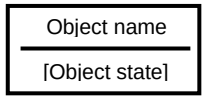
Tabla IV.1 Correspondencia entre elementos del metamodelo de un DA y entidades en la ontología definida

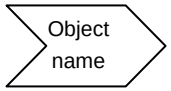
Elemento del DA basado en UML de AMENITIES	**Símbolo**	**Clase/Propiedad en la Ontología**	**Especificación en DL**
Initial node	●	Clase *First_Step*	First_Step $\sqsubseteq$ Control_Flow_Step First_Step $\sqsubseteq$ ($\geqslant$0 followed_by^{-}) $\sqcap$ ($\leqslant$0 followed_by^{-}) First_Step $\sqsubseteq$ ($\geqslant$1 followed_by) Control_Flow_Step $\sqsubseteq$ Step Step $\sqsubseteq$ $\forall$part_of.Sequence
Activity o Action, y Role	--	Clase *Work_Unit* (una *Activity* o una *Action*) Clase *Role*	Activity $\sqsubseteq$ Work_Unit Action $\sqsubseteq$ Work_Unit Role $\sqsubseteq$ $\forall$do.Work_Unit

			Role $\sqsubseteq$ ($\geqslant$1 part_of.Organization)
ActivityNode o ActionNode		Clase *Activity_Step* o *Action_Step*	Work_Unit_Step $\sqsubseteq$ Step Activity_Step $\sqsubseteq$ Work_Unit_Step Action_Step $\sqsubseteq$ Work_Unit_Step $\top \sqsubseteq \forall do^{-}$.Work_Unit $\sqcap \forall do$.Role
DecisionNode		Clase *Decision_Step*	Decision_Step $\sqsubseteq$ Control_Flow_Step Decision_Step $\sqsubseteq$ ($\geqslant$2 followed_by)
MergeNode		Clase *Merge_Step*	Merge_Step $\sqsubseteq$ Control_Flow_Step Merge_Step $\sqsubseteq$ ($\geqslant$1 followed_by) $\sqcap$ ($\leqslant$1 followed_by) Merge_Step $\sqsubseteq$ ($\geqslant$2 followed_by^{-})

Send-Signal ActionNode		Clase *Send_Signal_Action_Step*, clase *Action*, clase *Action_Step*, clase *Event* y propiedad *send*	Send_Signal_Action_Step ≡ Action_Step ⊓ ∃performs.Send_Signal_Action Send_Signal_Action ⊑ Action Send_Signal_Action ≡ Action ⊓ ∃send.Event
Receive-Signal ActionNode		Clase *Send_Signal_Action_Step*, clase *Action*, clase *Action_Step*, clase *Event* y propiedad *receive*	Receive_Signal_Action_Step ≡ Action_Step ⊓ ∃performs.Receive_Signal_Action Receive_Signal_Action ⊑ Action Receive_Signal_Action ≡ Action ⊓ ∃receive.Event
ForkNode/JoinNode		Clase *Fork_Step* y clase *Join_Step*	Fork_Step ⊑ Sync_Step

			Fork_Step $\sqsubseteq$ ($\geqslant$1 followed_by^{-}) $\sqcap$ ($\leqslant$1 followed_by^{-}) Join_Step $\sqsubseteq$ Sync_Step Join_Step $\sqsubseteq$ ($\geqslant$1 followed_by) $\sqcap$ ($\leqslant$1 followed_by) Sync_Step $\sqsubseteq$ Control_Flow_Step
Flow/Edge (Clase ActivityEdge en UML 2.0)		Property *followed_by* of clase *Step*, clase *Guard*, clase *Followed_by_Relation*, propiedad *evaluate* y propiedad *following_step*	Followed_by_Relation $\sqsubseteq$ ($\geqslant$0 evaluate) $\sqcap$ ($\leqslant$1 evaluate) $\top \sqsubseteq \forall$evaluate.Guard $\sqcap$ $\forall$evaluate^{-}.Followed_by_Relation $\top \sqsubseteq \forall$following_step.Step $\sqcap$ $\forall$following_step^{-}.Followed_by_Relation

ControlFlow		Clase *Control_Followed_by_Relation*	Control_Followed_by_Relation ⊑ Followed_by_Relation Control_Followed_by_Relation ⊑ (⩾0 following_step.Information_Object_Step)
ObjectFlow		Clase *Object_Followed_by_Relation*	Object_Followed_by_Relation ⊑ Followed_by_Relation Object_Followed_by_Relation ⊑ (⩾0 following_step.Work_Unit_Step)
Object	--	Clase *Information_Object*	Information_Object ⊑ ∀part_of.Cooperative_System
ObjectNode		Clase *Information_Object_Step* y propiedad *is_in_state*	Information_Object_Step ⊑ Step ⊤ ⊑ ∀is_in_state.State ⊓ ∀is_in_state⁻. Information_Object_Step

Signal ObjectNode		*Clase Information_Object*, clase *Information_Object_Step*, clase *Action*, clase *Event*, propiedad *send* y (la inversa de la) propiedad *produce*	$\text{Signal\_Object\_Step} \sqsubseteq \text{Information\_Object\_Step}$ $\text{Signal\_Object\_Step} \equiv \exists \text{produce}^{-}.\text{Send\_Signal\_Action}$
Final Node		Clase *Final_Step*	$\text{Final\_Step} \sqsubseteq \text{Control\_Flow\_Step}$ $\text{Final\_Step} \sqsubseteq (\geqslant 0\ \text{followed\_by}) \sqcap (\leqslant 0\ \text{followed\_by})$ $\text{Final\_Step} \sqsubseteq (\geqslant 1\ \text{followed\_by}^{-})$
El símbolo $\sqsubseteq$ denota subsunción y se interpreta como una inclusión de conjuntos $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. El símbolo $\equiv$ denota equivalencia de conjuntos, por tanto, $C \equiv D$ se interpreta como $C^{\mathcal{I}} = D^{\mathcal{I}}$.			

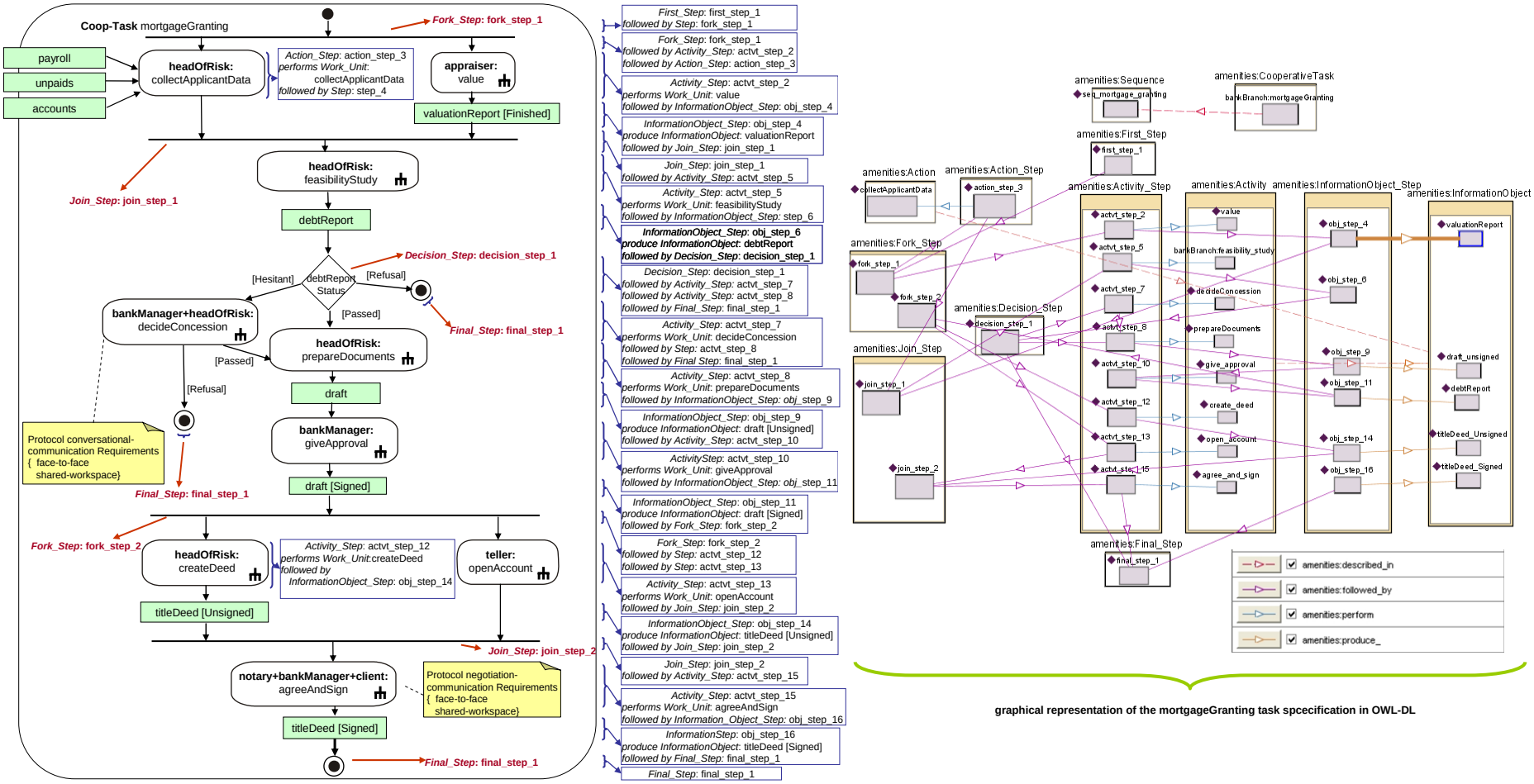


Figura IV.12 Representación de la tarea *concesiónDeHipoteca* en OWL con el conjunto de clases propuesto

Implementación de un caso concreto. Con objeto de mostrar un ejemplo de aplicación de la propuesta, se ha especificado en OWL el DA correspondiente a la tarea *concesiónDeHipoteca* (cfr. sección 3.7.3 y Figura II.7 del capítulo II), y se ha elaborado una figura para ilustrar los elementos de las dos representaciones.

Así, en la parte izquierda de la Figura IV.12 hemos vuelto a representar el DA correspondiente a la tarea *concesiónDeHipoteca*.

Asimismo, en la parte central, hemos tratado de esbozar la descripción ontológica de cada una de las fases por las que atraviesa dicha tarea. Por motivos de claridad y espacio en la figura, se ha utilizado un lenguaje ad-hoc más conciso que el de las sintaxis RDF/XML o funcional OWL. También se ha omitido de esta descripción ontológica la especificación de las guardas y los arcos entre nodos (lo que se haría por medio de instancias de la clase *Followed_by_Relation*). Para ello, se ha considerado que dos *steps* se conectan simplemente mediante instancias de la relación *followed_by*. Los recuadros y llaves azules destacan las descripciones de la ontología (p.e. *First_Step: first_step_1 followed_by Step: step_1*).

En la parte derecha de la Figura IV.12 se muestra la representación gráfica de la tarea a partir de su especificación en OWL. Los identificadores dados a cada paso de la secuencia son irrelevantes (*first_step_1*, *step_1*, *step_2*, etc.), aunque hemos tratado de asignarlos utilizando sufijos crecientes para facilitar su trazabilidad en el diagrama. Lo mismo ocurre con los nombres dados a los puntos de sincronización y las bifurcaciones (*decisions*). Se ha utilizado el color rojo para señalar la correspondencia entre los elementos del DA y los de la ontología, como por ejemplo ocurre en el caso de *fork_step_2* y *decision_step_1*. Las flechas y recuadros azules, y las flechas, llaves y etiquetas rojas no forman parte de los DA. Por su parte, los nombres de las unidades de trabajo (p.e. *createDeed*, *openAccount*, *agreeAndSign*, etc.) sí son relevantes y deben corresponder a instancias de actividades o acciones del marco conceptual de AMENITIES.

El potencial lugar de un *step* en la secuencia viene determinado por los *steps* a los que sigue y los *steps* por que puede ser *seguido* (*followed_by*). El orden en que de hecho se ejecuta un *step* depende de cada ejecución concreta de la tarea en la que está involucrado, ya que, antes de pasar al siguiente, en ciertos *steps* se tiene que elegir entre diferentes caminos alternativos en función del valor de las guardas asociadas a los arcos (p.e. la evaluación de dichas guardas en los *Decision_Steps* puede conllevar que el flujo de control varíe de una ejecución a otra).

2.4 Ontologías de aplicación: descripción de sistemas colaborativos de dominios específicos basados en AMENITIES

La estructura de clases presentada hasta el momento conforma un metamodelo con el que guiar el desarrollo de ontologías de sistemas colaborativos concretos basados en AMENITIES, así como, validar tales descripciones y facilitar la interoperatividad entre sistemas descritos siguiendo este marco de trabajo.

La descripción de estos sistemas particulares se realizará completando la ontología que ya hemos presentado con clases y relaciones más específicas para cada entorno colaborativo concreto (una sucursal bancaria, un aeropuerto, un departamento universitario, etc.). Estas descripciones ontológicas más específicas conformarán las que se conocen como *ontologías de aplicación* y complementan a las del primer tipo (*ontologías de dominio*) [Guarino 1998].

Así, tomando como referencia la ontología de dominio definida para el marco conceptual de AMENITIES, una organización puede encontrar útil subclasificar los *Roles* atendiendo a distintos perfiles como, por ejemplo, *Ejecutivo*, *Administrativo* o *Contable*, y asignar sus tareas consecuentemente. Asimismo, cada organización puede establecer su propia clasificación para los *Objetos_de_Información*. Por ejemplo, en una *Sucursal Bancaria* (un ejemplo de subclase de *Organización* para una entidad bancaria) puede ser útil distinguir entre *Facturas*, *Nóminas*, *Informes_de_Deudas*, etc. En cambio, para una *Notaría* (otra clase de organización) podría ser más útil definir un tipo de *Objetos_de_Información* especial como *Escritura*, y dentro de los de este tipo distinguir a su vez, entre escrituras correspondientes a *Fincas*, *Locales_Comerciales*, *Viviendas*, etc.

Lo mismo cabe aplicar con cualesquiera otras especializaciones de los conceptos de la ontología de dominio de AMENITIES. Un caso particular resulta cuando se especializan *Tareas* y *Unidades_de_Trabajo*, y que da lugar a lo que algunos autores denominan *ontologías de tareas* [Guarino 1998][Samoylov 2005]. Sin embargo, en el enfoque que venimos desarrollando las *ontologías de tareas* aparecerían a un nivel menos *general* que las *ontologías de dominio*, como adoptan otros trabajos [Xia 2007], y que difiere de lo propuesto por Guarino que las considera en el mismo nivel (cfr. Figura III.2 en capítulo III). No obstante, por centrarse nuestra propuesta en el modelado de sistemas colaborativos, no parece que tenga mucho sentido especificar por separado las definiciones acerca de las tareas de las descripciones de las organizaciones donde se van a llevar a cabo.

En el ámbito de la investigación en ontologías es frecuente distinguir entre la *ontología*, haciendo referencia a un conjunto de clases y relaciones, y la *base de conocimiento* asociada, significando un conjunto de instancias de dichas clases y relaciones. No obstante, aunque a veces incidan en tal distinción, los distintos autores reconocen que la línea que las separa es

muy estrecha [Gómez-Pérez 1999][Noy 2001]. De hecho, a la hora de implementar una ontología y su base de conocimiento asociada, buena parte de los lenguajes y herramientas de soporte consideran los elementos de ambas como parte de una misma *ontología*, es decir, sin distinguir entre una u otra, y concibiendo la ontología como compuesta de clases, relaciones e instancias de ellas. Tal es el caso también del lenguaje OWL [W3C OWL 2004].

Las instancias individuales se corresponden con los conceptos más específicos de una base de conocimiento [Noy 2001]. Por este motivo, dentro del diseño ontológico de un sistema colaborativo, nosotros también incluiremos la definición de las instancias de las clases y relaciones dentro de las ontologías de aplicación.

Así, en estas ontologías se declararán instancias concretas de las clases *Rol*, *Tarea*, *Organización*, etc. Por ejemplo, continuando con el sistema colaborativo de concesión de hipotecas donde aparecían involucradas una sucursal bancaria, una notaría y una oficina de tasación, podemos pensar, en principio, en tres ontologías de aplicación: una por cada organización. En la ontología correspondiente a la oficina bancaria se puede declarar la instancia *sucursal_nº5* como instancia de la clase *Sucursal_Bancaria* que, a su vez, se define como subclase de *Organización*, y que *consta_de* (instancias de una relación) los roles *cajero*, *jefeDeRiesgos* y *director*. En la ontología de aplicación para la notaría se definirían los roles *notario* y *agente*. Y para la oficina de tasación los roles *jefe* de la oficina y *tasador*.

Las ontologías de aplicación también podrían recoger cómo tiene lugar el desarrollo de tareas concretas. Por ejemplo, la ontología de aplicación de la *sucursal_nº15* podría contener la descripción de la tarea *concesiónDeHipoteca* que mostramos en la sección 2.3.2. Asimismo, este ejemplo concreto muestra que el hecho de tratar con instancias individuales de clases (en este caso *Unidades de Trabajo*), no implica que la ontología no se pueda reutilizar. Así, la descripción de la secuencia de pasos en los que se desarrolla la concesión de una hipoteca podría ser utilizada por otras sucursales de la misma entidad bancaria.

Finalmente, conviene no olvidar que en conjunto, desde un punto de vista lógico, todas las ontologías (la ontología de dominio que formaliza el marco conceptual de AMENITIES y las otras ontologías de aplicación que se puedan derivar de la primera) conforman una única ontología del sistema colaborativo de concesión de hipotecas. Sin embargo, a nivel físico cada ontología estará almacenada en un fichero distinto. Esta modularización de la ontología global forma parte de un esquema de diseño de ontologías estructurado en capas según el nivel de especificidad de las entidades modeladas y que pasamos a ver con mayor detalle en la sección siguiente.

3. INGENIERÍA DE ONTOLOGÍAS

Uno de los beneficios que reporta el uso de ontologías, suele ser el hecho de poder importar y/o establecer correspondencias entre clases definidas en ontologías diversas y, de este modo, poder reutilizarlas (si son equivalentes, si una incluye a otra, si son antagónicas,…) [Kalfoglou 2003].

Por otro lado, conviene no olvidar que el desarrollo de una ontología es siempre un proceso iterativo sujeto a numerosas revisiones de los axiomas declarados inicialmente. Por ejemplo, clases que se refinan en otras subclases, clases que resultan redundantes y/o innecesarias, clases o instancias que se encuentran equivalentes, instancias que encajan mejor como clases o viceversa etc. [Fdez-López 2002][Noy 2001].

En el caso concreto de OWL, el lenguaje proporciona una serie constructores específicos para la definición de comentarios, control de versiones e importación de ontologías (p.e. *rdfs:comment*, *owl:priorVersion*, *owl:imports*, *owl:sameAs*, etc.), que pueden usarse combinadamente con el poder de inferencia de razonadores basados en Description Logics [W3C OWL 2004]. Así, la propia descripción de la ontología permite especificar su correspondencia con otras ontologías.

En el contexto de los sistemas colaborativos, esto permite, por ejemplo, que varias organizaciones puedan comparar si las funciones atribuidas a un rol "*jefeDeProyectos*" por dos organizaciones son iguales o no, recuperar información acerca los miembros de otras organizaciones y así, definir grupos interorganizacionales, o reutilizar la descripción de procesos entre varios entornos colaborativos específicos, como acabamos de ver en la sección anterior para el caso de la tarea *concesiónDeHipoteca*.

Sin embargo, parte del beneficio derivado de la reutilización de ontologías se ve contrarrestado por el tiempo empleado por el diseñador de la nueva ontología en definir las correspondencias descritas anteriormente. En efecto, el mapeo entre ontologías no es una labor que pueda realizar automáticamente una computadora o de forma "desatendida" (del inglés *unattended*), sino que requiere ser realizado bajo la supervisión de un analista. Otro problema relacionado con la reutilización es la posible aparición de inconsistencias como resultado de integrar en el sistema de información de una organización datos de fuentes externas [Henderson 2004].

3.1 Niveles de Diseño para Crear Ontologías

Conscientes de la importancia que, en general, tiene la reutilización de ontologías presentamos un esquema de diseño de ontologías para sistemas colaborativos partiendo del proceso de desarrollo descrito en la sección 2. Si antes nos hemos centrado en el proceso

de creación de una ontología, esto es, su contenido, ahora nos vamos a fijar en su arquitectura.

Hasta el momento, el enfoque que hemos adoptado describe un sistema colaborativo mediante una ontología de dominio y varias ontologías de aplicación, y concuerda parcialmente con el propuesto por Guarino [Guarino 1998], que distingue entre varios niveles de generalidad que dan lugar a distintos tipos de ontologías (cfr. Figura III.2). En nuestro caso, el dominio es el de los sistemas CSCW.

Para ello, primero se ha formalizado el marco conceptual de AMENITIES mediante una ontología de dominio y, a partir de ahí, se han comenzado a definir ontologías concretas para cada tipo de organización de que se trate.

Con este esquema, cada vez que se pretende crear un nueva tarea o sistema colaborativo y que involucre a organizaciones distintas, la ontología que lo describa únicamente tiene que definir aquellos elementos que realmente sean "nuevos" en el sistema (por ejemplo, una nueva tarea cooperativa, un nuevo artefacto,…) y sus relaciones con las entidades ya definidas (por ejemplo, las organizaciones y los actores que ya existían).

En una herramienta de edición de ontologías para OWL como Protégé, las definiciones de los demás elementos pueden referenciarse e incluirse en la nueva ontología, tal y como están, de forma transparente [Noguera 2006]. Para ello, la herramienta hará uso internamente de algunos constructores ya mencionados, como *owl:imports*, así como de las URIs de dichos elementos.

3.2 Modularización de la Ontología

Según el planteamiento de las secciones anteriores, en el nivel más alto podríamos situar el documento que recoge el esquema mostrado en la Figura IV.11, esto es, la ontología de dominio.

Por lo que se refiere al nivel de las ontologías de aplicación, creemos que para un diseño más modular, que facilite el mantenimiento y evolución de la ontología, es aconsejable diferenciar dos niveles más. Esta distinción la hemos considerado atendiendo fundamentalmente al dinamismo o frecuencia con que pueden cambiar las entidades en un nivel u otro, y a la especificidad con respecto al sistema descrito (y en consecuencia, con respecto al nivel al que pueden reutilizarse).

Por ejemplo, en una sucursal bancaria el rol "*empleadoEnPrácticas*" puede cambiar su definición con mayor frecuencia que el rol "*jefeDeRiesgos*". Por otro lado, aunque es difícil generalizar, parece claro que cuestiones como la descripción de los actores o del estado

actual del sistema (qué actores están desempeñando actualmente qué roles), sólo tienen sentido en la ontología de aplicación del entorno colaborativo donde trabajan. Por ejemplo, si "*Julie_Knowles*" (instancia de la clase *Actor*) es la "*jefeDeRiesgos*" (un *Rol*) de la sucursal "*sucursal_nº15*", tal especificación no sirve para otra "sucursal" distinta y, por tanto, debe especificarse en un segundo nivel más bajo dentro del nivel de la ontología de aplicación. En cambio, las definiciones del rol "*jefeDeRiesgos*" o la tarea "*concesiónDeHipoteca*" junto con la secuencia de pasos en que se realiza, sí pueden servir para varias sucursales de la misma entidad financiera y, en consecuencia, debería especificarse en una ontología de aplicación de nivel superior.

En la Figura IV.13 se muestra un esquema con la estructura que presentarían las ontologías construidas siguiendo esta filosofía.

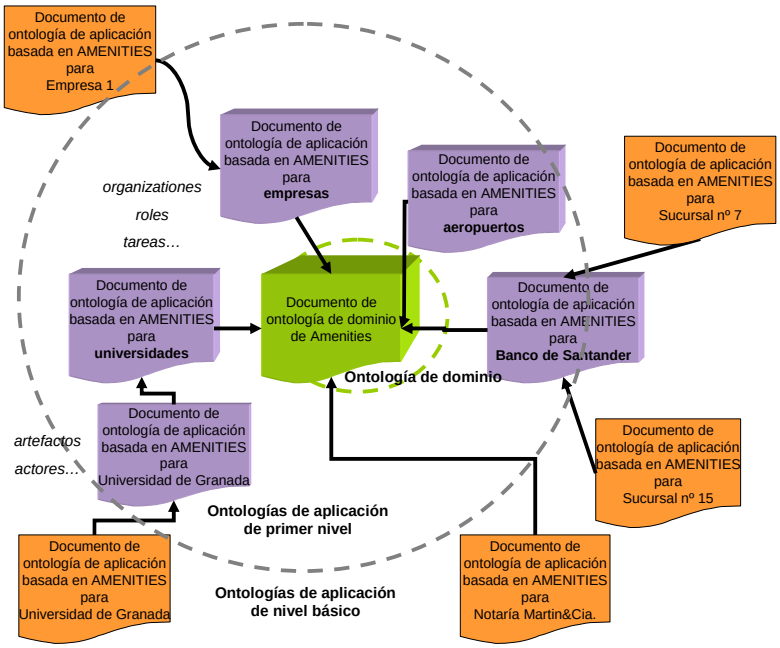


Figura IV.13 Arquitectura de ontologías basadas en AMENITIES

Se puede observar cómo este esquema modular también favorece que la intervención sobre la ontología del sistema pueda realizarse a distintos niveles de forma poco "intrusiva" [Andrade 2002].

La inclusión/eliminación/modificación de un elemento al modelo colaborativo puede realizarse al nivel de la ontología de dominio, como fueron en su momento la adición de clases adicionales para representar secuencias de unidades de trabajo, o el cambio de nomenclatura de la clase *Subactividad* por *Actividad* [Noguera 2009]. En este último caso, también resultan de gran ayuda los mecanismos de "*refactoring*" que proporcionan las

herramientas de edición de ontologías y que permiten renombrar entidades a través de varios archivos [Seidenberg 2007].

La inclusión/eliminación/modificación de un nuevo rol genérico para cierto tipo de sistemas puede realizarse en una ontología de aplicación de primer nivel. Modificar el rol que desempeña el actor "*Julie_Knowles*" puede llevarse a cabo a través de la ontología de aplicación de nivel más básico o específico.

Por otra parte, el mapeo entre ontologías de diferentes sistemas también es más fácil, ya que todas se ajustarían al patrón de sistema colaborativo propuesto en AMENITIES.

Finalmente, la separación de niveles que se presenta no pretende únicamente clasificar los conceptos y relaciones según el carácter más o menos reusable/general que posean. Al contrario, esta distinción también permite definir e inferir relaciones más complejas entre conceptos conforme las ontologías van siendo más específicas, con objeto de llevar a cabo procesos de razonamiento y adquirir conocimiento más refinado. Esta cuestión se abordará en el siguiente capítulo sobre razonamiento.

3.3 Ciclo de vida de desarrollo de la ontología y evolución

A la par que los sistemas que modelan, las ontologías evolucionan con ellos a lo largo del tiempo [Sure 2005]. Esta evolución se produce a todos los niveles, tanto de ontologías de dominio como de aplicación, y puede deberse a cambios en los requisitos iniciales del sistema, aparición de nuevas tecnologías, cambios en el entorno colaborativo, etc. Ello obliga a revisar las declaraciones formuladas inicialmente.

En su historia, la ontología de dominio del marco conceptual de AMENITIES y las ontologías desarrolladas a partir de ésta han pasado por diversas modificaciones, algunas de las cuales ya han sido comentadas, y también se espera que en el futuro continúe revisándose. Precisamente el diseño modular nos ha permitido hacer evolucionar la ontología y adaptarla a distintas necesidades más fácilmente [Duque 2009].

Muchas de ellas han tenido que ver la especificación de los modelos y vistas de comportamientos propuestos en la metodología, en los que implícitamente se describen relaciones entre conceptos que no aparecen definidos previamente en el marco conceptual como, por ejemplo, la ya comentada del orden entre actividades. Otro ejemplo sería el hecho de que inicialmente todo actor comienza desempeñando un rol en el sistema. Esto supone una relación, que hemos definido como *rol_inicial*, cuyo dominio son instancias de la clase *Actor* y cuyo rango son instancias de la clase *Rol*.

Otro cambio acometido, por sugerencia del Dr. Chung, fue definir la clase *Grupo* como subclase de la clase *Actor*, ya que, a veces todo un grupo se comporta o asume funciones como si se tratara de un sólo actor en el sistema. En la parte izquierda de la Figura IV.11 puede verse implementado este cambio en la jerarquía.

Las evoluciones siguientes se enfocarán a proporcionar la infraestructura de clases y relaciones necesaria que habilite un desarrollo dirigido por modelos especificados formalmente en ontologías. Las ontologías de dominio y aplicación corresponden a los denominados *Modelos Independientes de Computación* (CIM) en la filosofía MDA (con frecuencia olvidados en la literatura), y conforman modelos conceptuales de alto nivel de abstracción que representan entornos colaborativos concretos. Los elementos que describen no siempre tendrán su correspondencia en un componente tecnológico (*groupware*) concreto. Basta pensar, por ejemplo, en los actores del sistema o en las organizaciones para las que trabajan.

Por tanto, el siguiente paso para establecer un proceso de desarrollo dirigido por modelos/ontologías requerirá completar la ontología de dominio actual para delimitar qué entidades de la ontología se van a *implementar*, con qué elementos de las interfaces del sistema interactúan los usuarios del sistema colaborativo, etc. [Pahl 2005]. En definitiva, se trata de reducir el *gap* entre los modelos del mundo real y los modelos tecnológicos, que en MDA se corresponden con los Modelos Independientes de Plataformas (PIM) y los Modelos Específicos de Plataformas (PSM) [OMG 2003].

Otro campo de extensión y evolución de las ontologías presentadas consiste en definir sus conexiones con ontologías que describan los servicios que den soporte a las tareas a llevar a cabo, así como, las interdependencias entre ellos: si son compatibles, si han de ejecutarse unos antes que otros, etc. [W3C ODA 2006].

Finalmente, tal y como demandan los enfoques de desarrollo dirigidos por modelos, es precisa otra ontología, ortogonal a todas las mencionadas anteriormente, que agrupe las entidades modeladas en distintas *vistas* del sistema, como las que propone AMENITIES (aunque aún podrían definirse más vistas). Esta ontología podrá ser subsiguientemente utilizada por herramientas de edición para presentar la información de forma resumida y organizada por *concerns* o aspectos del sistema.

4. CONCLUSIONES DEL CAPÍTULO

Sintetizando los postulados defendidos en el borrador del *W3C Semantic Web Best Practices and Deployment Working Group* [W3C ODA 2006] que vimos al final del capítulo III, básicamente podemos resumir los enfoques ODA con la siguiente ecuación:

ODA = MDE + BPM + Web Semántica (ec. 1)

En este capítulo hemos presentado una propuesta para la formalización mediante ontologías de los modelos propuestos por la metodología AMENITIES para describir un sistema colaborativo. Para ello, se ha seguido un proceso de desarrollo de ontologías concreto basado en el propuesto por Noy y McGuinness en [Noy 2001], y se han adoptado una serie de patrones de diseño para ontologías [Gangemi 2007] con objeto de representar relaciones entre conceptos para los que el lenguaje OWL no da soporte de forma nativa.

Para ello, se han tomado como referencia los modelos basados en UML de los que hace uso AMENITIES. Asimismo, se han establecido correspondencias adecuadas entre ciertos elementos del metamodelo de UML, como los implicados en la especificación de diagramas de actividades, a un conjunto de clases en OWL para permitir capturar flujos de tareas en este último lenguaje.

Las ontologías resultantes constituyen modelos del sistema de alto nivel de abstracción –CIM–, según la filosofía MDA. Como resultado de aplicar nuestro enfoque al actualizar la ecuación anterior (ec. 1) obtenemos una expresión de la forma:

ODA = MDA + UML + OWL (ec. 2)

Por último, se ha propuesto un esquema de diseño de la arquitectura de la ontología del sistema que facilita la evolución y la reusabilidad de las entidades descritas a distintos niveles de detalle.

REFERENCIAS DEL CAPÍTULO

[Andrade 2002] Andrade, L.F., Fiadeiro, J.L.: "Agility through Coordination". Information Systems, 27 (2002), Elsevier, 411-424

[Barjis 2008] Barjis, J.: "The Importance of Business Process Modeling in Software Systems Design". Science of Computer Programming, Vol. 71, 1, 2008, pp. 73-87

[Berardi 2005] Berardi, D., Calvanese, D., De Giacomo, G.: "Reasoning on UML class diagrams". Artificial Intelligence 168 (1–2) (2005), pp. 70-118

[Calvanese 2001] Calvanese, D., De Giacomo, G., Lenzerini, L.: "Identification constraints and functional dependencies in description logics". Proc. of the 17th Int. Joint conf. on Artificial Intelligence (IJCAI 2001), Seattle, WA, 2001, pp. 155-160

[Calvanese 2007] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R: "Can OWL model football leagues?". In Proc. of the 2007 International Workshop on OWL: Experiences and directions (OWLED 2007), 2007

[Calvanese 2008] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. "Path-based identification constraints in description logics". In Proc. of KR 2008, pp. 231-241

[Drummond 2006] Drummond, N., Rector, A.L., Stevens, R., Moulton, G., Horridge, M., Wang, H, Seidenberg, J.: "Putting OWL in Order: Patterns for Sequences in OWL", Proceedings of the 2nd OWL Experiences and Directions Workshop (OWL-ED 2006), Athens, Georgia, USA, 2006

[Duque 2009] Duque, R., Noguera, M., Bravo, C., Garrido, J.L., Rodríguez, M.L.: "Construction of interaction observation systems for collaboration analysis in groupware applications". Advances in Engineering Software, Elsevier, 2009. doi: 10.1016/j.advengsoft.2009.01.028

[Evermann 2005] Evermann, J., Wand, Y.: "Toward Formalizing Domain Modeling Semantics in Language Syntax". IEEE Transactions on Software Engineering, vol. 31, no. 1, pp. 21-37, January 2005

[Evermann 2008] Evermann, J.: "A UML and OWL Description of Bunge's Upper-level Ontology Model". Software and Systems Modeling. In press (doi:10.1007/ s10270-008-0082-3)

[Fdez-López 2002] Fernández-López, M., Gómez-Pérez, A.: "Overview and analysis of methodologies for building ontologies". The Knowledge Engineering Review, Vol. 17:2, 2002, Cambridge University Press, 129-156

[Fox 1998] Fox, M., Barbuceanu, M., Gruninger, M., Lin., J.: "An Organizational Ontology for Enterprise Modeling". In Simulating Organizations: Computational Models of Institutions and Groups. AAAI Press / MIT Press, 1998, pp. 131-152

[Gamma 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA (1995)

[Gangemi 2007] Gangemi, A., Gómez-Pérez, A., Presutti, V., Suárez-Figueroa, M.C.: "Towards a Catalog of OWL-based Ontology Design Patterns", CAEPIA 07, Gangemi, (2007-11)

[Garrido 2003] Garrido, J.L.: "AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas". Tesis Doctoral, Granada 2003

[Garrido 2005] Garrido, J.L., Gea, M., Rodríguez, M.L.: "Requirements Engineering in Cooperative Systems". In: Requirements. Engineering for Sociotechnical Systems, Chapter XIV, IDEA GROUP, USA, 2005, pp. 226-244

[Gómez-Pérez 1999] Gómez-Pérez, A.: "Tutorial on Ontological Engineering". En IJCAI'99

[Gruber 1993] Gruber, T. R.: "A translation approach to portable ontology specifications". Knowledge Acquisition 5, 1993, pp. 199-220

[Guarino 1998] Guarino, N.: "Formal Ontology in Information systems". Proceedings of FOIS '98, (Trento, Italy, June, 1998). IOS Press, Amsterdam, 1998, pp. 3-15

[Henderson 2004] Henderson, P., Crouch, S., Walters, R. J.: Information Invasion in Enterprise Systems: Modelling, Simulating and Analysing System-level Information Propagation. 6th International Conference on Enterprise Information Systems (ICEIS 2004) 1, pp. 473-481

[Horrocks 2001] Horrocks, I., Sattler, U.: "Ontology reasoning in the SHOQ(D) description logic". In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), 199-204, 2001

[Hurtado 2002] Hurtado, M.V.: "Un Modelo de Integración Evolutivo entre Sistemas de Información y Sistemas de Ayuda a la Decisión". Tesis Doctoral, Granada 2002

[Jarrar 2007] Jarrar, M.: "Mapping ORM into the SHOIN/OWL Description Logic - Towards a Methodological and Expressive Graphical Notation for Ontology Engineering". LNCS, Vol. 4805, 2007, pp. 729-741

[Kalfoglou 2003] Kalfoglou, Y., Schorlemmer, M.: "Ontology mapping: the state of the art". Knowl. Eng. Rev. 18, 1 (Jan. 2003), pp. 1-31

[Keet 2007] Keet, C.M.: "Mapping the Object-Role Modeling language ORM2 into Description Logic language $\mathcal{DLR}_{ifd}$". Technical Report KRDB07-2, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy. 15 February 2007. arXiv:cs.LO/0702089v1

[Knublauch 2004] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.: "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", LNCS 3298, 2004, pp. 229-243

[Lange 2006] Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: "UML Software Architecture and Design Description". IEEE Software, 23 (2006), pp. 40-46

[Loom 1995] Tutorial for Loom version 2.1., http://www.isi.edu/isd/LOOM/documentation/tutorial2.1.html, May 1995

[Motik 2008] Motik, B., Cuenca Grau, B., and Sattler, U.: "Structured objects in owl: representation and reasoning". Proc. of the 17th International Conference on World Wide Web (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 555-564

[Noguera 2006] Noguera, M., Hurtado, M.V., Garrido, J.L.: "An Ontology-Based Scheme Enabling the Modeling of Cooperation in Business Processes", LNCS 4277, Springer-Verlag (2006), pp. 863-872

[Noguera 2009] Noguera, M., Hurtado, M.V., Rodríguez, M.L., Chung, L., Garrido, J.L.: "Ontology-driven Analysis of UML-Based Collaborative Processes using OWL-DL and CPN". Science of Computer Programming, (in press, under review), 2009

[Noy 2001] Noy, N.F., McGuinness, D.L.: "Ontology development 101: A guide to creating your first ontology". Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001

[OMG 2003] OMG, "MDA Guide Version 1.0.1". OMG Document Number: omg/03-06-01. Available at: http://www.omg.org/docs/omg/03-06-01.pdf

[OMG 2007] OMG, "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[Pahl 2005] Pahl, C.: "Ontology Transformation and Reasoning for Model-Driven Architecture". CoopIS/DOA/ODBASE 2005, LNCS 3761, 2005, pp. 1170-1187

[Parsia 2008] Parsia, B., Sattler, U., Schneider, T.: "Easy Keys for OWL". In Proc. of the 2008 International Workshop on OWL: Experiences and directions (OWLED 2008)

[Penichet 2008] Penichet, V.M.R., Lozano, M.D., Gallud, J.A.: "An Ontology to Model Collaborative Organizational Structures in CSCW Systems". International Chapter Book: Engineering the User Interface. From Research to Practice. Ed. Springer, 2008, pp. 127-139

[Plisson 2007] Plisson, J., Ljubic, P., Mozetic, I., Lavrac, N.: "An Ontology for Virtual Organization Breeding Environments". IEEE Transactions on Systems, Man, and Cybernetics, Part C 37(6), 2007, pp. 1327-1341

[Russell 2006] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wohed, P.: "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling", APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual Modelling, Australian Computer Society, 53, 2006, pp. 95-104

[Samoylov 2005] Samoylov, V., Gorodetsky, V.: "Ontology Issue in Multi-agent Distributed Learning". AIS-ADM 2005, LNAI 3505, 2005, pp. 215-230

[Seidenberg 2007] Seidenberg, J., Rector, A.L.: "The State of Multi-User Ontology Engineering". WoMO 2007

[Seidewitz 2003] Seidewitz, E.: "What models mean". IEEE Software 20 (2003), pp. 26-32

[Sure 2005] Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., Oberle, D.: "The SWRC ontology - semantic web for research communities". In: Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Covilha, Portugal. LNCS 3808, Springer-Verlag, Berlin, 2005, pp. 218-231

[Uschold 1996] Uschold, M., Gruninger, M.: "Ontologies: Principles, Methods and Applications". Knowledge Engineering Review 11 (2), 1996, pp. 93-155

[Wand 1995] Wand, Y., Monarchi, D.E., Parsons, J., Woo, C.C.: "Theoretical foundations for conceptual modelling in information systems development", Decision Support Systems, Volume 15, Issue 4, December 1995, pp. 285-304

[W3C ODA 2006] W3C, (Editores) Tetlon, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., Kendall, E.: "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering". W3C Editors' Draft of 11 February 2006, http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[W3C OWL 2004] W3C, (Editores) Smith, M.K., Welty, C., McGuinness, D.L.: "OWL Web Ontology Language: Guide". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/

[W3C OWL 2004c] W3C, (Editores) Dean, M., Schreiber, G. (Editores): "OWL Web Ontology Language Reference". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-ref/

[W3C OWL 2 2008] W3C, (Editores) Motik, B., Patel-Schneider, P.F., Parsia, B.: "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax". W3C Working Draft, 02 December 2008. http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/

[W3C OWL 2 2008c] W3C, (Editores) Parsia, B., Patel-Schneider, P.F.: "OWL 2 Web Ontology Language: Primer". W3C Working Draft, 11 April 2008. http://www.w3.org/TR/owl2-primer/

[W3C OWL-S 2004] W3C, (Editor) Martin. D: "OWL-S: Semantic Markup for Web Services". W3C Member Submission, 22 November 2004, http://www.w3.org/Submission/OWL-S/

[W3C SWBP 2006] W3C, (Editores) Noy, N.F., Rector, A.:"Defining N-ary Relations on the Semantic Web". W3C Working Group Note, 12 April 2006, http://www.w3.org/TR/swbp-n-aryRelations/

[W3C SWBP 2006b] Knublauch, H., Oberle, D., Tetlow, P., Wallace, E.: "A Semantic Web Primer for Object-Oriented Software Developers". W3C Working Group Note, 9 March 2006, http://www.w3.org/TR/sw-oosd-primer/

[Xia 2007] Xia, F., Jihua, S.: "A Study in Knowledge Ontology Building in the Area of Knowledge Engineering". Third International Conference on Semantics, Knowledge and Grid, 29-31 Oct. 2007, pp. 586-587

Capítulo V

ANÁLISIS DE PROPIEDADES DE UNA ESPECIFICACIÓN ONTOLÓGICA

1. INTRODUCCIÓN

En el capítulo III, se ha descrito cómo la complejidad inherente a los sistemas CSCW precisa grandes dosis de especificación y desarrollo debido a cuestiones como la distribución y coordinación de sus usuarios, o los requisitos de comunicación entre los miembros de un grupo [Carstensen 2003]. El modelado de las entidades de un entorno de trabajo colaborativo y los procesos que tienen lugar en él es también una tarea colaborativa que involucra participantes (*stakeholders*, esto es, arquitectos de sistemas, analistas, usuarios, probadores, programadores, etc.) de las diferentes organizaciones implicadas en las tareas que tienen en común [Jonkers 2004].

Durante el proceso de modelado, los *stakeholders* suelen hacer uso de diferentes tipos de modelos para capturar las reglas de negocio que gobiernan las relaciones entre los distintos activos de una organización, tal y como hemos visto, por ejemplo, para el caso de AMENITIES, donde se proponen distintas vistas de un sistema colaborativo haciendo uso de una extensión al lenguaje UML (cfr. sección 3.4 en capítulo II). Más concretamente, es frecuente utilizar y compartir *modelos estructurales* para describir las entidades principales de un dominio y las relaciones entre ellas (p.e. un diagrama de clases o de objetos), y *modelos de comportamiento* que describen procesos de negocio colaborativos (p.e. un diagrama de clases o una máquina de estados). El conjunto de todos ellos configura la arquitectura del sistema de un sistema colaborativo que permite analizar y comprender cómo funciona.

Aunque los modelos resultantes pueden discutirse con los *stakeholders*, en tanto que son bastante intuitivos, presentan el inconveniente de que la información acerca de las entidades de un sistema y las decisiones de diseño sobre el mismo se dispersan a través de los distintos modelos [Lange 2006]. La Figura V.1 ilustra esta situación.

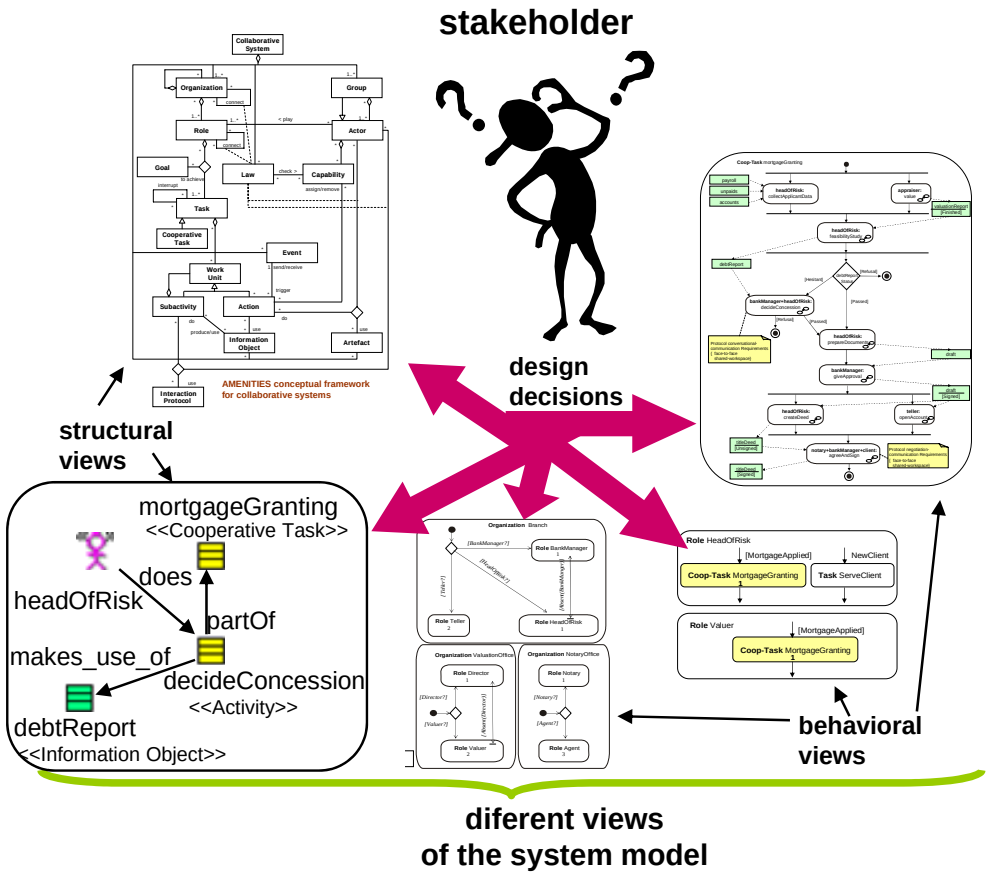


Figura V.1 Dispersión decisiones de diseño entre varias vistas del sistema

Asimismo, un factor clave para la eficacia de la etapa de modelado es que, además de los propios modelos, se comparta también su semántica con objeto de que las partes que colaboran se entiendan mutuamente [Jaekel 2005].

El mapeo propuesto en el capítulo IV para modelos basados en UML de AMENITIES a ontologías, minimiza algunos de los inconvenientes anteriores. Así, la posibilidad de describir ontológicamente tanto de la estructura de las entidades de un sistema colaborativo (clases, instancias y relaciones), como de sus procesos de negocio colaborativos, proporciona una base subyacente sobre la que los distintos *stakeholders* pueden construir modelos de sistema cohesivos y formales, al tiempo que utilizan notaciones más intuitivas tales como UML [Berardi 2005][Kogut 2002].

Asimismo, la semántica bien definida que poseen las ontologías posibilita llevar a cabo un análisis omnicomprensivo de las principales entidades de un entorno colaborativo, así como, compartir su estructura de tal forma que pueda ser utilizada e interpretada por otros programas (p.e. agentes software, sistemas de gestión de bases de datos, etc.) reduciendo ambigüedades [Noy 2001].

En el caso de ontologías implementadas en OWL, como las que se han propuesto en el capítulo anterior, podemos beneficiarnos de los mecanismos de inferencia que proporcionan los razonadores basados en Description Logics. En las siguientes secciones se presenta cómo, mediante el uso de ontologías, es posible analizar propiedades de la especificación de un sistema colaborativo. Los ejemplos se han desarrollado para el caso de estudio del sistema colaborativo correspondiente al proceso de negocio de concesión de hipotecas. Se trata de mostrar el potencial y la aplicabilidad de la propuesta que se presenta en este trabajo de investigación.

2. RAZONAMIENTO SOBRE LOS MODELOS OBTENIDOS

Los mecanismos de razonamiento basados en Description Logics están concebidos fundamentalmente para comprobar propiedades como la consistencia de una jerarquía de clases o conceptos (es posible definir una instancia para todas las clases –o conceptos– de la ontología), subsunción y equivalencia de clases, consecuencias implícitas, etc. [Berardi 2005]. Esta sección presenta un conjunto de casos y situaciones de ejemplo donde pueden aplicarse mecanismos de validación y razonamiento para analizar propiedades de la especificación de un modelo, tales como, su completitud o corrección, etc., o también, inferir relaciones entre conceptos como, por ejemplo, si una actividad precede a otra en una secuencia.

En primer lugar, trataremos de analizar la consistencia de la ontología de dominio en OWL donde se recoge el marco conceptual de AMENITIES. Posteriormente, analizaremos ontologías de aplicación basadas en la ontología de dominio anterior. Para ello, tomaremos como caso de estudio de referencia la especificación del sistema colaborativo de concesión de hipotecas presentado en capítulos anteriores. Asimismo, consideraremos que la ontología que lo describe, presenta una arquitectura que sigue el esquema de diseño de ontologías presentado en el capítulo anterior (cfr. sección 3.2).

2.1 Comprobación de la consistencia de la ontología de dominio para AMENITIES

El proceso de especificación ontológica del marco conceptual de AMENITIES ha servido para revisar su diseño e introducir mejoras. En el capítulo IV, apuntábamos la decisión de habilitar el modelado de un grupo de actores como si se tratara de un sólo actor en el sistema. Esto permite que la asignación/revocación de capacidades pueda hacerse a nivel de grupo y extenderse a todos sus miembros, en lugar de hacerlo uno a uno, con el consecuente ahorro de tiempo y facilidad de gestión.

A priori, parece que bastaría simplemente con definir la clase Grupo como subclase de la clase Actor. De esta forma, la representación del marco conceptual en un diagrama de clases quedaría de la siguiente forma:

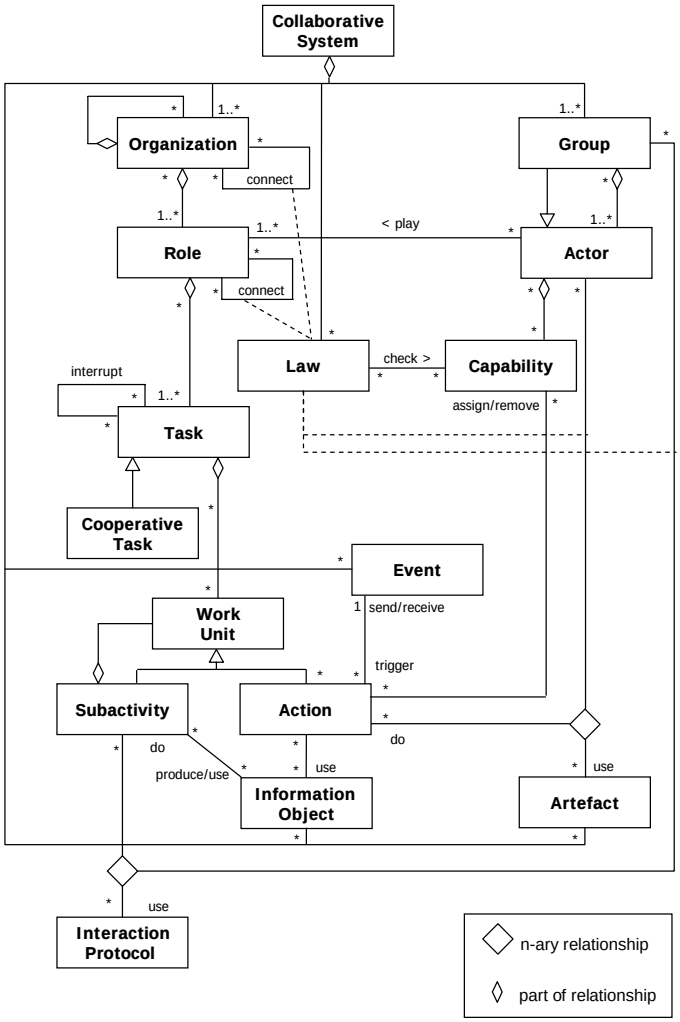


Figura V.2 Marco conceptual de AMENITES con Grupo como subclase de Actor

En cambio, si tras realizar esta modificación lanzamos un razonador sobre la ontología de dominio resultante, obtenemos que la clase *Grupo* es clasificada como subclase de *Capacidad* (ver Figura V.3).

Normalmente, esto se debe a especificaciones incompletas. En principio, la solución que se presenta más fácil o intuitiva para el diseñador de una ontología es la de declarar ambas clases disjuntas. Sin embargo, al lanzar de nuevo el clasificador sobre la ontología, obtenemos que la clase *Grupo* se ha vuelto inconsistente, es decir, no pueden existir instancias de dicha clase (ver Figura V.4). Esto tiene su justificación, ya que, es como si estuviéramos forzando a que una clase y su subclase fueran disjuntas, algo a todas luces inconsistente: si A es subclase de B, A está incluido en B.

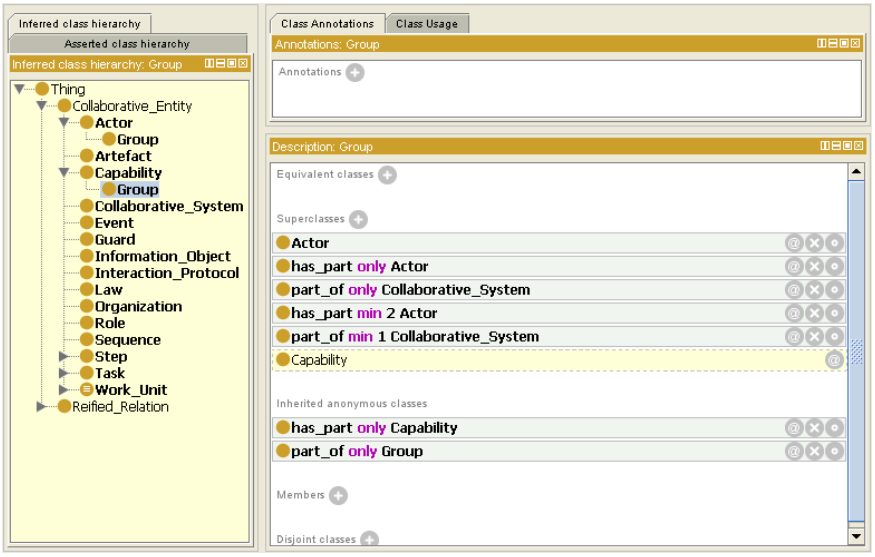


Figura V.3 Clase Grupo clasificada como subclase de Capacidad

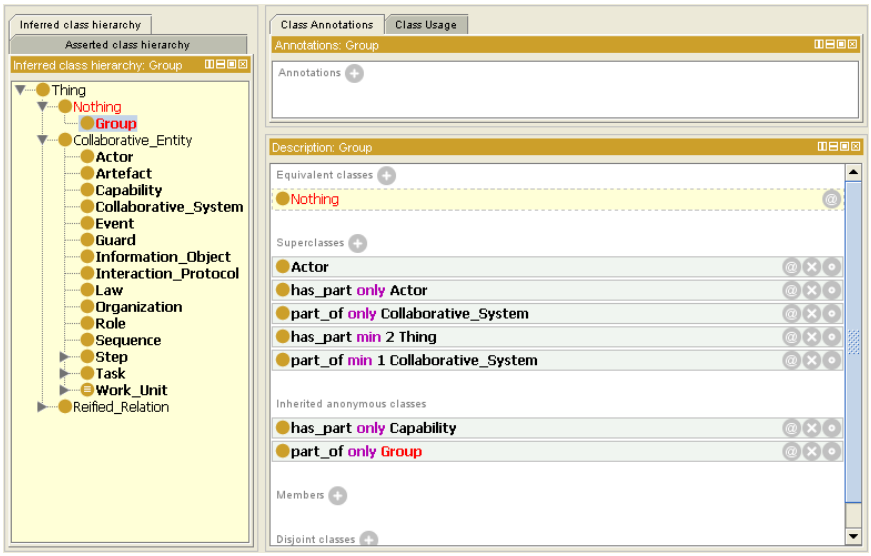


Figura V.4 Clase Grupo inconsistente

Con la clasificación inicial que realizó el razonador, en cierto modo podemos pensar que este considera que *Actor* y *Capacidad* "se parecen", ya que, ambas tienen como subclase a *Grupo*. Ahora bien, si observamos la definición de la clase *Actor* (ver Figura V.5), encontramos que, procedente de la especificación inicial de la ontología, contiene una restricción según la cual, para la relación *se_compone_de* (*has_part*) sólo admite elementos de la clase *Capacidad*. Por otro lado, los Grupos, que tras el cambio introducido también son *Actores*, sólo admiten *Actores* en el rango de la relación *se_compone_de*, lo que entra en conflicto con su superclase. De ahí también que la única forma de conciliar ambas restricciones sea considerar los Grupos como un tipo de Capacidad.

Eliminando dicha restricción o ampliando su rango para que también incluya elementos de la clase *Grupo* el razonador deja de clasificar *Grupo* como subclase de *Capacidad*. No obstante en este caso, lo más adecuado parece ser relacionar los Actores con sus Capacidades por medio de otra relación de tal forma que se puedan mantener las restricciones anteriores.

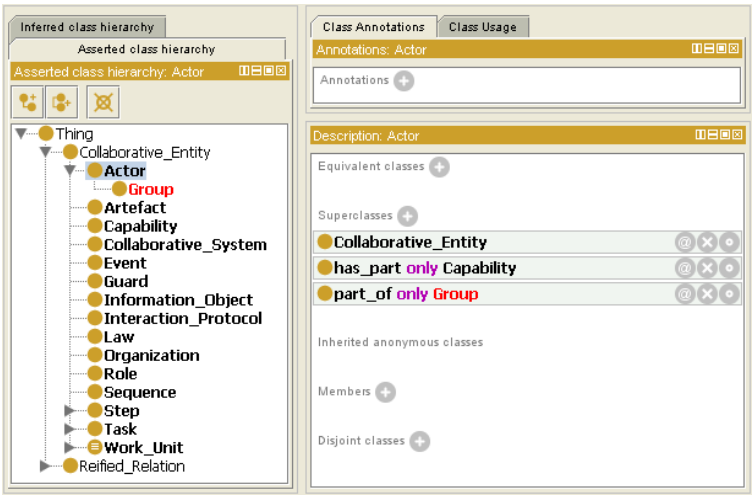


Figura V.5 Clase Actor

En esta sección hemos visto cómo los razonadores basados en Description Logics pueden ayudar a detectar inconsistencias en una jerarquía de clases de una ontología de dominio, aparecidas como consecuencia de un cambio menor en su estructura. En la siguiente sección, ilustraremos cómo se puede aprovechar la potencia de los razonadores en ontologías de aplicación y también cómo la introducción de nuevas clases y propiedades a este nivel nos puede ayudar a adaptar e inferir conocimiento para necesidades específicas.

2.2 Análisis de la completitud en cuanto a estructura entre diferentes modelos o vistas del sistema a través de ontologías de aplicación

En este primer ejemplo vamos a considerar que un analista del sistema ha definido varias entidades en el sistema instanciando algunas clases y relaciones de la ontología de dominio (marco conceptual) de AMENITIES (p.e. *roles*, *tareas*, etc.), y posteriormente ha comenzado a describir el comportamiento del sistema en un diagrama de actividades. Para ello, ha utilizado respectivamente, el diagrama de objetos de la Figura V.6 y el diagrama de actividades de la Figura V.7. Ambas figuras recogen modelos que hacen referencia al mismo escenario en torno a la actividad *decidirConcesión*, pero desde perspectivas diferentes: el diagrama de objetos ofrece una vista estructural, mientras que el diagrama de actividades proporciona una vista de comportamiento. Asimismo, en la parte derecha de las dos figuras

se presenta la especificación en OWL de los diagramas representados utilizando la sintaxis funcional para este lenguaje.

La Figura V.6 describe un escenario en el que se ha declarado la tarea *concesiónDeHipoteca* como *Tarea_Cooperativa*. La actividad *decidirConcesión* es *parte_de* la tarea *concesiónDeHipoteca* y hace uso del objeto de información *informeDeDeudas*. El rol *jefeDeRiesgos se_encarga* (*does* en el modelo) de la actividad *decidirConcesión*.

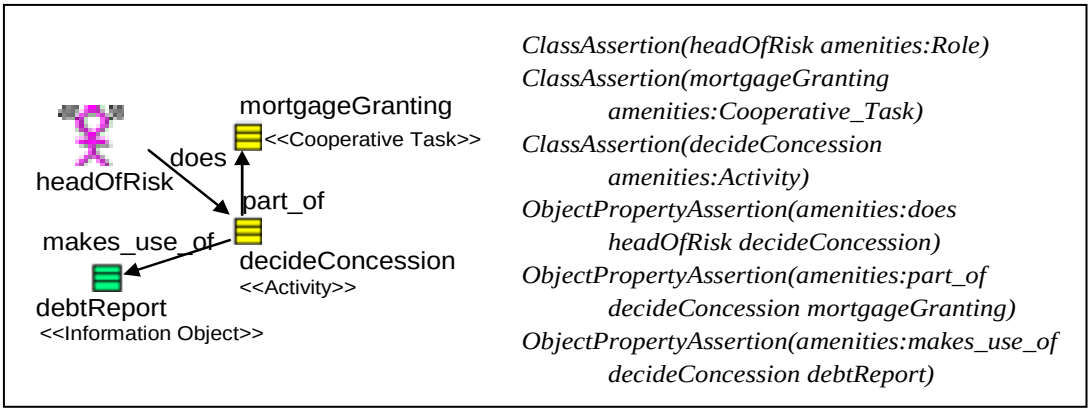


Figura V.6 Especificación de algunas entidades del sistema de concesión de hipotecas (modelo estructural)

La Figura V.7 muestra parte del diagrama de actividades para la tarea cooperativa *concesiónDeHipoteca* y donde la actividad *decidirConcesión* es asignada a los roles *director* (que no se ha declarado en el modelo estructural del diagrama de objetos de la Figura V.6) y *jefeDeRiesgos* (los actores que *desempeñen* dichos roles serán quienes la lleven a cabo), y *hace_uso_del* objeto de información *escritura* en el estado *sin _firmar*.

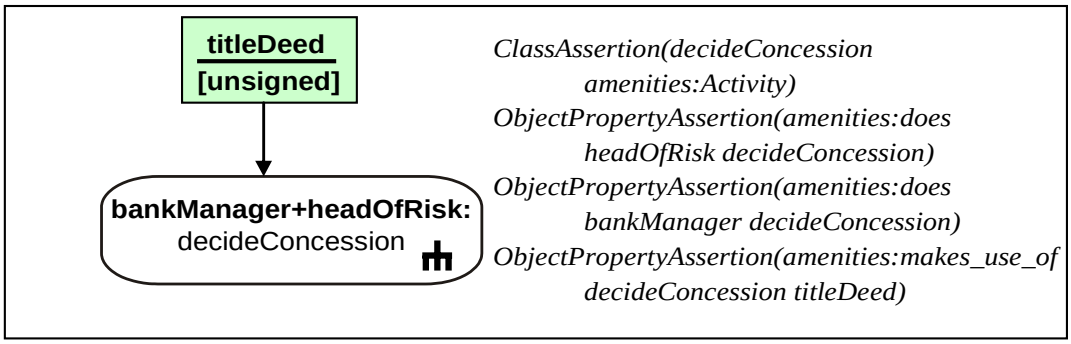


Figura V.7 Especificación de la actividad *decidirConcesión* (modelo de comportamiento)

Sin embargo, imaginemos que el analista mencionado ha cometido un error definiendo los modelos y la información de ambos modelos no coincide exactamente (y por tanto, tampoco lo hacen las correspondientes declaraciones de la ontología; cfr. parte derecha de

las figuras V.2 y V.3). En particular, según el diagrama de objetos de la Figura V.6, sólo el rol *jefeDeRiesgos* es quien *se_encarga* de la actividad *decidirConcesión*, mientras que en la Figura V.7 se declara que son el *jefeDeRiesgos* **junto con** el *director* (símbolo "+" que separa ambos roles dentro del nodo de la actividad) quienes *se encargan* de ella. Además, recordemos que la instancia (objeto) *director* aún no tiene tipo asignado, esto es, todavía no se ha declarado como instancia de la clase *Rol*.

A continuación mostramos las conclusiones que, a partir de la especificación en OWL del escenario anterior, podría inferir un razonador automático basado en Description Logics, así como, ejemplos de acciones que podrían derivarse por medio de herramientas que hagan uso del conocimiento inferido.

Incompleción (I). Desde una perspectiva estructural, el diagrama de objetos (parte derecha de la Figura V.6) especifica que la tarea *decidirConcesión* utiliza (relación *hace_uso_de*) el objeto de información *informeDeDeudas*, pero debido a un error humano durante el diseño, en el diagrama de actividades correspondiente (perspectiva de comportamiento) se ha declarado que la tarea mencionada hace uso del objeto de información *escritura*. Con la ayuda de un razonador, puede tratar de inferirse si ambos objetos de información son el mismo, y de esta forma, comprobar si las siguientes declaraciones extraídas de la especificación ontológica de los dos diagramas son semánticamente equivalentes:

decidirConcesión **amenities:hace_uso_de** informeDeDeudas ≠

decidirConcesión **amenities:hace_uso_de** escritura

En este caso, un razonador no podrá pronunciarse en un sentido un otro, por lo que una herramienta de apoyo a la construcción de la ontología del sistema podría plantear al desarrollador optar entre varias decisiones de diseño con objeto de completar la especificación del sistema y de tal forma que la información representada en los diagramas coincida.

Por ejemplo, si *escritura* es otra forma de denominar el *informeDeDeudas*, las dos instancias pueden declararse idénticas por medio de la sentencia *sameAs* de OWL. En otro caso, la herramienta podría preguntar por otras acciones a realizar sobre la ontología. Por ejemplo, eliminar la sentencia inicial del diagrama de objetos y añadir otra sentencia indicando que es una *escritura* de lo que hace uso la tarea *decidirConcesión* (porque se encontrara correcta la especificación del diagrama de actividades), o simplemente añadir una sentencia para indicar que *decidirConcesión*, aparte del *informeDeDeudas*, hace uso también del objeto de información *escritura*. La Figura V.8 describe esta situación.

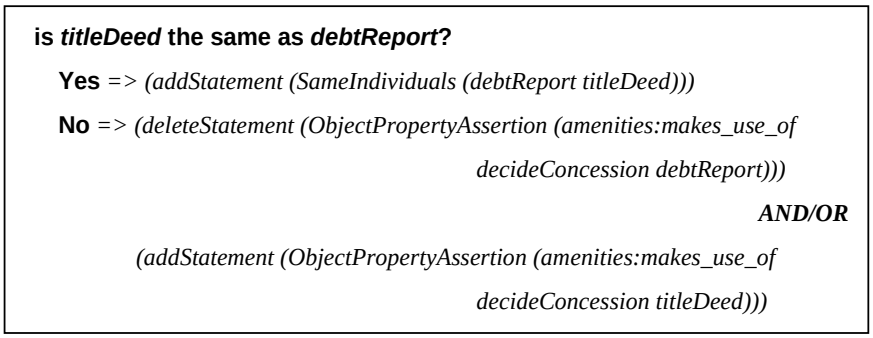


Figura V.8 Acciones sugeridas tras detectar falta de compleción entre dos especificaciones

Es importante destacar que el razonamiento en OWL adopta la *Hipótesis de Mundo Abierto* (*Open World Assumption*, OWA) [W3C OWL 2004], relacionado con las lógicas monotónicas, y bajo el que la imposibilidad de demostrar un hecho no implica que sea verdadero su contrario. En otras palabras, la base de conocimiento definida por una ontología en OWL es presumiblemente incompleta. En el ejemplo anterior, si los objetos de información *informeDeDeudas* y *escritura* no están definidos como iguales, tampoco se puede dar por supuesto que sean diferentes. Simplemente, no se dispone del conocimiento necesario para determinarlo.

El esquema de razonamiento descrito parte más bien del supuesto contrario, es decir, el de mundo cerrado (*Close World Assumption*, CWA), y trata de aprovechar su semántica para guiar al desarrollador. En efecto, la introducción en el diagrama de actividades de un axioma con información distinta a la que ya había en el diagrama de objetos es el punto de partida para pensar que se está produciendo algún error en la especificación. Bajo la hipótesis CWA, al consultar la ontología sobre si la tarea *decidirConcesión* hace uso del objeto de información *escritura*, obtendríamos una respuesta negativa y desencadenaríamos el proceso de consulta al desarrollador descrito más arriba (hasta ese momento la ontología sólo contiene la información correspondiente al diagrama de objetos de la Figura V.6).

El razonamiento basado en la OWA también tiene sus ventajas, ya que, permite al *stakeholder* dejar abiertas ciertas decisiones de diseño sobre un modelo conceptual y de las que no está seguro, al tiempo que puede hacer efectivo y razonar sobre lo que realmente conoce [W3C OWL 2 2008c].

Mediante procesos de soporte al desarrollador como el presentado, pretendemos que sean los diferentes *stakeholders* quienes tomen el control sobre cuáles de las conclusiones inferidas son finalmente incorporadas a la ontología del sistema, es decir, cómo completar la información de la base de conocimiento a partir de la información proporcionada por los razonadores automáticos.

La propia comunidad de desarrolladores de ontologías en el lenguaje OWL se ha hecho eco de la necesidad de adoptar en ciertas situaciones un enfoque mixto que permita en ciertas ocasiones cambiar del supuesto OWA a uno CWA, y viceversa [Grimm 2005], ya que ambos presentan sus ventajas. El paso a procesos de razonamiento partiendo de la CWA conlleva el uso del llamado *operador epistémico* [de Brujin 2005][Donini 1998]. Sin embargo, el soporte al operador epistémico tanto de herramientas como de razonadores, y también desde el propio lenguaje OWL, es limitado por tratarse de un campo de trabajo incipiente [Katz 2005].

Una alternativa para "cerrar" el mundo consiste en especificar restricciones y consultas por medio del *Lenguaje de Axiomas de Protégé* (*Protégé Axiom Language*, PAL [Grosso 2002]). Sin embargo, este lenguaje presenta varios inconvenientes que llevan a descartarlo como solución. Por un lado, se trata de otro lenguaje con una sintaxis y semántica diferentes a la de OWL, con lo que las restricciones que se especifican con él quedan fuera de la especificación de la ontología y se hace necesario otro intérprete para el nuevo lenguaje. Por otro lado, su potencia expresiva y soporte son bastante limitados y no son suficientes para representar implementar la funcionalidad pretendida.

Incompleción (II): razonamiento en cascada. El diagrama de actividades de la tarea *concesiónDeHipoteca* que se muestra (parcialmente) en la Figura V.7, especifica que un elemento del dominio, denominado *director*, y el actor desempeñando el rol *jefeDeRiesgos*, se encargan conjuntamente (relación *does* de la ontología) de la actividad *decidirConcesion*. Sin embargo, hasta ahora hemos supuesto que no hay ninguna afirmación previa sobre qué es (es decir, su tipo o clase) un *director*.

Por otro lado, la formalización de marco conceptual de AMENITIES en OWL1, especifica el dominio y el rango de la relación does de la manera siguiente:

ObjectProperty(amenities:does)

ObjectProperty**Domain**(amenities:does amenities:Role)

ObjectProperty**Range**(amenities:does amenities:Work_Unit)

[1] Y que se corresponde con la ontología de dominio vista en el capítulo anterior. Lamentamos la coincidencia del término "dominio" para hacer referencia también a la clase a la que pertenecen el primer elemento de relación binaria.

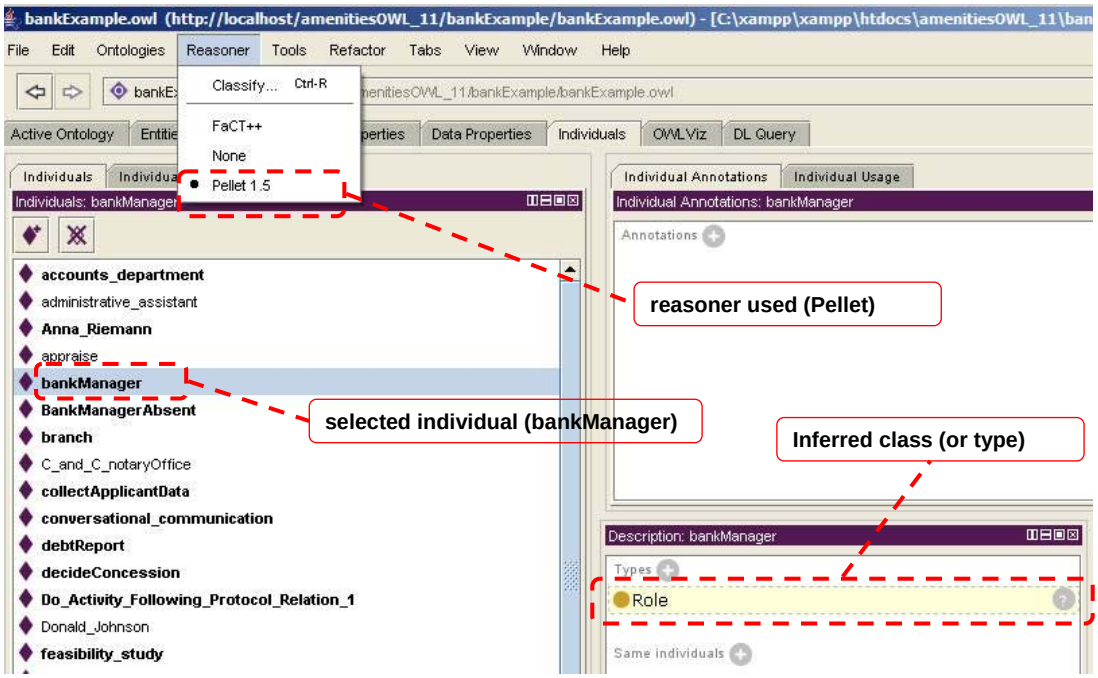


Figura V.9 Inferencia de la clase de una instancia a partir del dominio de una relación

De esta forma, un razonador para OWL podría inferir que la instancia individual *director* es uno de los roles a desempeñar en el sistema, puesto que la relación *does* sólo admite individuos de la clase *Rol* como primer elemento de la asociación. La Figura V.9 muestra una instantánea del editor de ontologías Protégé donde puede verse destacada en beige la clase inferida *Rol* para el elemento *director*. El razonador utilizado para esta captura es Pellet [Sirin 2007].

A partir de este momento, se puede proceder con un proceso en cascada de compleción de la ontología. Por ejemplo, según el marco definido por AMENITIES, cada rol debe *formar parte de*[2], al menos, una organización, lo que se expresa mediante la restricción (ver Tabla IV.I):

$$\text{Role} \sqsubseteq (\geqslant 1\ \text{part\_of.Organization})$$

Sobre la base del nuevo conocimiento inferido por el razonador, una herramienta de soporte al desarrollo podría preguntar, bien por la conveniencia de vincular el rol *descubierto* a alguna de las organizaciones implicadas (en este caso serían la *sucursal*, la *notaría* y la *agenciaDeTasación*), o bien, ofrecer la posibilidad de describir una nueva. Asimismo, AMENITIES define otras restricciones, según las que cada *Rol* debe ser desempeñado por, al menos, un *Actor*, que a su vez, formará parte de un grupo, etc. Por tanto, el proceso

[2] Propiedad *part_of* en la implementación del capítulo IV.

podría continuar igualmente completando la descripción del rol *director* en este sentido, y se terminaría, bien porque el desarrollador decida dejar transitoriamente incompleta la especificación para retomarla más adelante, o bien porque no queden restricciones por satisfacer.

Comprobación de la corrección de una especificación. En el ejemplo que venimos desarrollando, la actividad *decidirConcesión* ha sido asignada (mediante la relación *does*) al *jefeDeRiesgos* en el diagrama de objetos de la Figura V.6.

Consideremos ahora que se decide la tarea cooperativa *concesiónDeHipoteca*, va a ser una de las tareas constituyentes (relación *part_of*) del rol *jefeDeRiesgos*. A su vez, puesto que la actividad *decidirConcesión* forma parte de *concesiónDeHipoteca*, se sigue que también quedará comprendida dentro del rol *jefeDeRiesgos* y así podría especificarlo un analista.

La naturaleza transitiva de la relación de composición (*part_of* en la ontología), permite comprobar la corrección de esta declaración (es decir, que *decidirConcesión* está contenida dentro de las actividades del rol *jefeDeRiesgos*) mediante un proceso de inferencia de acuerdo a la estructura de instancias y clases definida. En particular, la que definen el diagrama de objetos de ejemplo de la Figura V.6 y la ontología de dominio subyacente. En realidad, la correcta inferencia de los hechos mencionados requiere de una estructura de relaciones algo más compleja, ya que:

1. El marco conceptual de AMENITIES impone ciertas restricciones de cardinalidad a algunas clases para la relación de composición *part_of* (cfr. Figuras II.4 y IV.9 en capítulos II y IV, respectivamente, de este documento), como por ejemplo, la que hemos visto en el epígrafe anterior acerca de que un *Rol* debe pertenecer, al menos, a una *Organización*.
2. Como se comentó en el capítulo anterior (cfr. sección 2.1.1), OWL no permite definir restricciones de cardinalidad sobre relaciones transitivas, como podría caracterizarse la relación *part_of*, ya que, torna indecidible el razonamiento [W3C OWL 2004c].

Para sortear esta dificultad se define como transitiva la relación *contenido_en* (*is_within* en la implementación), superpropiedad de *part_of*. De esta forma, las restricciones de cardinalidad se definen en la subpropiedad (*part_of*), mientras que la transitividad se define para la superpropiedad (*is_within*), y apoyar sobre esta última el razonamiento sobre las relaciones de composición/agregación entre individuos. Este mismo enfoque puede encontrarse en otros trabajos [Drummond 2006]. La Figura V.10 esquematiza el proceso de razonamiento seguido. La conclusión inferida se ajusta a lo que podría especificar un analista.

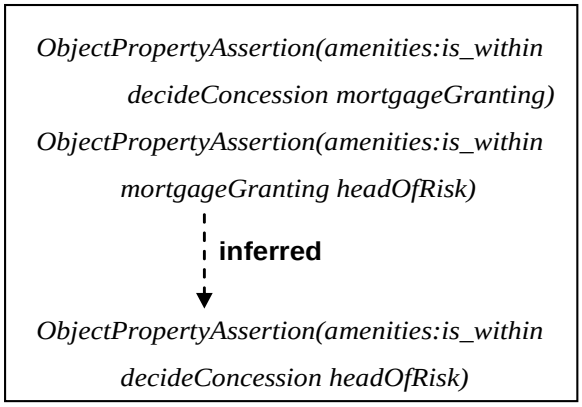


Figura V.10 Ejemplo de inferencia sobre una relación de composición

2.3 Análisis de la especificación del comportamiento (relaciones entre actividades en un flujo de trabajo)

A la hora de modelar procesos de negocio colaborativos, aún pueden obtenerse beneficios adicionales de los mecanismos de razonamiento.

Por ejemplo, en el capítulo anterior (ver sección 2.3.2) presentamos la relación *seguido_por* (*followed_by*) entre dos *steps* para articular secuencias de actividades y acciones dentro de una tarea. Por medio de esta relación se puede trazar cómo transcurre el flujo de control entre actividades dentro de una tarea y también, puesto que se trata de una relación transitiva, inferir relaciones de orden entre unidades de trabajo: si el *step_x* es *seguido_por* el *step_y*, y el *step_y* es *seguido_por* el step_z, un razonador podrá inferir que el *step_x* también es *seguido_por* el *step_z*.

Sin embargo, en este caso volvemos a toparnos con las restricciones para mantener la decidiblidad de los procesos de razonamiento. En efecto, existen múltiples restricciones de cardinalidad sobre la relación *followed_by* que varían según los diferentes tipos de clases *Step* (ver Tabla IV.1 y Figura IV.11 en capítulo IV), y en consecuencia, no puede declararse como transitiva [W3C OWL 2004c]. A ello se añade el hecho explicado también anteriormente de que la relación *followed_by* no conecta dos pasos directamente, sino que lo hace a través de la clase reificada *Followed_by_Relation*.

En este caso, para poder razonar aprovechando el carácter transitivo de esta relación recurrimos al constructor/operador de composición de relaciones que proporciona OWL 2. Así, definimos una nueva propiedad transitiva, que denominaremos *precedes*, y que se define como la composición de la relación *followed_by* con la propiedad *following_step*, y que nos permitirá establecer relaciones e inferir conocimiento sobre el orden en que tienen lugar las actividades en un flujo de trabajo [Drummond 2006]. En Description Logics, la relación *precedes* se definiría de la siguiente forma:

$$followed\_by \circ following\_step \sqsubseteq precedes$$

Conviene tener presente que la declaración anterior puede formularse gracias a la potencia expresiva de los *axiomas complejos de inclusión de roles*[3] (*complex role inclusion axioms*) proporcionados por la lógica descriptiva $\mathcal{SROIQ}$ y soportada recientemente por la nueva versión de OWL (conocida como OWL 2) y los razonadores basados en este lenguaje[4].

En concreto, la propiedad *precedes* permite especificar o inferir conocimiento sobre si el *step* en el que una *actividad* se *lleva a cabo* tendrá lugar algún tiempo después de la realización de otra *actividad* en algún *step* previo. De este modo, se habilita la expresión de nociones como que una actividad tendrá lugar *eventualmente* después de otra, es decir, independientemente de los pasos intermedios por los que atravesaría la secuencia de steps de que se trate.

En un contexto de modelado de procesos de negocio, esto puede resultar especialmente útil para verificar que se siguen ciertos protocolos de actuación con un cierto nivel de generalidad. Por ejemplo, mediante la especificación de restricciones adicionales se puede reforzar la semántica del modelo de tareas para asegurar que ciertas tareas tienen lugar antes que otras, detectar ciclos (p.e. una o varias actividades que se preceden a sí mismas), evitar realizar dos veces una misma actividad u obligar a que sea así, etc. Ilustraremos esto con un ejemplo un poco más adelante.

La Figura V.11 muestra un ejemplo de los hechos o instancias de la relación *precedes* que puede inferir un razonador para la instancia individual *first_step_1* y que corresponde al primer paso de la tarea *concesiónDeHipoteca* que implementamos en el capítulo IV (cfr. Figura IV.12). Estos hechos corresponden a inferencias realizadas por el razonador a partir de la descripción de dicha tarea en OWL, y no a hechos explícitamente declarados. Como es lógico, en este caso el resultado del razonamiento muestra que el primer paso, instancia *first_step_1*, antecede (*precedes*) al resto de pasos de la secuencia. Los hechos inferidos aparecen destacados con color beige de fondo.

[3] Rol: propiedad o relación.

[4] http://pellet.owldl.com/features/#standard

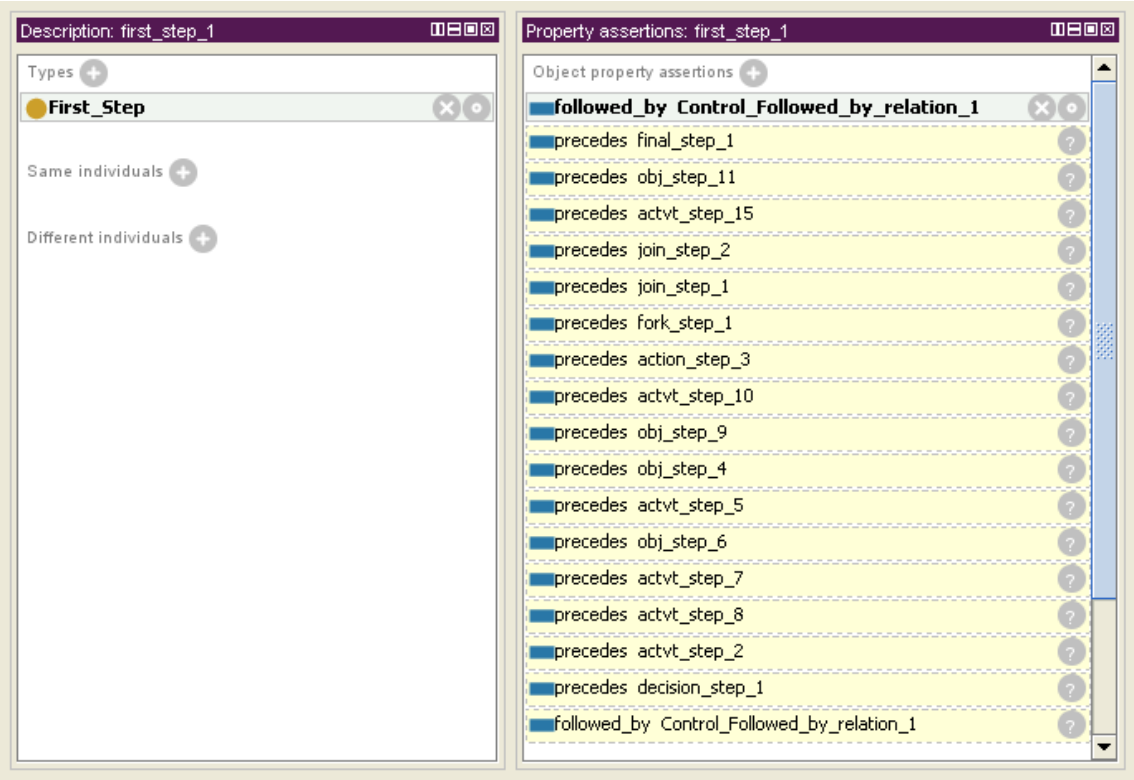


Figura V.11 Ejemplo de deducciones para la relación *precedes*

Explotación del razonamiento sobre el orden entre actividades. En esta subsección pretendemos mostrar cómo podemos valernos de la información obtenida sobre el orden en que se ejecuta una actividad para caracterizar un flujo de trabajo.

En particular, vamos a tratar de incrementar la confiabilidad de los procesos del sistema colaborativo de gestión hipotecas. Para ello, en primer lugar estableceremos una clasificación de las unidades de trabajo que distinga entre aquellas que entrañen algún *riesgo* y aquellas otras que más bien realicen una función de *supervisión*, lo que se traduce en dos nuevas subclases de *Unidad_de_Trabajo*, *Riesgo* y *Supervisión*. Esta clasificación será ortogonal a la ya existente que diferencia entre *Actividades* y *Acciones*, es decir, podrán existir *Actividades* y *Acciones* de riesgo, y *Actividades* y *Acciones* de supervisión, ya que, aprovechando una característica del lenguaje OWL, las nueva clases se definen mediante condiciones necesarias y suficientes[5].

Una vez hecho esto, será posible identificar aquellos pasos en los que tiene lugar una Actividad o Acción de riesgo, y aquellos en los que tiene lugar una Actividad o Acción de supervisión. A su vez, con esto podremos verificar una propiedad deseable en una secuencia o flujo de tareas, como es que toda actividad (o acción) que conlleve un cierto

[5] En OWL estas clases se conocen como "clases definidas"

riesgo, preceda a alguna actividad de supervisión. Aunque algo general, de esta forma se estaría contribuyendo a mejorar la confiabilidad de un proceso de negocio colaborativo, ya que, cada actividad de riesgo estaría supervisada por la existencia alguna otra actividad eventual posterior donde sepamos que se van a revisar decisiones que se hayan tomado anteriormente.

Para comprobar la característica que acabamos de mencionar podemos hacer uso de la relación *precedes*. Por ejemplo, en la tarea *concesiónDeHipoteca* (cfr. Figuras II.7 y IV.12), la actividad *decidirConcesión* se va a considerar como de riesgo (de la clase *Riesgo*). Asimismo, la actividad *darAprobación* se clasifica como una actividad de supervisión (clase *Supervisión*). Por simplicidad, también consideraremos que los *steps* asociados a *decidirConcesión*, *prepararDocumentos*, *borrador* y *darAprobación* (ver Figura IV.12) son, respectivamente, *step_w*, *step_x*, *step_y*, y *step_z*, y las instancias correspondientes de la relación *Followed_by_Relation* que enlazan estos pasos: *fwd_by_x*, *fwd_by_y* y *fwd_by_z*. Por tanto, en esta ocasión, la declaración a comprobar es ver si se cumple que:

$$step\_w \quad precedes \quad step\_z$$

Conforme a la semántica de la relación *precedes* tenemos:

$$step\_w \quad followed\_by \quad fwd\_by\_x$$

$$fwd\_by\_x \quad following\_step \quad step\_x$$

$$(step\_w \quad followed\_by \quad fwd\_by\_x) \circ (fwd\_by\_x \quad following\_step \quad step\_x) \rightarrow$$

$$step\_w \quad precedes \quad step\_x$$

Repitiendo el mismo proceso para cada par de pasos adyacentes en el flujo de control de *concesiónDeHipoteca* tenemos:

$$step\_w \quad precedes \quad step\_x \quad precedes \quad step\_y \quad precedes \quad step\_z \rightarrow step\_w \quad precedes \quad step\_z$$

La Figura V.12 ilustra el proceso de razonamiento.

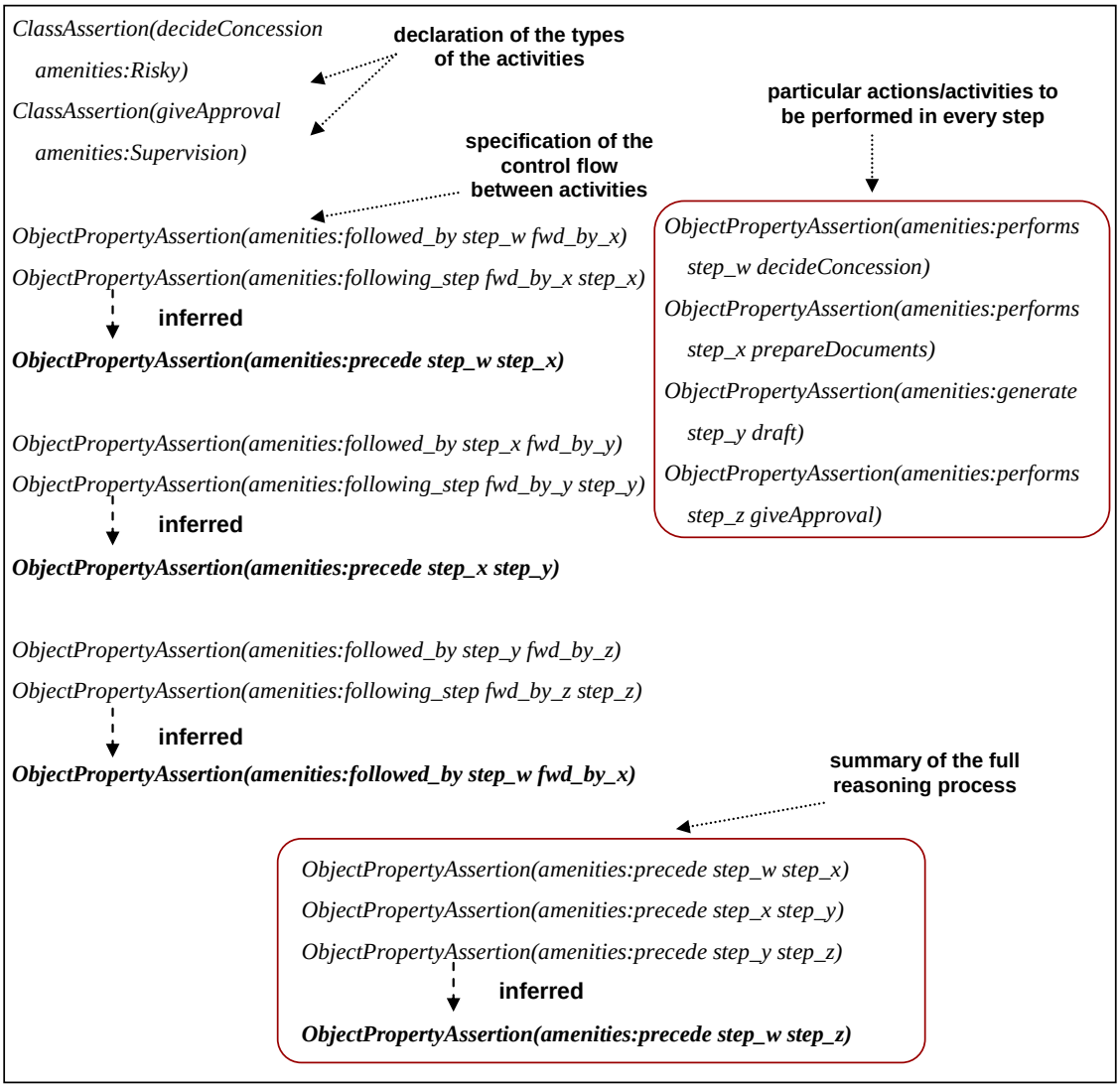


Figura V.12 Razonamiento sobre el orden entre actividades

El esquema propuesto no pretende ser una alternativa a otras técnicas y formalismos de análisis de flujos de trabajo, como las basadas en Redes de Petri [Kristensen 2004] y sus derivados [Aalst 2005]. Estos formalismos poseen una semántica de ejecución para sus modelos que permite simularlos y analizar sus propiedades (p.e. vivacidad, estados/actividades inalcanzables, ausencia de interbloqueos, etc.). Por esta razón, en otros trabajos se han definido mapeos que traducen la representación ontológica de los diagramas de tareas de AMENITIES al formalismo *Redes de Petri Coloreadas* [Noguera 2009].

Lejos de esto, lo que se presenta es una primera aproximación, al análisis de flujos de trabajo desde una perspectiva ontológica, complementaria a las técnicas citadas, por medio de procesos de razonamiento basados en Description Logics. Dichos procesos están concebidos para analizar fundamentalmente cómo es la estructura de un conjunto de

conceptos en un modelo conceptual. Por otro lado, puesto que OWL no proporciona de forma nativa constructores para el modelado de flujos de trabajo, los modelos construidos con este lenguaje (ontologías) tampoco poseen una semántica de ejecución que permitiese simularlos. Además, las pasos por los que discurre finalmente una tarea, variará con cada ejecución concreta de la misma y dependerá de que se vayan cumpliendo las condiciones expresadas en las guardas que regulan la transición de una actividad a otra. Por estas razones, a la hora de analizar una secuencia de actividades es difícil ir más allá de un proceso de caracterización de los procesos basado en las relaciones entre conceptos.

Por ejemplo, el esquema de razonamiento propuesto no puede considerar separadamente las distintas alternativas que introduce una bifurcación en un flujo de control, lo que permitiría analizar si para ciertas ejecuciones de un proceso no se cumple que toda actividad de riesgo precede a una actividad de supervisión. Para la completa verificación de esta regla habría que seguir un algoritmo que para cada paso en el que tiene lugar una actividad de riesgo, recorriera todos sus posibles subsiguientes y para cada uno comprobara que se alcanza un paso en el que se realiza una actividad de supervisión. Sin embargo, el soporte para el desarrollo de algoritmos de este tipo, que puedan acceder y consultar la ontología de esta forma, es todavía limitado.

Por último, la Hipótesis de Mundo Abierto (OWA) vuelve a jugar en contra del razonamiento que se puede llevar a cabo. En efecto, cabría plantearse la posibilidad de *identificar como inseguras aquellas secuencias que contienen algún paso donde se lleva a cabo una actividad de riesgo y que* ***no precede*** *a un paso donde se lleva a cabo una actividad de supervisión*. La sentencia en cursiva de la frase anterior puede formularse sin grandes problemas en OWL. Sin embargo, bajo el supuesto OWA, un razonador siempre concluirá que no dispone de certeza suficiente para clasificar así una secuencia: aunque en la información de que disponga no encuentre que el paso en cuestión preceda a otro de supervisión, tampoco encuentra información que confirme lo contrario. La forma de evitar este problema sería "cerrar" el mundo, es decir, aplicar el operador epistémico.

3. CONCLUSIONES DEL CAPÍTULO

La especificación de un sistema suele recogerse en diferentes vistas o modelos, posiblemente desconectados entre sí, que contienen distintas decisiones de diseño sobre el mismo. El lenguaje UML, por ejemplo, ofrece 13 diagramas diferentes para modelar tanto la estructura como el comportamiento de un sistema. La diversidad de modelos y representaciones es fuente de inconsistencias entre ellos y dificulta a los *stakeholders* pensar sobre el sistema desde un punto vista global.

Por otro lado, capturar la información referente a cada modelo o vista del sistema en una representación formal subyacente permite llevar a cabo procesos de razonamiento automáticos que ayudan a detectar posibles inconsistencias en la especificación del sistema en su totalidad. Esto contribuye también a proporcionar un modelo del sistema cohesivo y coherente.

A partir del conocimiento o consecuencias lógicas inferidas por un razonador, las herramientas de edición de modelos pueden, en tiempo de diseño, sugerir cambios que corrijan algunas inconsistencias detectadas, verifiquen la validez de algunas declaraciones o ayuden a completarlas, eliminar redundancias, etc. Todo ello simplifica la labor de los distintos analistas implicados [Oberle 2005].

En esto difiere un proceso de especificación de un sistema basado en ontologías de uno basado en UML, donde no se presta especial cuidado a que la semántica de los modelos pueda ser utilizada para propósitos de inferencia automática [OMG 2007c].

Por último, la modularización de la ontología subyacente distinguiendo, según el grado de especificidad de las entidades descritas, entre ontologías de dominio y ontologías de aplicación, ayuda a razonar y detectar inconsistencias a diferentes niveles. Esto facilita la acotación y gestión de las inconsistencias detectadas, así como la extensión estructurada y ordenada de la especificación de un sistema para adaptarla a necesidades específicas.

En nuestro caso concreto, en primer lugar se han abordado algunas inconsistencias detectadas sobre la ontología de dominio que recoge el marco conceptual de AMENITIES. Posteriormente, basándonos en el caso de estudio del sistema de concesión de hipotecas se han tratado distintos aspectos como la detección de incompleciones en un entorno concreto. Sobre el mismo caso de estudio, se ha mostrado un ejemplo de cómo razonar sobre el orden de las actividades y también definir relaciones y clases más específicas a las existentes para razonar y caracterizar secuencias de flujos de trabajo.

REFERENCIAS DEL CAPÍTULO

[Aalst 2005] van der Aalst, W.M.P., ter Hofstede, A.H.M.: "YAWL: Yet Another Workflow Language". Information Systems 30, 2005, pp. 245-275

[Berardi 2005] Berardi, D., Calvanese, D., De Giacomo, G.: "Reasoning on UML class diagrams". Artificial Intelligence 168 (1–2), pp. 70–118 (2005)

[de Brujin 2005] de Bruijn, J., Lara, R., Polleres, A., Fensel, D.: "OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web". Proceedings of the 14th International Conference on World Wide Web (WWW '05). ACM, New York, NY, pp. 623-632

[Carstensen 2003] Carstensen, P.H., Schmidt, K.: "Computer supported cooperative work: new challenges to systems design". Handbook of Human Factors/ Ergonomics, Asakura Publishing, Tokyo, 2003, pp. 619-636

[Donini 1998] Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: "An epistemic operator for description logics". Artificial Intelligence, 100(1-2), April 1998, pp. 225-274

[Drummond 2006] Drummond, N., Rector, A.L., Stevens, R., Moulton, G., Horridge, M., Wang, H, Seidenberg, J.: "Putting OWL in Order: Patterns for Sequences in OWL", Proceedings of the 2nd OWL Experiences and Directions Workshop (OWL-ED 2006), Athens, Georgia, USA, 2006

[Grimm 2005] Grimm, S., Motik, B.: "Closed-World Reasoning in the Semantic Web through Epistemic Operators". Proceedings of the OWL Experiences and Directions Workshop, Galway, Ireland, 2005

[Grosso 2002] Grosso, W.: "The Protégé Axiom Language and Toolset (PAL)", 2002 (Last updated: September, 2005): http://protege.stanford.edu/plugins/paltabs/paldocumentation/index.html

[Jaekel 2005] Jaekel, F.W., Perry, N., Campos, C., Mertins, K., Chalmeta, R.: "Interoperability Supported by Enterprise Modelling", LNCS 3762, Springer-Verlag, Berlin, 2005, pp. 552-561

[Jarrar 2007] Jarrar, M.: "Mapping ORM into the SHOIN/OWL Description Logic - Towards a Methodological and Expressive Graphical Notation for Ontology Engineering". LNCS, Vol. 4805, 2007, pp. 729-741

[Jonkers 2004] Jonkers, H., Lankhorst, M.M., van Buuren, R., Hoppenbrouwers, S., Bonsangue, M.M., van der Torre, L.W.N.: "Concepts for Modeling Enterprise Architectures". Int. J. Cooperative Inf. Syst. 13, 2004, pp. 257-287

[Katz 2005] Katz, Y., Parsia, B.: "Towards a Nonmonotonic Extension to OWL". Proceedings of the OWL Experiences and Directions Workshop, Galway, Ireland, 2005

[Knublauch 2004] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.: "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", LNCS 3298, 2004, pp. 229-243

[Kogut 2002] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J.: "UML for ontology development". Knowl. Eng. Rev. 17, 1 (Mar. 2002), pp. 61-64

[Kristensen 2004] Kristensen, L.M., Jorgensen, J.B., Jensen, K.: "Application of Coloured Petri Nets in System Development". ACPN 2003. LNCS 3098, Springer-Verlag (2004), pp. 626-685

[Lange 2006] Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: "UML Software Architecture and Design Description". IEEE Software, 23 (2006), pp. 40-46

[Noguera 2009] Noguera, M., Hurtado, M.V., Rodríguez, M.L., Chung, L., Garrido, J.L.: "Ontology-driven Analysis of UML-Based Collaborative Processes using OWL-DL and CPN". Science of Computer Programming, (in press, under review), 2009

[Noy 2001] Noy, N.F., McGuinness, D.L.: "Ontology development 101: A guide to creating your first ontology". Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001

[Oberle 2005] Oberle, D., Staab, S., Studer, R., Volz, R.: "Supporting application development in the semantic web". ACM Trans. Interet Technol. 5, 2 (May. 2005), pp. 328-358

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09. Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Sirin 2007] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: "Pellet: A practical OWL-DL reasoner", Journal of Web Semantics, 5(2), 2007, pp. 51-53

[W3C OWL 2004] W3C, (Editores) Smith, M.K., Welty, C., McGuinness, D.L.: "OWL Web Ontology Language: Guide". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/

[W3C OWL 2004c] W3C, (Editores) Dean, M., Schreiber, G. (Editores): "OWL Web Ontology Language Reference". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-ref/

[W3C OWL 2 2008c] W3C, (Editores) Parsia, B., Patel-Schneider, P.F.: "OWL 2 Web Ontology Language: Primer". W3C Working Draft, 11 April 2008. http://www.w3.org/TR/owl2-primer/

[W3C SWRL 2004] W3C, (Editores) Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: "SWRL: A Semantic Web Rule Language Combining OWL and RuleML". W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/SWRL/

Capítulo VI

APLICACIONES DE LA PROPUESTA

1. INTRODUCCIÓN

La ontología de dominio de AMENITIES para la creación de ontologías de sistemas colaborativos, así como, el esquema de desarrollo de ontologías general propuesto –que sigue un enfoque modular, diferenciando entre ontologías de dominio y varios niveles de ontologías de aplicación–, se han puesto en práctica en distintos ámbitos entre los que se encuentran: el diseño de un sistema de observación de la interacción para aplicaciones groupware y razonadores basados en casos.

Asimismo, se presenta el desarrollo de una herramienta gráfica de edición de ontologías que guía el desarrollo modular de ontologías propuesto en el capítulo IV.

A continuación se presentan los sistemas concretos en los que se han aplicado.

2. SISTEMA DE OBSERVACIÓN DE LA INTERACCIÓN PARA COLLECE

La ontología de dominio de AMENITIES ha sido tomada como referencia para el diseño de un *sistema de observación de la interacción* (SOI) para la aplicación colaborativa COLLECE.

Los SOI procesan todas las interacciones efectuadas por los usuarios con un sistema y las almacenan en documentos de *log*. Estos documentos permiten analizar el proceso de trabajo llevado a cabo por los usuarios y estudiar cómo estos interactúan con la aplicación colaborativa [Viégas 2004].

Por su parte, COLLECE es un entorno CSCW que soporta la implementación de programas software mediante el trabajo colaborativo distribuido síncrono de parejas de programadores [Bravo 2007]. En estos sistemas, uno de los miembros del grupo de trabajo

desempeña el rol de codificador, o *driver*, y se encarga de implementar el código fuente del programa en construcción, mientras que el otro miembro del grupo, bajo el rol *observador* o *supervisor*, revisa el trabajo que se está desarrollando y aporta sugerencias [Williams 2002].

COLLECE integra un conjunto de herramientas colaborativas para soportar las interacciones propias de este tipo de sistemas (ver Figura VI.1):

- Editor compartido (Figura VI.1-a). Es el espacio de trabajo común donde el codificador inserta o modifica el código fuente del programa que se está construyendo. Un telepuntero (Figura VI.1-b) indica en cada momento la línea del código donde el codificador está trabajando de manera que el supervisor pueda seguir las evoluciones de su trabajo en tiempo real.
- Panel de sesión. Este panel contiene la fotografía y el nombre de cada uno de los programadores (Figura VI.1-c) que colaboran en la construcción del programa.
- Paneles de coordinación:
    - Panel de edición (Figura VI.1-d). Utilizado por el supervisor cuando desea adquirir el rol de codificador y hacer una modificación en el código fuente.
    - Panel de compilación (Figura VI.1-e). Cuando alguno de los programadores estime que el código fuente construido debe ser compilado, utiliza este panel para transmitir la solicitud al compañero y utilizar la consola (Figura VI.1-h) para entrar en un proceso de compilación.
    - Panel de ejecución (Figura VI.1-f). Cuando alguno de los programadores quiere iniciar un proceso de pruebas que compruebe la validez del programa construido, utiliza este panel para transmitir la solicitud al compañero.
- Chat estructurado (Figura VI.1-g). Concebido para que la comunicación entre el codificador y el supervisor sea constante y fluida. Este chat contiene botones con el inicio de los actos conversacionales que habitualmente se utilizan durante la construcción de programas (por ejemplo, "En el bucle…"). Así, cuando el programador selecciona uno de estos actos conversacionales, sólo tiene que completar el final de frase.

En particular, la captura de pantalla que muestra la Figura VI.1 corresponde a un estado del sistema para una configuración concreta en la que existen dos actores, *cbravo* y *rafa*. La figura muestra la aplicación tal y como se vería por el primero de ellos (*cbravo*), quien está desempeñando el rol de *driver*. Esto último se indica mediante la etiqueta *Editing* debajo del

nombre del actor (ver Figura VI.1-c; la edición del programa sólo puede hacerse bajo el rol *driver*).

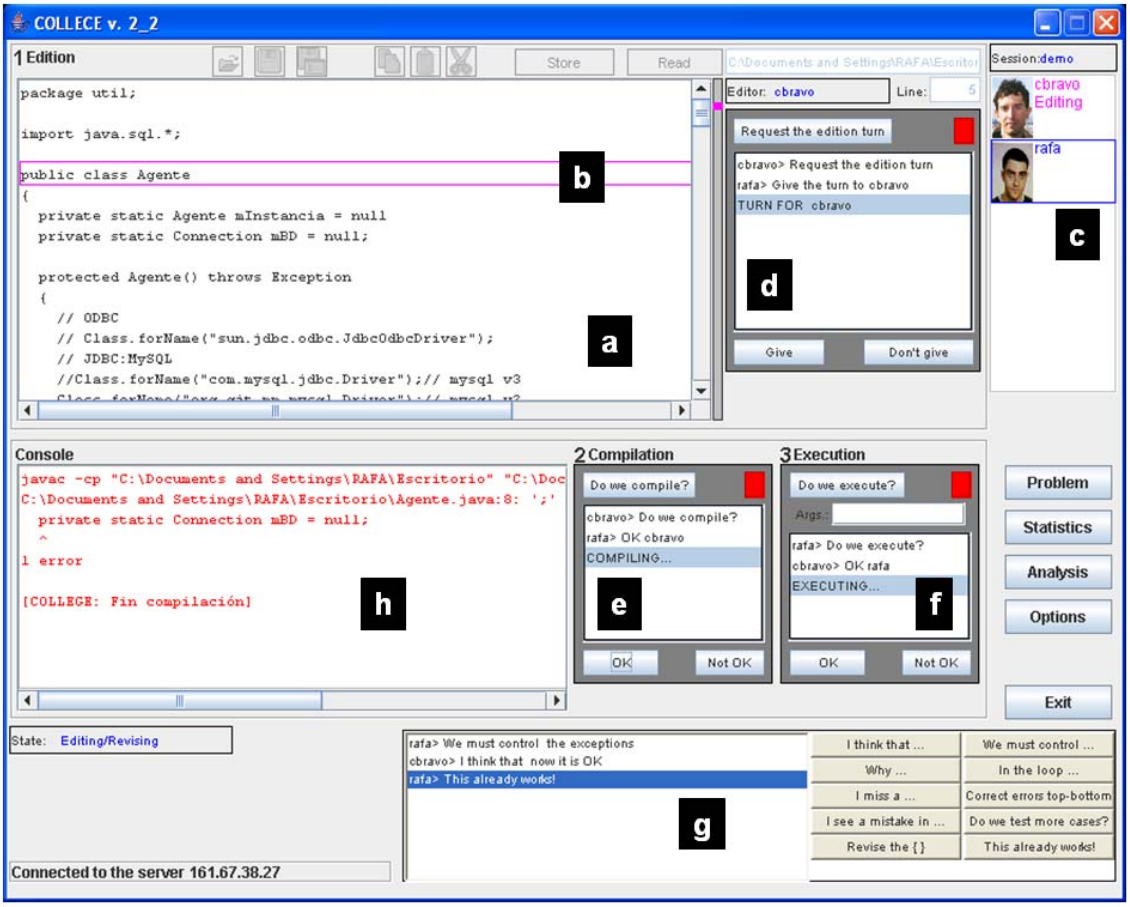


Figura VI.1 Interfaz de usuario de COLLECE

2.1 Aplicación de la propuesta

Conforme al modo de describir la estructura de un sistema colaborativo en AMENITIES (en base a roles, tareas, grupos de actores, etc.), y a partir de la ontología de dominio para esta metodología –es decir, reutilizándola–, se ha definido una ontología de aplicación para describir la forma de operar en COLLECE. En esta nueva ontología de aplicación se especifican los roles que puede desempañar un actor (*driver* y *observador*), las tareas y acciones asociadas (*edición*, *compilación*, *cesión_de_turno*, etc.), los artefactos con los que llevarlas a cabo (en este caso serán los *widgets* de la aplicación, como por ejemplo, el chat estructurado o los paneles de edición), etc.

Esta información es la que posteriormente utiliza el SOI diseñado para la generación de archivos log que registren la actividad del usuario. Por ejemplo, cada vez que se solicita o se cede el turno de edición (acciones), el sistema de observación puede recuperar de la ontología qué roles estaban desempeñando los actores en ese momento e introducir esta

información el archivo log junto a una marca de tiempo. Posteriormente, se podrá analizar cuál de los dos actores tiene más iniciativa a la hora de solicitar el turno, quiénes lo ceden más fácilmente, etc.

En la Figura VI.2 se muestra una representación gráfica parcial del entorno colaborativo COLLECE correspondiente a la configuración del sistema que aparece en la Figura VI.1.

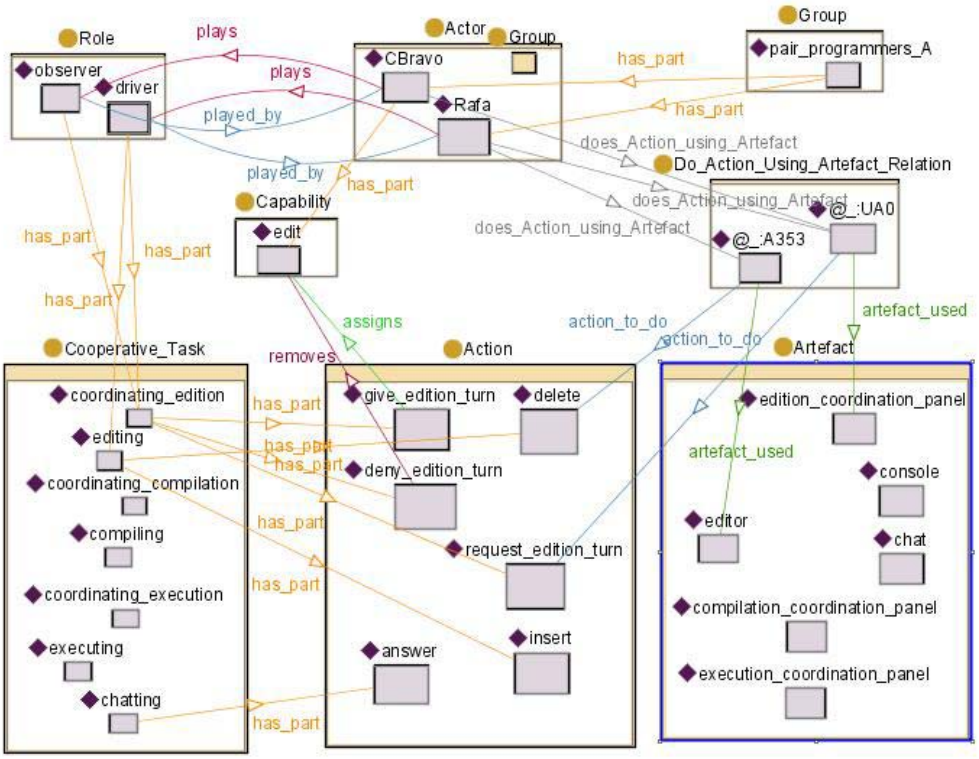


Figura VI.2 Modelo ontológico instanciado para una configuración de COLLECE

3. APLICACIÓN DEL ESQUEMA DE DISEÑO DE ONTOLOGÍAS EN OTROS DOMINIOS

Al margen de su aplicación para el desarrollo de sistemas colaborativos basado en ontologías, el esquema modular de diseño de ontologías distinguiendo entre los niveles de dominio y aplicación se ha exportado a otros ámbitos como el de los sistemas de Razonamiento Basado en Casos (*Case-Based Reasoning*, CBR).

3.1 Representación ontológica de conocimiento para sistemas CBR

En general, los enfoques CBR son estrategias de resolución de problemas que tratan de abordar nuevas cuestiones (*casos*) mediante la adaptación de las soluciones empleadas para resolver situaciones anteriores [Riesbeck 1989]. Cada caso es aplicable en un contexto

determinado denominado *dominio* y, al menos, contiene una *descripción* formal del problema o situación que se pretende resolver (esta descripción muchas veces se recoge en forma de pares *atributo-valor*) y su *solución*. Opcionalmente, también pueden incluir un conjunto de *reglas de adaptación*, una *descripción en lenguaje natural del problema*, *índices*, etc.

Estos elementos constituyen las clases de una ontología de dominio para describir la estructura del conocimiento en un CBR. Las instancias individuales de dichas clases o especializaciones de las mismas se realizarán en ontologías de aplicación subsiguientes [Garrido 2008]. La Figura VI.3 muestra una representación gráfica de la ontología de dominio implementada para describir casos de sistemas CBR.

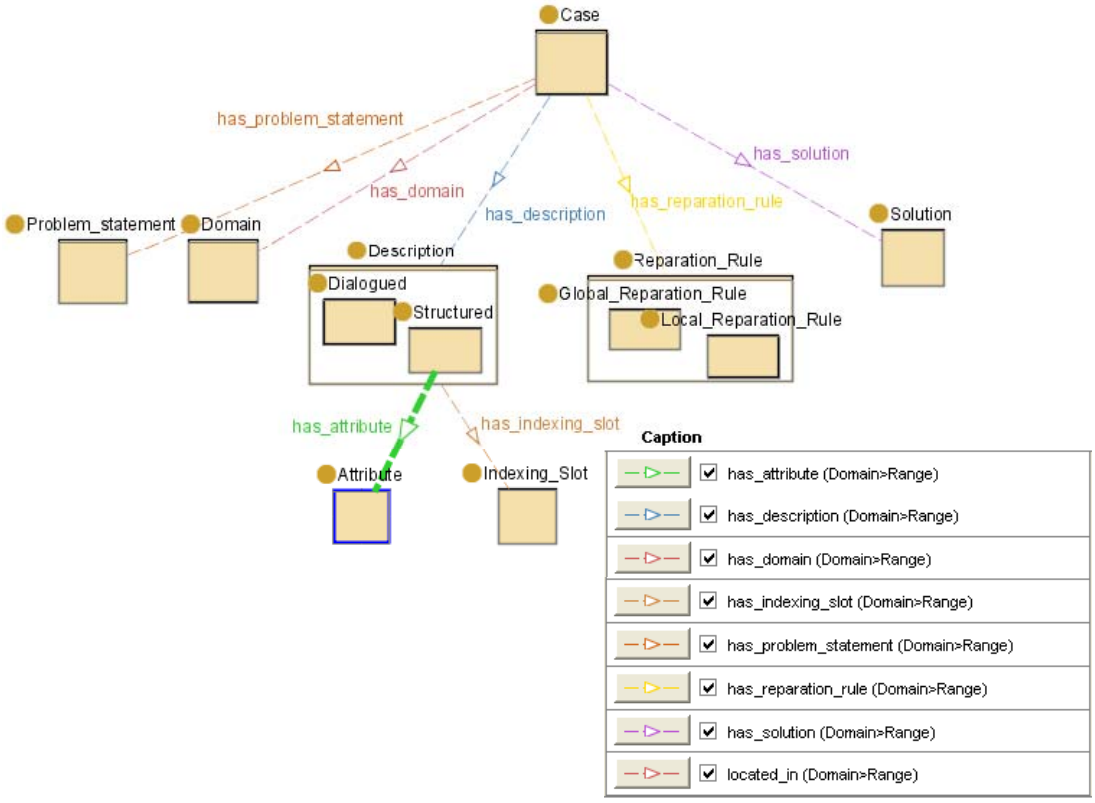


Figura VI.3 Representación gráfica de ontología de dominio para describir casos de sistemas CBR

La descripción ontológica de casos presenta la ventaja de que facilita a las aplicaciones software que los utilizan la identificación de las distintas partes del mismo y su recuperación: descripción, solución, etc. Asimismo, también se facilita el establecimiento de correspondencias con la información contenida en repositorios de casos remotos. Por ejemplo, si un mismo concepto tiene dos nomenclaturas diferentes, pero se declara la equivalencia de los términos empleados para referirnos a él, un agente software encargado de la recuperación de información de casos pasados no tendrá problema en identificar

ambos términos con el mismo concepto y de esta forma podrá aumentar la base de casos con la que comparar.

4. EDICIÓN DE ONTOLOGÍAS

Sin duda, gran parte de la difusión alcanzada por el lenguaje OWL se debe gracias a dos factores fundamentales:

1. La rápida aparición de herramientas edición, como Protégé-OWL [Knublauch 2004] o Swoop [Kalyanpur 2005], que permiten abstraer la prolija sintaxis basada en XML del lenguaje, descargando al desarrollador de tediosas tareas de codificación. No hay que olvidar que los lenguajes para la Web Semántica, como OWL y RDF/Schema, están pensados para formalizar la descripción de un dominio con objeto de que sea procesable por computadoras, antes que para la comunicación entre personas [W3C OWL 2004].
2. La realidad de los razonadores para este lenguaje, como Pellet [Sirin 2007] o FaCT++ [Tsarkov 2006], capaces de inferir conocimiento implícito en la especificación de una ontología o detectar inconsistencias en la misma [Parsia 2005]. A diferencia de las herramientas de edición, el desarrollo de un razonador requiere amplios conocimientos de lógica matemática y complejidad computacional.

No obstante, las herramientas de soporte a la creación de ontologías como las que acabamos de mencionar, aún presentan ciertas deficiencias en lo que se refiere a guiar al desarrollador durante su proceso de construcción. Por ejemplo, en la línea de lo comentado en el capítulo anterior sobre razonamiento, a la hora de especificar las tuplas de una relación sería de esperar que un editor mostrara en primer lugar, las instancias individuales correspondientes al dominio o al rango de una relación. Si aun así, el desarrollador pensara en utilizar instancias de otra clase, la herramienta podría ofrecer la posibilidad de cambiar el dominio o el rango de la relación, o la clase de la instancia. Este tipo de soporte, que requiere de cierta integración entre editores de ontologías y razonadores, no lo ofrecen las herramientas actuales.

Por otro lado, ciertos usuarios pueden preferir la edición gráfica de los modelos ontológicos, al igual que ocurre con las herramientas *CASE*[1] para otros lenguajes de modelado como, por ejemplo, ArgoUML[2] y Rational Rose[3] para UML, que suelen ser más

[1] CASE: Computer-aided Software Engineering

[2] http://argouml.tigris.org/

intuitivas [Berardi 2005][Brockmans 2004]. Además, el empleo de una sintaxis visual en la actividad de modelado conceptual se ha demostrado capaz de reducir el número de errores cometidos por el diseñador y el tiempo de especificación de los modelos [Brockmans 2006]. Este tipo de sintaxis también convierte a los modelos resultantes en un vehículo de comunicación más intuitivo que facilita la comunicación informal entre *stakeholders*.

En este sentido, la mayoría de las herramientas o *widgets* actuales para la edición de ontologías en OWL sólo permiten la representación gráfica de una especificación, y con frecuencia, la que proveen no es muy amigable o fácil de manipular. El ejemplo más destacable es el *Integrated Ontology Development Toolkit* (IODT) [Pan 2006], de IBM, aunque no implementaba todas las características de OWL e interrumpió su desarrollo en diciembre de 2007.

Conscientes de la importancia de contar con herramientas de soporte a la edición de modelos conceptuales, en general, y también para la formalización de ontologías, en particular, hemos comenzado el desarrollo de una herramienta para la edición gráfica de las ontologías en el lenguaje OWL que supla las carencias que acabamos de mencionar. Este tipo de herramientas también resulta fundamental para la implementación efectiva de enfoques de desarrollo dirigidos por modelos, sobre todo, en lo que respecta a la ayuda que pueden prestar al transformar unos modelos en otros [OMG 2003].

En lo que respecta a la visualización, otra característica interesante que pretendemos incorporar es la de poder definir vistas de la ontología que permitan a los desarrolladores centrarse en algún aspecto particular del dominio que se esté modelando, al mismo tiempo que la ontología subyacente sigue manteniendo la descripción del sistema en su totalidad. El soporte de las herramientas actuales a este tipo de funcionalidades es inexistente.

Finalmente, la herramienta está concebida fundamentalmente para habilitar la creación modular, o estructurada, de ontologías en el lenguaje OWL. El objetivo es que pueda tomar ontologías de dominio como referencia para guiar al desarrollador en la creación de las ontologías de aplicación correspondientes según el enfoque presentado en la sección 3.2 del capítulo IV.

4.1 Herramienta visual de edición de ontologías de forma modular

La herramienta que estamos desarrollando permite la especificación gráfica de ontologías de dominio y sus correspondientes ontologías de aplicación, como forma de guiar la edición modular de ontologías.

---

[3] http://www-01.ibm.com/software/rational/

Por ejemplo, si estamos intentando modelar un sistema colaborativo siguiendo el enfoque de AMENITIES, la herramienta podrá procesar la ontología de dominio definida en el capítulo IV para el marco conceptual de la metodología. A la hora de especificar ontologías de aplicación, la herramienta ofrecería como posibles entidades a instanciar las clases y relaciones de dicho marco: actores, roles, qué roles desempeña cada actor, etc. Siguiendo el enfoque presentado en el capítulo IV, también se podrían definir subclases de las ya definidas en la ontología de dominio, como tipos de actividades, tipos de roles, etc.

Por el contrario, si nos encontramos modelando la información de un caso para un CBR, la herramienta podría tomar como referencia la ontología presentada en la sección 3.1, y ofrecer la posibilidad de definir conjuntos de pares atributo-valor para describir un caso.

En ambos casos, la herramienta permitirá previamente la edición gráfica las ontologías de dominio para AMENITIES y para los CBR, así como, cualesquiera otras para diferentes ámbitos.

La definición de las entidades de un modelo (clases, instancias, relaciones) se lleva a cabo mediante una notación gráfica similar a la utilizada por los editores de UML para representar los diagramas de clases (p.e. cajas para representar clases, flechas para representar relaciones, etc.). La Figura VI.4 muestra una captura de la herramienta. La definición de las restricciones se recogen también en cajas de un color especial (en la Figura VI.4 aparecen representadas en cajas verdes y asociadas a las clases a las que hacen referencia).

Los diagramas resultantes se codifican finalmente en ficheros OWL con información de las clases, instancias y relaciones, y en ficheros SVG [W3C SVG 2003] con información acerca de la disposición de los componentes gráficos en cada diagrama.

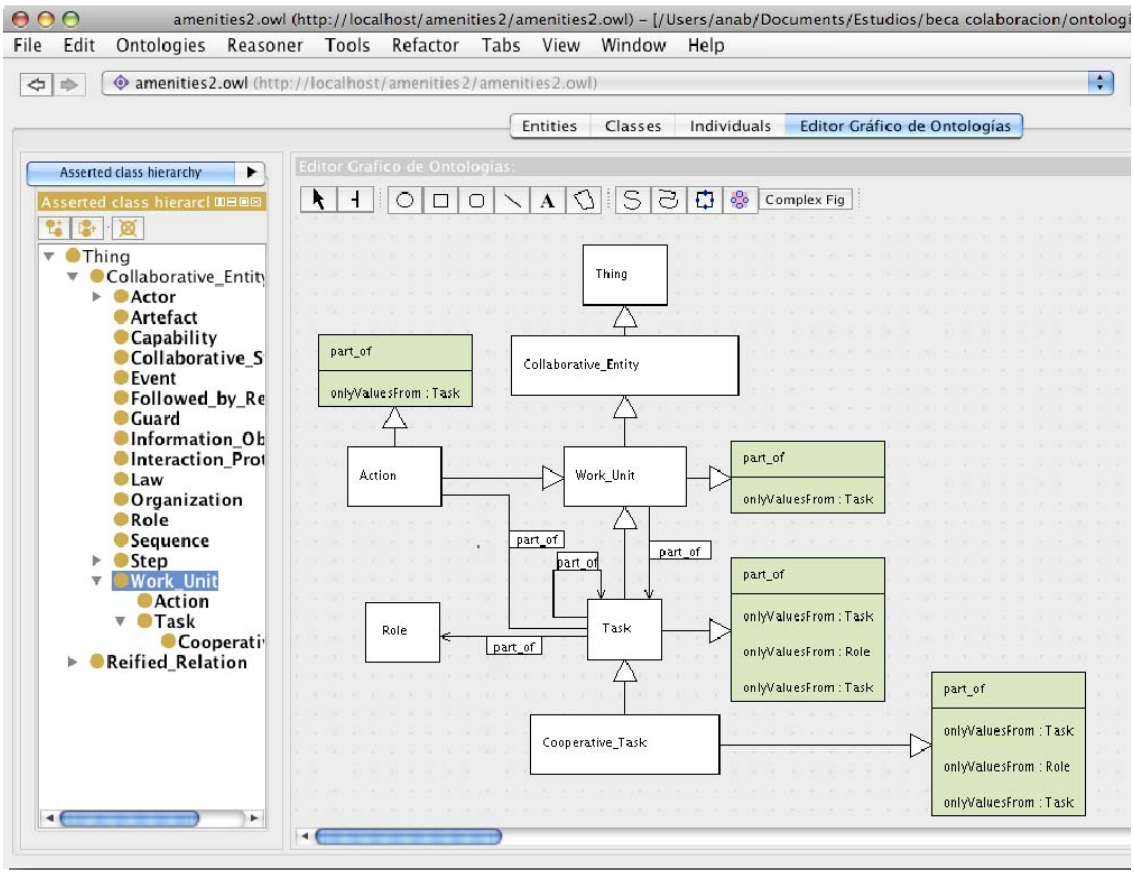


Figura VI.4 Representación gráfica de la herramienta desarrollada (Fuente: Ana B. Pelegrina Ortiz)

4.1.1 Arquitectura

La herramienta se está desarrollando como *"plug-in"* de la popular herramienta de creación de ontologías denominada *Protégé*. La Figura VI.5 proporciona una representación de la arquitectura del editor, la cual se ha tratado de diseñar de la forma más modular posible, de manera que en un futuro se puedan añadir otros módulos o componentes.

Las cajas blancas son componentes externos al editor gráfico desarrollado. Las flechas negras gruesas indican interfaces de comunicación entre componentes o subsistemas. Las flechas negras finas indican relaciones de uso de componentes. Las cajas naranjas son componentes secundarios pensados para futuras extensiones.

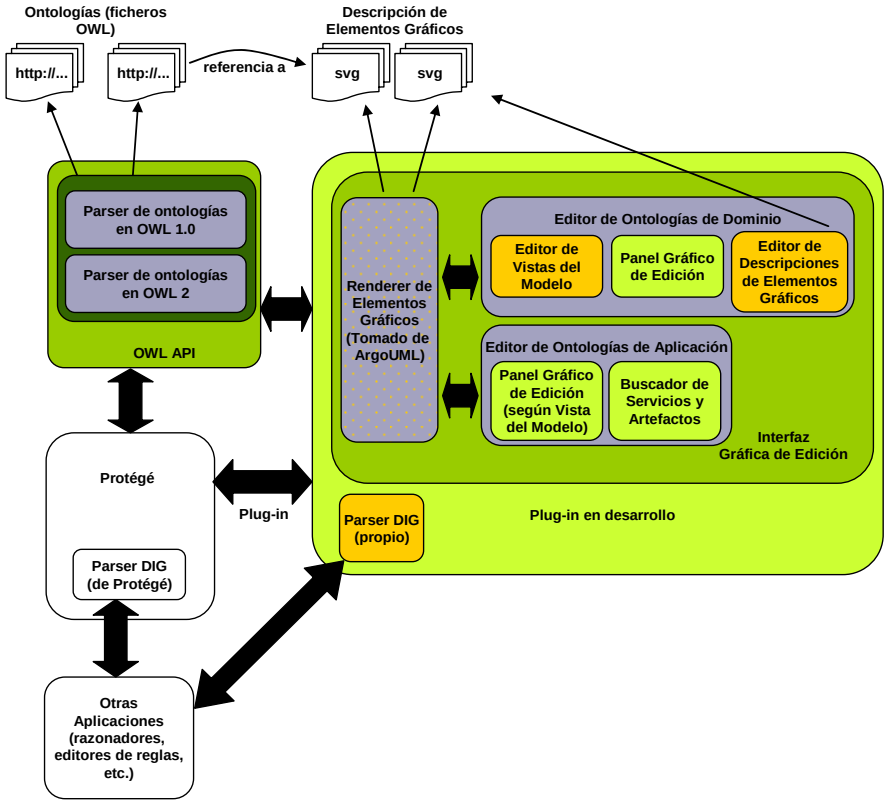


Figura VI.5 Arquitectura del editor gráfico de ontologías

4.1.2 Extensiones

Una de las extensiones consideradas para la herramienta consiste en un editor de vistas de la ontología. Esto resulta útil cuando se quiere organizar la información para razonar mejor sobre un aspecto particular del sistema abstrayendo u ocultando otros. El conjunto de todas estas vistas conforma la arquitectura de la empresa o del sistema que se está modelando. La posibilidad de definir dichas vistas y las relaciones (correspondencias, transformaciones, etc.) entre ellas constituye, a su vez, uno de los pilares sobre los que se asientan los enfoques de Ingeniería Dirigida por Modelos (cfr. capítulo III, sección 6).

La herramienta permitiría definir dichas vistas en el nivel de las ontologías dominio, así como, representar y proporcionar la funcionalidad conveniente a cada vista en el nivel de aplicación. La disposición, conexión, navegabilidad, etc., entre vistas es un tipo de información "ortogonal" a la descripción del sistema que se está modelando y que dependerá de cada dominio concreto (como ejemplo, se pueden tomar las vistas del sistema propuestas en AMENITIES –de organización, interacción, cognitiva, etc.–, para el caso particular de los sistemas colaborativos; cfr. sección 3.4 en el capítulo III).

Por otro lado, los procesos de transformación de vistas pueden beneficiarse de la semántica formal del lenguaje OWL subyacente para comprobar que una transformación entre vistas no produce una vista inconsistente [OMG 2007c].

Otras posibles extensiones se refieren a habilitar la personalización de los componentes gráficos utilizados y la creación de un analizador sintáctico (*parser*) DIG[4] para poder interoperar con otros *plug-ins* o aplicaciones para Protégé.

5. CONCLUSIONES

En este capítulo se han presentado algunos de los ámbitos en los que se están poniendo en práctica las propuestas presentadas en capítulos anteriores.

En concreto, el diseño de un sistema de observación para COLLECE basado en ontologías muestra una aplicación real y directa del enfoque de desarrollo de ontologías para describir sistemas colaborativos presentado en capítulo IV.

Las ontologías propuestas para la descripción de sistemas CBR y los casos que almacenan, representan un ejemplo de la aplicabilidad de la propuesta de desarrollo de ontologías a otros ámbitos distintos al de los sistemas colaborativos.

Por último, una herramienta software de modelado suele ser el resultado y colofón de un desarrollo teórico previo, así como, un aval y medida de su viabilidad y pragmatismo. En este sentido, se ha presentado una herramienta de edición gráfica de ontologías diseñada para guiar al desarrollador en el proceso de creación de ontologías de aplicación enmarcadas dentro de ontologías de dominio más generales. La herramienta también se ha diseñado para poder dar soporte en un futuro a la implementación de enfoques de Ingeniería Dirigida por Modelos mediante la habilitación de diferentes vistas de las ontologías que se estén especificando.

REFERENCIAS DEL CAPÍTULO

[Berardi 2005] Berardi, D., Calvanese, D., De Giacomo, G.: "Reasoning on UML class diagrams". Artificial Intelligence 168 (1–2) (2005), pp. 70-118

[Bravo 2007] Bravo, C., Duque, R., Gallardo, J., García, J., García, P.: "A Groupware System for Distributed Collaborative Programming: Usability Issues and Lessons Learned". International Workshop on Tools Support and Requirements Management for Globally Distributed Software Development, Munich, 2007, pp. 50-56

[Brockmans 2004] Brockmans, S., Volz, R., Eberhart, A., Loffler, P.: "Visual modeling of OWL DL ontologies using UML". Proceedings of 3rd International Semantic Web Conference, Hiroshima, Japan. LNCS 3298, pp. 198-213

[4] DIG: DL Implementation Group. http://dl.kr.org/dig/

[Brockmans 2006] Brockmans, S., Haase, P., Hitzler, P., Studer, R.: "A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies". ESWC 2006. LNCS 4011, pp. 303-316

[Garrido 2008] Garrido, J.L., Hurtado, M.V., Noguera, M., Zurita, J.M.: "Using a CBR Approach based on Ontologies for Recommendation and Reuse of Knowledge Sharing in Decision Making". 8th International Conference on Hybrid Intelligent Systems (HIS 2008). IEEE Press, 2008, pp. 837-842

[Kalyanpur 2005] Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B., Hendler, J.: "SWOOP: a web ontology editing browser". Journal of Web Semantics 4 (2) (2005), pp. 144-153

[Knublauch 2004] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.: "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", LNCS 3298, 2004, pp. 229-243

[OMG 2003] OMG, "MDA Guide Version 1.0.1". OMG Document Number: omg/03-06-01. Available at: http://www.omg.org/docs/omg/03-06-01.pdf

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09. Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Pan 2006] Pan, Y., Xie, G., Ma, L., Yang, Y., Qiu, Z., Lee, J.: "Model-Driven Ontology Engineering". J. Data Semantics VII (2006), pp. 57-78

[Parsia 2005] Parsia, B., Sirin, E., Kalyanpur, A.: "Debugging OWL ontologies". Proceedings of the 14th International Conference on World Wide Web (WWW '05). ACM, New York, pp. 633-640

[Riesbeck 1989] Riesbeck, C.K., Schank, R.C.: "Inside Case-Based Reasoning". Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989

[Sirin 2007] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: "Pellet: A practical OWL-DL reasoner", Journal of Web Semantics, 5(2), 2007, pp. 51-53

[Tsarkov 2006] Tsarkov, D., Horrocks, I.: "FaCT++ Description Logic Reasoner: System Description". Proc. of the International Joint Conference on Automated Reasoning (IJCAR 2006), 2006, LNAI 4130, pp. 292-297

[Viégas 2004] Viégas, F.B., Wattenberg, M., Kushal, D.: "Studying cooperation and conflict between authors with history flow visualizations". Proceedings of the 2004 Conference on Human Factors in computing Systems, (2004), pp. 575-582

[Williams 2002] Williams, L., Kessler, R.: "Pair Programming Illuminated". Addison-Wesley (2002)

[W3C OWL 2004] W3C, (Editores) Smith, M.K., Welty, C., McGuinness, D.L.: "OWL Web Ontology Language: Guide". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/

[W3C SVG 2003] W3C, (Editores) Ferraiolo, J., 藤沢 淳 (Jun, F.), Jackson, D.: "Scalable Vector Graphics (SVG) 1.1 Specification". W3C Recommendation, 14 January 2003, http://www.w3.org/TR/SVG11/

Chapter VII

CONCLUSIONS & ONGOING WORK

1. CONCLUSIONS

Pervasiveness of both intra and cross-organizational collaborative processes with compelling demands for work efficiency in a highly competitive market has drawn the attention of many practitioners from the CSCW field.

As in other Systems and Software Engineering (SSE) research areas, conceptual modelling is recognized as a primary activity for successfully conducting the subsequent development of groupware artefacts for CSCW systems. Thus, conceptual modelling is targeted at providing stakeholders with meaningful system descriptions to reason about the arrangement and distribution of organizational assets. The resulting models also ground the basis so that stakeholders can discuss with each other and share their insight about the collaborative-system model in whole.

In this context, Model-driven Engineering (MDE) has burst upon the scene in the last years. This approach to systems and software development emphasizes the role of conceptual modelling as a means to connect and capture user demands in software products to be subsequently developed. MDE-based approaches begin with the specification of high abstraction models, also known as Computation Independent Models (CIM's), insulated from specific technological platforms.

A key factor in this process is that, besides the models themselves, their inherent semantics be also shared so as to these models are interpreted the same by all collaborating participants.

In this sense, ontology-related technologies are proving to be a suitable means to enable comprehensive analysis and craft sharable descriptions of the main entities involved in a collaborative system thanks to their formal and clearly-defined semantics. A prominent example is the Web Ontology Language (OWL) equipped with decidable reasoning

capabilities based on Description Logics that enable model consistency checks to be accomplished automatically.

This thesis has described a systematic approach to the development of ontologies for collaborative systems in the context of the AMENITIES methodology. The main contributions contained herein can be summarized as follows:

- We have accomplished a thorough review of different formalisms to describe ontologies. As a result of this study, OWL has demonstrated to be the most suitable language for that aim due to its support for automated inference capabilities. OWL has a Web-oriented design, what makes it especially appropriate to describe collaborative settings, since the Web has become the space where most of nowadays collaborative processes take place.

- We have reviewed and formalized the AMENITIES conceptual framework by its specification in a *domain ontology*. This ontology serves as a reference conceptual structure from which to start describing particular collaborative systems through more specific *application ontologies*. The resulting specifications make up cohesive formal descriptions of collaborative models upon which it is possible to carry out reasoning procedures.

  These ontologies constitute formal underlying CIM's (i.e., core modelling artefacts) of an ontology-driven engineering approach for the development of collaborative systems.

  Additionally, we have depicted a modular approach for the construction of system ontologies that has been exported to domains other than that of collaborative systems.

- Several guidelines and design patterns have been provided for representing some constructs used in the AMENITIES UML-based models that do not have their corresponding counterpart in OWL, e.g., n-ary relationships, or prevent the use of logic inference, e.g., cardinality restrictions on transitive properties. We have shown that the use of these patterns still preserves concept structure sound and complete.

- We have defined an extra structure of both classes and relationships intended to enable the description of workflows in the OWL language. This set of supplementary classes and relationships enriches the AMENITIES conceptual framework ontology (a domain ontology) and serves as a skeleton so that the behaviour of a collaborative system can be described. Additionally, it circumvents

the use RDF list constructs that turn OWL descriptions undecidable and thus, out of the scope of Description Logic reasoners.

Furthermore, we have defined a mapping between the entities of the UML metamodel for activity diagrams and the just-mentioned set of entities used to describe workflows in OWL. This mapping allows UML activity diagrams to be represented in OWL descriptions and vice-versa. The translation of one model into the other does not entail any information lost since the mapping has been defined through a bijective correspondence.

- We have shown the benefits of formalizing collaborative-system specifications by means of OWL ontologies, namely:
    - Early detection of inconsistencies and/or meaningless concept structures. In the review process of the AMENITIES conceptual framework, new changes were introduced that lead to inconspicuous entailments. For example, one of these changes implied that the class *Group* ought to be subclass of the class *Capability*.
    - Inference of non-explicitly declared facts, e.g., class membership. This accrues the completeness of system specifications.
    - Further reasoning capabilities. In OWL, it is possible to define new classes on the basis certain properties of the individuals and to characterize properties (e.g. as transitive, functional, etc.). We can use this power to reason on particular, tailored properties of the entities in a domain. We have illustrated this fact by reasoning on the order of the activities in a workflow for granting a mortgage.
- Regularly, the development of an ontology is not an aim in itself. We have devised an interaction observation system that makes use of domain and application ontologies for the COLLECE groupware to analyse user interactions in pair programming activities. In the same vein, we have implemented a domain ontology about the structure of case descriptions to be used by CBR systems in solution searching and retrieval.
- Finally, we are developing a visual ontology editor that guides the designer in the modular construction of both domain and application ontologies according to an ontology-driven engineering process.

2. ONGOING WORK

Ongoing work is focussing on two main, cutting-edge areas of concern for CSCW: Service Oriented Architecture (SOA) paradigms and Requirements Engineering (RE).

2.1 Service ontology

Another main approach to SSE that has burst upon the scene in the last years is SOA, which promotes the development of systems and software architectures through the deployment, discovery and reuse of capabilities which may be supplied by different providers [OASIS 2006].

In this paradigm, once the functionality offered by a service provider is made openly available through accurate interfaces, system and software construction are simplified and significantly differ from what they have been so far. Actually, system and software development become a high-abstraction level activity consisting in the composition of already-developed functionalities spread throughout and provided by a network of collaborative partners, rather than consisting in the design and implementation of new ones. This activity is also called *orchestration*.

Computing systems based on SOA demand the ability to specify, discover, offer and use services and policies in a declarative manner. This requires that metadata about services (functionality, incompatibility, pre and post conditions, etc.) be reliable and accurate enough. That is where Semantic Web languages like OWL come into play: a reasoner on an OWL-based ontology may check whether two services are compatible, who offers a particular service or foresee the activation of other services depending on certain conditions.

In this context, we aim at defining an Ontology of Services (OfS) intended to characterize their semantics as fully as possible. This ontology would enable important improvements in the:

- Discovery of services.
- Detection of inconsistencies in service composition.
- Inference of non-explicitly declared properties in service descriptions and specifications of service compositions applying reasoning procedures based on Description Logics.

The ultimate goal would be to connect the resulting ontology with collaborative system ontologies such as the ones presented in this work. The OfS would help to achieve automated, precise, rapid analysis and development of the systems to be built. Figura VII.1

depicts a sample scenario of the *mortgageGranting* used as a leading example in this thesis. At the upper side of the figure, the ontological description the organizations is graphically represented. The requested service providers and their interrelationships have been depicted at the bottom.

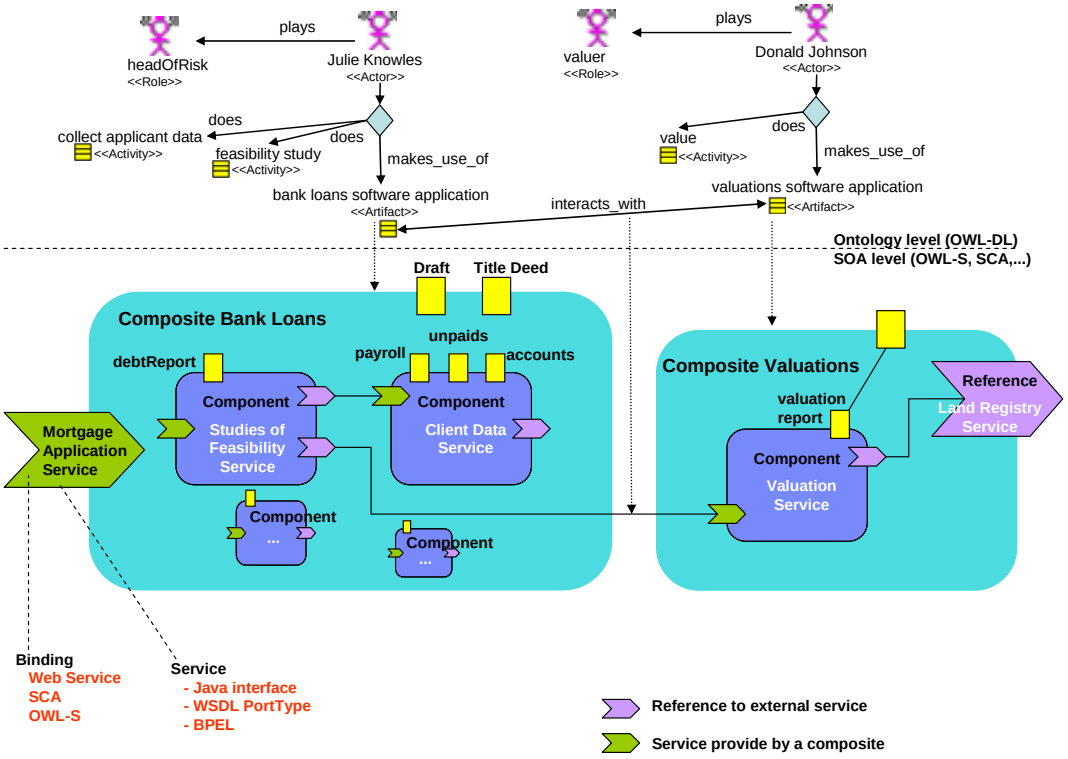


Figura VII.1 Relation between ontology-based models and SOA's

2.2 Ontology of goals

Beside a priori expectations, experience in the development and subsequent use of groupware systems, however, demonstrates that many of these systems fail to provide useful tools for effective collaboration. Consequently (and in addition to technological issues), it becomes necessary to consider the social dimension of the collaborative work by analyzing how groups work and evolve, and most importantly, their goals.

Goals put in motion collaborative work. Organizations plan strategic goals for the company yearly. The achievement of these goals requires they are subsequently divided into smaller goals that are associated with different assets (human and non-human) of the organization for easy of management. Actors have their own goals and necessities –i.e., requirements– and even groups, and these may conflict with organizational ones. Additionally, activities may serve to attain more than one goal at a time.

In summary, goals appear crosscutting each other all over a system and place restraints on system design decisions [Chung 2000]. In this context, we plan the construction of an ontology of goals intended to characterize them and detect system design problems or matchings (e.g., incompatibility, coupling, conflict, etc.) on the basis of the identified relationships between the goals of the different assets of an organization.

CHAPTER REFERENCES

[Chung 2000] Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: "Non-Functional Requirements in Software Engineering". Kluwer Academic Publishing, 2000

[OASIS 2006] OASIS, "Reference Model for Service Oriented Architecture 1.0", Committee Specification 1, 2 August 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

Bibliografía

REFERENCIAS

[Aalst 2005] van der Aalst, W.M.P., ter Hofstede, A.H.M.: "YAWL: Yet Another Workflow Language". Information Systems 30, 2005, pp. 245-275

[ACM 1992] ACM SIGCHI Curriculum Development Group. "ACM SIGCHI Curricula for Human Computer Interaction". New York: ACM, 1992. ISBN 0-89791-474-0

[Alberts 1993] Alberts, L.K.: "YMIR: an Ontology for Engineering Design". Ph.D. Thesis, University of Twente, Twente, The Netherlands, 1993

[Ambler 2003] Ambler, S. W., Nalbone, J., Vizdos, M.: "Enterprise Unified Process: Extending the Rational Unified Process". Prentice Hall PTR, 2003

[Andrade 2002] Andrade, L.F., Fiadeiro, J.L.: "Agility through Coordination". Information Systems, 27 (2002), Elsevier, 411-424

[Baader 2003] Baader, F., Calvanese, D., McGuineness, D., Nardi, D., Patel-Schneider, P.: "The Description Logic Handbook". Cambridge University Press, 2003

[Bajec 2005] Bajec, M., Krisper, M.: "A methodology and tool support for managing business rules in organisations", Information Systems, Volume 30, Issue 6, September 2005, pp. 423-443

[Bannon 1989] Bannon, L.J., Schmidt, K.: "CSCW: Four Characters in Search of a Context". Proc. 1st European Conf. on Comp. Supported Cooperative Work, pp. 358-372. Computer Sciences House, Sloug, UK, Sep. 1989

[Barjis 2008] Barjis, J.: "The Importance of Business Process Modeling in Software Systems Design". Science of Computer Programming, Vol. 71, 1, 2008, pp. 73-87

[Bass 2003] Bass, L., Clements, P., Kazman, R.: "Software Architecture in Practice". 2nd edition, Addison-Wesley, 2003

[Berardi 2005] Berardi, D., Calvanese, D., De Giacomo, G.: "Reasoning on UML class diagrams". Artificial Intelligence 168 (1–2), pp. 70-118 (2005)

[Bergholtz 2000] Bergholtz, M., Johannesson, P.: "Validating Conceptual Models - Utilising Analysis Patterns as an Instrument for Explanation Generation", NLDB 2000, LNCS, vol. 1959, 2001, pp. 325-339

[Berners-Lee 2001] Berners-Lee, T., Hendler, J., Lassila, O.: "The Semantic Web," Scientific American, May 2001, pp. 34-43

[Boman 1997] Boman, M., Bubenko, J.A., Johannesson, P., Wangler, B.: Conceptual Modelling, Prentice Hall Series in Computer Science, 1997

[Borst 1997] Borst, W.N.: "Construction of Engineering Ontologies for Knowledge Sharing and Reuse". PhD thesis, Universiteit Twente, Enschede, The Netherlands (1997)

[Bourimi 2007] Bourimi, M., Lukosch, S., Kühnel, F.: "Leveraging Visual Tailoring and Synchronous Awareness in Web-Based Collaborative Systems". CRIWG 2007. LNCS 4715, Springer-Verlag, Berlin, 2007, pp. 40-55

[Bravo 2007] Bravo, C., Duque, R., Gallardo, J., García, J., García, P.: "A Groupware System for Distributed Collaborative Programming: Usability Issues and Lessons Learned". International Workshop on Tools Support and Requirements Management for Globally Distributed Software Development, Munich, 2007, pp. 50-56

[Brockmans 2004] Brockmans, S., Volz, R., Eberhart, A., Loffler, P.: "Visual modeling of OWL DL ontologies using UML". Proceedings of 3rd International Semantic Web Conference, Hiroshima, Japan. LNCS 3298, pp. 198-213

[Brockmans 2006] Brockmans, S., Haase, P., Hitzler, P., Studer, R.: "A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies". ESWC 2006. LNCS 4011, pp. 303-316

[de Brujin 2005] de Bruijn, J., Lara, R., Polleres, A., Fensel, D.: "OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web". Proceedings of the 14th International Conference on World Wide Web (WWW '05). ACM, New York, NY, pp. 623-632

[Bubenko 1980] Bubenko, J.A.: "Information Modelling in de Context of Systems Development", Information Processing 1980, proceedings of IFIP Congress, North Holland, pp. 395-411

[Bubenko 2001] Bubenko, J.A., Persson, A., Stirna, J.: "D3 Appendix B: EKD User Guide", Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden, 2001

[Bunge 1977] Bunge, M.A.: "Ontology I: The Furniture of the World. Volume 3 of Treatise on Basic Philosophy". D. Reidel Publishing Company, Dordrecht, Holland (1977)

[Bunge 1979] Bunge, M.A.: "Ontology II: A World of Systems. Volume 4 of Treatise on Basic Philosophy". D. Reidel Publishing Company, Dordrecht, Holland (1979)

[Calvanese 2001] Calvanese, D., De Giacomo, G., Lenzerini, L.: "Identification constraints and functional dependencies in description logics". Proc. of the 17th Int. Joint conf. on Artificial Intelligence (IJCAI 2001), Seattle, WA, 2001, pp. 155-160

[Calvanese 2007] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R: "Can OWL model football leagues?". In Proc. of the 2007 International Workshop on OWL: Experiences and directions (OWLED 2007), 2007

[Calvanese 2008] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. "Path-based identification constraints in description logics". In Proc. of KR 2008, pp. 231-241

[Carstensen 2003] Carstensen, P.H., Schmidt, K.: "Computer supported cooperative work: new challenges to systems design", Handbook of Human Factors/Ergonomics, Asakura Publishing, Tokyo, 2003, pp. 619-636

[Chandrasekaran 1999] Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: "What are ontologies, and why do we need them?". Intelligent Systems and Their Applications, IEEE, vol.14, no.1, Jan/Feb 1999, pp. 20-26

[Chang 1990] Chang, C. C., Keisler, H. J.: "Model Theory, Studies in Logic and the Foundations of Mathematics", 3rd ed., 1990, Elsevier, ISBN 978-0-444-88054-3

[Chen 1976] Chen, P.: "The Entity-Relationship Model – Toward a Unified View of Data", ACM Transactions on Database Systems 1(1), pp. 9-36, 1976

[Codd 1970] Codd, E.F.: "A relational model of data for large shared data banks", Commun. ACM, vol. 13, no. 6, June 1970, pp. 377-387

[Cuenca-Grau 2008] Cuenca-Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: "OWL 2: The next step for OWL". Web Semantics: Science, Services and Agents on the World Wide Web, Volume 6, Issue 4, Semantic Web Challenge 2006/2007, November 2008, pp. 309-322

[Dardenne 1993] Dardenne, A., van Lamsweerde, A., Fickas, S.: "Goal-directed requirements acquisition", (1993) Science of Computer Programming, 20 (1-2), pp. 3-50

[DeSanctis 1987] DeSanctis, G., Gallupe, B.: "A Foundation for the Study of Group Decision Support Systems". Management Science 33 (5) (1987), pp. 589-609

[Dillenbourg 1995] Dillenbourg, P., et al.: "The Evolution of Research on Collaborative Learning". In: Reimann, P., Spada, H. (eds.)

Vol. Learning in humans and machines. Towards an interdisciplinary learning science, London, pp. 189–211 (1995)

[Dix 1998] Dix, A., Finlay, J., Abowd, G., Beale, R.: Human Computer Interaction, second edition. Prentice Hall (1998)

[Donini 1998] Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: "An epistemic operator for description logics". Artificial Intelligence, 100(1-2), April 1998, pp. 225-274

[Drummond 2006] Drummond, N., Rector, A.L., Stevens, R., Moulton, G., Horridge, M., Wang, H, Seidenberg, J.: "Putting OWL in Order: Patterns for Sequences in OWL", Proceedings of the 2nd OWL Experiences and Directions Workshop (OWL-ED 2006), Athens, Georgia, USA, 2006

[Duque 2009] Duque, R., Noguera, M., Bravo, C., Garrido, J.L., Rodríguez, M.L.: "Construction of interaction observation systems for collaboration analysis in groupware applications". Advances in Engineering Software, Elsevier, 2009. doi: 10.1016/j.advengsoft.2009.01.028

[Ellis 1991] Ellis, C.A., Gibbs, S.J., Rein, G.L.: "Groupware: Some Issues and Experiences", Communications of the ACM, Vol. 34, No. 1 (January 1991), pp. 38-58

[Euzenat 2001] Euzenat, J.: "An infrastructure for formally ensuring interoperability in a heterogeneous semantic web". Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001, pp. 345-360

[Evermann 2003] Evermann, J.: "Using Design Languages for Conceptual Modelling: The Uml Case". PhD thesis, University of British Columbia, Canada, 2003

[Evermann 2005] Evermann, J., Wand, Y.: "Toward Formalizing Domain Modeling Semantics in Language Syntax. IEEE Transactions on Software Engineering, vol. 31, no. 1, pp. 21-37, January 2005

[Evermann 2005b] Evermann, J., Wand, Y.: "Ontology based object-oriented domain modelling: fundamental concepts", Requirements Engineering, 10 (2), 2005, pp. 146-160

[Evermann 2008] Evermann, J.: "A UML and OWL Description of Bunge's Upper-level Ontology Model". Software and Systems Modeling. In press (doi:10.1007/s10270-008-0082-3)

[Fdez-López 2002] Fernández-López, M., Gómez-Pérez, A.: "Overview and analysis of methodologies for building ontologies". The Knowledge Engineering Review, Vol. 17:2, 2002, Cambridge University Press, 129-156

[Farquhar 1997] Farquhar, A., Fikes, R., Rice, J.: "The Ontolingua Server: a Tool for Collaborative Ontology Construction".

International Journal of Human-Computer Studies (IJHCS), 46(6): pp. 707-728, 1997

[Fensel 2000] Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., Klein, M.: "OIL in a nutshell". Proceedings of the European Knowledge Acquisition Conference (EKAW-2000). LNAI 1937, Springer-Verlag, 2000, pp. 1-16

[Fiadeiro 1992] Fiadeiro, J., Sernadas, C., Maibaum, T., Sernadas, A.: "Describing and Structuring Objects for Conceptual Schema Developments", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 117-138

[Fox 1998] Fox, M., Barbuceanu, M., Gruninger, M., Lin., J.: "An Organizational Ontology for Enterprise Modeling". In Simulating Organizations: Computational Models of Institutions and Groups. AAAI Press / MIT Press, 1998, pp. 131-152

[Gamma 1995] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, Reading, MA (1995)

[Gangemi 2007] Gangemi, A., Gómez-Pérez, A., Presutti, V., Suárez-Figueroa, M.C.: "Towards a Catalog of OWL-based Ontology Design Patterns". CAEPIA 07, (2007-11)

[Garrido 2002] Garrido, J.L., Gea, M., Padilla, N., Cañas, J.J., Waern, Y.: "AMENITIES: Modelado de Entornos Cooperativos". In: Aedo, I., Díaz, P., Fernández, C. (eds.): Actas del III Congreso Internacional Interacción Persona-Ordenador 2002 (Interacción'02), Madrid, Spain (Mayo 2002), pp. 97-104

[Garrido 2002b] Garrido, J.L., Gea, M.: "A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling". In: Interactive Systems - Design, Specification and Verification. Revised papers. LNCS 2545. Springer (2002), pp. 16-28

[Garrido 2003] Garrido, J.L., "AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas". Tesis Doctoral, Granada 2003

[Garrido 2005] Garrido, J.L., Gea, M., Rodríguez, M.L.: "Requirements Engineering in Cooperative Systems". In: Requirements. Engineering for Sociotechnical Systems, Chapter XIV, IDEA GROUP, USA, 2005, pp. 226-244

[Garrido 2007] Garrido, J.L., Noguera, M., González, M., Hurtado, M.V., Rodríguez, M.L.: "Definition and Use of Computation Independent Models in an MDA-Based Groupware

Development Process". Science of Computer Programming 66 (1), 2007, pp. 25-43

[Garrido 2008] Garrido, J.L., Hurtado, M.V., Noguera, M., Zurita, J.M.: "Using a CBR Approach based on Ontologies for Recommendation and Reuse of Knowledge Sharing in Decision Making". 8th International Conference on Hybrid Intelligent Systems (HIS 2008). IEEE Press, 2008, pp. 837-842

[Genesereth 1987] Genesereth, M.R., Nilsson, N.J.: "Logical Foundations of Artificial Intelligence". San Mateo, CA: Morgan Kaufmann Publishers, 1987

[Genesereth 1998] Genesereth. M.: "Knowledge Interchanged Format (KIF)"; 1998 http://logic.stanford.edu/kif/kif.html

[Gómez-Pérez 1999] Gómez-Pérez, A.: "Tutorial on Ontological Engineering". En IJCAI'99

[Gómez-Pérez 2004] Gómez-Pérez, A., Ferández-López, M., Corcho, O.: "Ontological Engineering: with Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web". Springer-Verlag, London, 2004

[Green 2007] Green, P., Rosemann, M., Indulska, M., Manning, C.: "Candidate interoperability standards: An ontological overlap analysis", Data & Knowledge Engineering, Volume 62, Issue 2, August 2007, pp. 274-291

[Grimm 2005] Grimm, S., Motik, B.: "Closed-World Reasoning in the Semantic Web through Epistemic Operators". Proceedings of the OWL Experiences and Directions Workshop, Galway, Ireland, 2005

[Grosso 2002] Grosso, W.: "The Protégé Axiom Language and Toolset (PAL)", 2002 (Last updated: September, 2005): http://protege.stanford.edu/plugins/paltabs/paldocumentation/index.html

[Gruber 1993] Gruber, T. R.: "A translation approach to portable ontology specifications". Knowledge Acquisition 5, 1993, pp. 199-220

[Gruber 1995] Gruber, T.R., "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928

[Grudin 1988] Grudin, J.: "Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces". Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work. CSCW '88. ACM, New York, NY, pp. 85-93

[Grudin 1994] Grudin, J.: "Computer-supported cooperative work: History and focus." IEEE Computer, 27, 5, pp. 19-26 (1994)

[Gruninger 2000] Gruninger, M., Atefi, K., Fox, M.S.: "Ontologies to support process integration in enterprise engineering". Kluwer Academic Publishers 2001, Computational and Mathematical Organization Theory 6, 2000, pp. 381–394

[Guarino 1995] Guarino, N., Giaretta, P.: "Ontologies and Knowledge Bases: Towards a Terminological Clarification". In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam, 1995, pp. 25-32

[Guarino 1997] Guarino, N.: "Understanding, Building, and Using Ontologies: A Commentary to 'Using Explicit Ontologies in KBS Development', by van Heijst, Schreiber, and Wielinga". International Journal of Human and Computer Studies 46, 1997, pp. 293-310

[Guarino 1998] Guarino, N.: "Formal Ontology in Information systems". Proceedings of FOIS '98, (Trento, Italy, June, 1998). IOS Press, Amsterdam, 1998, pp. 3-15

[van Heijst 1997] van Heijst, G., Schreiber, A.T., Wielinga, B.J.: "Using explicit ontologies in KBS development". International Journal of Human and Computer Studies 46, 1997, pp. 183-292

[Henderson 2004] Henderson, P., Crouch, S., Walters, R. J.: Information Invasion in Enterprise Systems: Modelling, Simulating and Analysing System-level Information Propagation. 6th International Conference on Enterprise Information Systems (ICEIS 2004) 1, pp. 473-481

[Horrocks 2006] Horrocks, I., Kutz, O., Sattler, U.: "The even more irresistible SROIQ". In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, 2006, pp. 57-67

[Hurtado 2002] Hurtado, M.V., "Un Modelo de Integración Evolutivo entre Sistemas de Información y Sistemas de Ayuda a la Decisión". Tesis Doctoral, Granada 2002

[Hurtado 2007] Hurtado, M.V., Noguera, M., Rodríguez, M.L., Garrido, J.L., Chung, L.: "An Ontology-based Approach to the Modeling of Collaborative Enterprise Processes: Dynamic Managing of Functional Requirements". ENASE 2007, Barcelona, España. INSTICC, 2007, pp. 87-94

[ISO 2007] ISO/IEC FDIS 24707:2007(E) Information technology – Common Logic (Common Logic) – A framework for a family of logic-based languages. Available at http://cl.tamu.edu/

[Jacovi 2006] Jacovi, M., Soroka, V., Gilboa-freedman, G., Ur, S., Shahar, E., Marmasse, N.: "The chasms of CSCW: a citation graph analysis of the CSCW conference". Proceedings of the 2006 ACM Conference on Computer-Supported Cooperative Work, ACM Press, pp. 289-298

[Jaekel 2005] Jaekel, F.W., Perry, N., Campos, C., Mertins, K., Chalmeta, R.: "Interoperability Supported by Enterprise Modelling", LNCS 3762, Springer-Verlag, Berlin, 2005, pp. 552-561

[Jarke 1994] Jarke, M., Pohl, K.: "Establishing visions in context: Towards a model of requirements processes". EMISA Forum 4 (2), pp. 49-62

[Jarrar 2007] Jarrar, M.: "Mapping ORM into the SHOIN/OWL Description Logic - Towards a Methodological and Expressive Graphical Notation for Ontology Engineering". LNCS, Vol. 4805, 2007, pp. 729-741

[Jensen 1996] Jensen, K.: "Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use". Second Edition, Springer (1996)

[Johansen 1988] Johansen, R.: "Groupware: Computer Support for Business Teams". The Free Press, 1988

[Jonkers 2004] Jonkers, H., Lankhorst, M.M., van Buuren, R., Hoppenbrouwers, S., Bonsangue, M.M., van der Torre, L.W.N.: "Concepts for Modeling Enterprise Architectures". Int. J. Cooperative Inf. Syst. 13, 2004, pp. 257-287

[Kalfoglou 2003] Kalfoglou, Y., Schorlemmer, M.: "Ontology mapping: the state of the art". Knowl. Eng. Rev. 18, 1 (Jan. 2003), pp. 1-31

[Kalyanpur 2005] Kalyanpur, A., Parsia, B., Sirin, E., Cuenca-Grau, B., Hendler, J.: "SWOOP: a web ontology editing browser". Journal of Web Semantics 4 (2) (2005), pp. 144-153

[Karp 1999] Karp, P.D., Chaudhri, V.K., Thomere, J.: "XOL: An XML-based ontology exchange language". Version 0.4, 1999. http://www.ai.sri.com/ ~pkarp/xol/xol.html

[Katz 2005] Katz, Y., Parsia, B.: "Towards a Nonmonotonic Extension to OWL". Proceedings of the OWL Experiences and Directions Workshop, Galway, Ireland, 2005

[Keet 2007] Keet, C.M.: "Mapping the Object-Role Modeling language ORM2 into Description Logic language $\mathcal{DLR}$ifd". Technical Report KRDB07-2, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy. 15 February 2007. arXiv:cs.LO/0702089v1

[Kishore 2004] Kishore, R., Zhang, H., Ramesh, R.: "A Helix-Spindle Model for Ontological Engineering". Communications of ACM, February 2004/Vol. 47, No. 2, pp. 69-75

[Knublauch 2004] Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.: "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", LNCS 3298, 2004, pp. 229-243

[Kogut 2002] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J.: "UML for ontology development". Knowl. Eng. Rev. 17, 1 (Mar. 2002), pp. 61-64

[Kristensen 2004] Kristensen, L.M., Jorgensen, J.B., Jensen, K.: "Application of Coloured Petri Nets in System Development". ACPN 2003. LNCS 3098, Springer-Verlag (2004), pp. 626-685

[Lange 2006] Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: "UML Software Architecture and Design Description". IEEE Software, 23 (2006), pp. 40-46

[Lassila 2001] Lassila, O., McGuinness, D.: "The role of frame-based representation on the semantic web". Technical Report KSL-01-02. Knowledge System Laboratory, Stanford University, Stanford, CA, 2001

[Letier 2001] Letier, E.: "Reasoning about Agents in Goal-Oriented Requirements Engineering". PhD thesis, Louvain-la-Neuve, Belgium, 2001

[Letier 2004] Letier, E. and van Lamsweerde, A.: "Reasoning about partial goal satisfaction for requirements and design engineering". In Proceedings of the 12th ACM SIGSOFT Twelfth international Symposium on Foundations of Software Engineering (Newport Beach, CA, USA, October 31 - November 06, 2004). ACM, New York, NY, pp. 53-62

[Liu 2001] Liu, K., Sun, L., Dix, A., Narasipuram, M.: "Norm-based agency for designing collaborative information systems". Information Systems Journal 11 (3), 2001, pp. 229-247

[Loom 1995] Tutorial for Loom version 2.1., http://www.isi.edu/isd/LOOM/documentation/tutorial2.1.html, May 1995

[Luke 2000] Luke, S., Heflin, J.D.: "SHOE 1.01. Proposed Specification". Technical Report. Parallel Understanding Systems Group. Department of Computer Science. University of Maryland. http://www.cs.umd.edu/projects/plus/SHOE/spec.html

[Lynch 1990] Lynch, K.J., Snyder, J.M., Vogel, D.R., McHenry, W.K.: "The Arizona Analyst Information System: Supporting Collaborative Research on international Technological Trends". Proceedings of IFIP WG8.4 Conference on Multi-User Interfaces and Applications, Crete, Greece (1990)

[MacGregor 1999] MacGregor, R.: "Retrospective on LOOM". Technical Report. Information Sciences Institute. University of Southern California. http://www.isi.edu/isd/LOOM/papers/macgregor/Loom_Retrospective.html

[Malone 1990] Malone, T.W., Crowston, K.: "What is Coordination Theory and How Can It Help Design Cooperative Work Systems". Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'90). ACM Press, New York (1990), pp. 357-370

[Miller 2004] Miller, G., Ambler, S., Cook, S., Mellor, S., Frank, K., Kern, J.: "Model driven architecture: the realities, a year later". In Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (Vancouver, BC, CANADA, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, pp. 138-140

[Moody 2005] Moody, D.L.: "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions", Data & Knowledge Engineering, Volume 55, Issue 3, Quality in conceptual modeling - Five examples of the state of art, December 2005, pp. 243-276

[Motik 2008] Motik, B., Cuenca Grau, B., and Sattler, U.: "Structured objects in owl: representation and reasoning". Proc. of the 17th International Conference on World Wide Web (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 555-564

[Mylopoulos 1990] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M.: "Telos: representing knowledge about information systems". ACM Trans. Inf. Syst. 8, 4 (Oct. 1990), pp. 325-362

[Mylopoulos 1992] Mylopoulos, J.: "Conceptual Modelling and Telos" in "Conceptual Modelling, Databases and CASE", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 49-68

[Noguera 2006] Noguera, M., Hurtado, M.V., Garrido, J.L.: "An Ontology-Based Scheme Enabling the Modeling of Cooperation in Business Processes", LNCS 4277, Springer-Verlag (2006), pp. 863-872

[Noguera 2009] Noguera, M., Hurtado, M.V., Rodríguez, M.L., Chung, L., Garrido, J.L.: "Ontology-driven Analysis of UML-Based Collaborative Processes using OWL-DL and CPN". Science of Computer Programming, (in press, under review), 2009

[Noy 2001] Noy, N.F., McGuinness, D.L.: "Ontology development 101: A guide to creating your first ontology". Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001

[Nurcan 1998] Nurcan, S.: "Analysis and Design of Co-operative Work Processes: A Framework". Information and Software Technology, Elsevier, 40(3), (1998), pp. 143-156

[Oberle 2005] Oberle, D., Staab, S., Studer, R., Volz, R.: "Supporting application development in the semantic web". ACM Trans. Interet Technol. 5, 2 (May. 2005), pp. 328-358

[Olivé 2007] Olivé, A.: "Conceptual Modeling of Information Systems", Chapters 1 and 17, Springer-Verlag, 2007

[OASIS 2006] OASIS, "Reference Model for Service Oriented Architecture 1.0", Committee Specification 1, 2 August 2006, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[OMG 2001] OMG, "Unified Modeling Language Specification", version 1.5, formal/2001-09-67. http://www.omg.org/spec/UML/1.4/

[OMG 2003] Object Mangagement Group: Model Driven Architecture (MDA) Guide, v1.0.1, OMG, omg/03-06-01

[OMG 2006] OMG, "Object Constraint Language. OMG Available Specification. Version 2.0". (OCL 2.0 2006-06-01). Latest version is available at: http://www.omg.org/docs/formal/06-05-01.pdf

[OMG 2007] OMG, "Unified Modeling Language: Superstructure", version 2.1.1 (with change bars), formal/2007-02-03. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[OMG 2007b] OMG, MDA: "Committed Companies and Their Products" http://www.omg.org/mda/committed-products.htm

[OMG 2007c] OMG, "Ontology Definition Metamodel". OMG Adopted Specification. OMG Document Number: ptc/2007-09-09". Available at: http://www.omg.org/docs/ptc/07-09-09.pdf

[Pahl 2005] Pahl, C.: "Ontology Transformation and Reasoning for Model-Driven Architecture". CoopIS/DOA/ODBASE 2005, LNCS 3761, 2005, pp. 1170-1187

[Parsia 2005] Parsia, B., Sirin, E., Kalyanpur, A.: "Debugging OWL ontologies". Proceedings of the 14th International Conference on World Wide Web (WWW '05). ACM, New York, pp. 633-640

[Parsia 2008] Parsia, B., Sattler, U., Schneider, T.: "Easy Keys for OWL". In Proc. of the 2008 International Workshop on OWL: Experiences and directions (OWLED 2008)

[Paternò 2000] Paternò, F.: "Model-based Design and Evaluation of Interactive Applications". Springer-Verlag (2000)

[Peltz 2003] Peltz C.: "Web Services Orchestration and Choreography". IEEE Computer 36 (2003), pp. 46-52

[Penichet 2008] Penichet, V.M.R., Lozano, M.D., Gallud, J.A.: "An Ontology to Model Collaborative Organizational Structures in CSCW Systems". International Chapter Book:

Engineering the User Interface. From Research to Practice. Ed. Springer, 2008, pp. 127-139

[Peppard 2001] Joe Peppard (2001) Bridging the gap between the IS organization and the rest of the business: plotting a route Information Systems Journal 11 (3), pp. 249–270

[Plisson 2007] Plisson, J., Ljubic, P., Mozetic, I., Lavrac, N.: "An Ontology for Virtual Organization Breeding Environments". IEEE Transactions on Systems, Man, and Cybernetics, Part C 37(6), 2007, pp. 1327-1341

[Riesbeck 1989] Riesbeck, C.K., Schank, R.C.: "Inside Case-Based Reasoning". Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989

[Rolland 1992] Rolland, C., Cauvet, C.: "Trends and perspectives in conceptual modelling", in: P. Loucopoulos, R. Zicari (Eds.), Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development, John Wiley & Sons, 1992, pp. 27-48

[Rumbaugh 1999] Rumbaugh, J., Jacobson, I., Booch, G.: "The Unified Modeling Language - Reference Manual". Addison-Wesley (1999)

[Russell 2006] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wohed, P.: "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling", APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual Modelling, Australian Computer Society, 53, 2006, pp. 95-104

[Schlichter 1998] Schlichter, J.H., Koch, M., Bürger, M.: "Workspace Awareness for Distributed Teams". Coordination Technology for Collaborative Applications 1998: pp. 199-218

[Schmid 1975] Schmid, J. and Swenson, R.: "On the Semantics of the Relational Data Model", Proceedings ACM SIGMOD International Conference on Management of Data, pp. 211-223, 1975

[Schmidt 2006] Schmidt, D.C.: "Guest Editor's Introduction: Model-Driven Engineering". IEEE Computer, 39 (2006), pp. 25-31

[Schreiber 1994] Schreiber, G., Wielinga, B., de Hoog, R., Akkermans, H., Van de Velde, W.: "CommonKADS: A Comprehensive Methodology for KBS Development," IEEE Expert: Intelligent Systems and Their Applications, vol. 09, no. 6, pp. 28-37, Dec., 1994

[Schreiber 1994b] Schreiber, G., Wielinga, B., Akkermans, H., Van de Velde, W., Anjewierden, A.: "CML: The CommonKADS Conceptual Modelling Language". A Future for Knowledge Acquisition, 8th European Knowledge Acquisition

Workshop. LNCS 867, Springer-Verlag, Berlin, 1994, pp. 1-25

[Seidenberg 2007] Seidenberg, J., Rector, A.L.: "The State of Multi-User Ontology Engineering". WoMO 2007

[Seidewitz 2003] Seidewitz, E.: "What models mean". IEEE Software 20 (2003), pp. 26-32

[Sirin 2007] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: "Pellet: A practical OWL-DL reasoner", Journal of Web Semantics, 5(2), 2007, pp. 51-53

[Sure 2005] Sure, Y., Bloehdorn, S., Haase, P., Hartmann, J., Oberle, D.: "The SWRC ontology - semantic web for research communities". In: Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Covilha, Portugal. LNCS 3808, Springer-Verlag, Berlin, 2005, pp. 218-231

[Tsarkov 2006] Tsarkov, D., Horrocks, I.: "FaCT++ Description Logic Reasoner: System Description". Proc. of the International Joint Conference on Automated Reasoning (IJCAR 2006), 2006, LNAI 4130, pp. 292-297

[Uschold 1996] Uschold, M., Gruninger, M.: "Ontologies: Principles, Methods and Applications". Knowledge Engineering Review 11 (2), 1996, pp. 93-155

[Viégas 2004] Viégas, F.B., Wattenberg, M., Kushal, D.: "Studying cooperation and conflict between authors with history flow visualizations". Proceedings of the 2004 Conference on Human Factors in computing Systems, (2004), pp. 575-582

[Wand 1995] Wand, Y., Monarchi, D.E., Parsons, J., Woo, C.C.: "Theoretical foundations for conceptual modelling in information systems development", Decision Support Systems, Volume 15, Issue 4, December 1995, pp. 285-304

[Williams 2002] Williams, L., Kessler, R.: "Pair Programming Illuminated". Addison-Wesley (2002)

[W3C DAML 2001] W3C, (Editores) Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A. (Editores):"DAML+OIL (March 2001) Reference Description". W3C Note, 18 December 2001. http://www.w3.org/TR/daml+oil-reference

[W3C HTML 1999] Raggett, D., le Hors, A., Jacobs, I.: "HTML 4.01 Specification". W3C Recommendation, 24 December 1999, http://www.w3.org/TR/html401/

[W3C ODA 2006] W3C, (Editores) Tetlon, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., Kendall, E.: "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering". W3C Editors' Draft of 11 February

2006, http://www.w3.org/2001/sw/BestPractices/SE/ODA/

[W3C OWL 2004] W3C, (Editores) Smith, M.K., Welty, C., McGuinness, D.L.: "OWL Web Ontology Language: Guide". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-guide/

[W3C OWL 2004b] W3C, (Editores) McGuinness, D.L., van Harmelen, F. (Editores): "OWL Web Ontology Language: Overview". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-features/

[W3C OWL 2004c] W3C, (Editores) Dean, M., Schreiber, G. (Editores): "OWL Web Ontology Language Reference". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/owl-ref/

[W3C OWL 1.1 2007] W3C, (Editores) Patel-Schneider, P.F., Horrocks, I.: "OWL 1.1 Web Ontology Language: Overview". Editor's Draft of 23 May 2007, http://webont.org/owl/1.1/overview.html

[W3C OWL 2 2008] W3C, (Editores) Motik, B., Patel-Schneider, P.F., Parsia, B.: "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax". W3C Working Draft, 02 December 2008. http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/

[W3C OWL 2 2008b] W3C, (Editores) Motik, B., Patel-Schneider, P.F., Cuenca-Grau, B.: "OWL 2 Web Ontology Language: Direct Semantics". http://www.w3.org/TR/2008/WD-owl2-semantics-20081202/

[W3C OWL 2 2008c] W3C, (Editores) Parsia, B., Patel-Schneider, P.F.: "OWL 2 Web Ontology Language: Primer". http://www.w3.org/TR/owl2-primer/

[W3C OWL-S 2004] W3C, (Editor) Martin. D: "OWL-S: Semantic Markup for Web Services", W3C Member Submission, 22 November 2004, http://www.w3.org/Submission/OWL-S/

[W3C RDF 2004] W3C, (Editores) Manola, F., Miller, E. (Editores): "RDF Primer". W3C Recommendation, 10 February 2004. http://www.w3.org/TR/rdf-primer/

[W3C RDF 2004b] W3C, (Editores) Brickley, D., Guha, R.V. (Editores): "RDF Vocabulary Description Language 1.0: RDF Schema". W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-schema/

[W3C SWBP 2006] W3C, (Editores) Noy, N.F., Rector, A.:"Defining N-ary Relations on the Semantic Web". W3C Working Group Note, 12 April 2006, http://www.w3.org/TR/swbp-n-aryRelations/

[W3C SWBP 2006b] Knublauch, H., Oberle, D., Tetlow, P., Wallace, E.: "A Semantic Web Primer for Object-Oriented Software

Developers". W3C Working Group Note, 9 March 2006, http://www.w3.org/TR/sw-oosd-primer/

[W3C SWRL 2004] W3C, (Editores) Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: "SWRL: A Semantic Web Rule Language Combining OWL and RuleML". W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/SWRL/

[W3C WSA 2004] W3C, (Editores) Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, I.M., Ferris, C., Orchard, D.: "Web Services Architecture". W3C Working Group Note, 11 February 2004. http://www.w3.org/TR/ws-arch/

[W3C WSDL 2007] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", W3C Recommendation, 26 June 2007, http://www.w3.org/TR/wsdl20/

[W3C XML 2000] W3C, (Editores) Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: "Extensible Markup Language (XML) 1.0". W3C Recommendation, 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[W3C XML-S 2004] W3C, (Editores) Fallside, D.C., Walmsley, P. (Editores): "XML Schema Part 0: Primer". W3C Recommendation, 28 October 2004, http://www.w3.org/TR/xmlschema-0/

[Xexeo 2005] Xexeo, G., Vivacqua, A., De Souza, J.M., Braga, B., D'Almeida Jr., J.N., Almentero, B.K., Castilho, R., Miranda, B.: "COE: A collaborative ontology editor based on a peer-to-peer framework". (2005) Advanced Engineering Informatics, 19 (2), pp. 113-121

[Xia 2007] Xia, F., Jihua, S.: "A Study in Knowledge Ontology Building in the Area of Knowledge Engineering". Third International Conference on Semantics, Knowledge and Grid, 29-31 Oct. 2007, pp. 586-587