Universidad de León

Departamento de Ingeniería Eléctrica
y de Sistemas y Automática

Modelos de Conocimiento Basados en Ontologías para la Construcción de Software en el Dominio de la Ingeniería de Control

Tesis doctoral

Isaías García Rodríguez
León, marzo de 2008

A mi familia

A todos los que me han ayudado durante estos años

A todo el que encuentre una idea útil en estas páginas

Agradecimientos

Mi más profundo agradecimiento a todos los que me han ayudado, guiado, alentado y apremiado durante la realización de esta tesis. En especial, agradecer:

A mi tutor, Ángel Alonso, por haberme introducido en esta línea de investigación y guiado en el proceso de realización de la tesis.

A mis compañeros, por ayudarme en la realización de la tesis de todas las formas que puedan concebirse. En particular gracias a Javier Alfonso, Héctor Aláiz y, en especial, a Carmen Benavides.

A mis amigos, que me animaron a continuar y llegar hasta aquí. Además de los citados en el párrafo anterior, a Ramón Ángel, Perfecto, Fran, Alija, Cynthia y Marta.

A los compañeros de departamento y escuela, que me animaron durante estos años. En particular a Maria José y Maribel.

Espero que el trabajo realizado sea merecedor de toda la ayuda recibida.

...porque estamos hechos de palabras y las cosas también; porque nosotros somos tan sólo memoria y las cosas existen y son verdaderas cuando se dejan vestir, mansas, del mundo de palabras, del aura iridiscente y temblorosa de voltios y vatios y protones y neutrones con la cual los positivistas enmascararon el grito del relámpago…

Fernando del Paso – Palinuro de México

Lista de acrónimos y abreviaturas utilizadas

AAAI – Association for the Advancement of Artificial Intelligence.
ACE – Advances in Control Education (Simposio internacional de IFAC).
ACM – Association of Computing Machinery.
AI – Artificial Intelligence.
AIED – Artificial Intelligence in Education (Simposio internaiconal).
AIMSA – (International Conference on) Artificial Intelligence: Methodology, Systems, and Applications.
AIS – Association for Information Systems.
AMIA – American Medical Informatics Association.
API – Application Program Interface – Interfaz de Programación de Aplicaciones.
ASME – American Society of Mechanical Engineers.
CACE – Computer-Aided Control Engineering.
CACSD – Computer-Aided Control Systems Design.
CAD – Computer Aided Design.
CASE – Computer-Aided Software Engineering.
CIM – Computation-Independent Model.
CL – Common Logic.
CLIPS – C-Language Integrated Production System.
CSLI – Center for the Study of Language and Information.
CWA – Closed World Assumption.
DAML – Darpa Agent Markup Language.
DARPA – Defense Advanced Research Projects Agency.
DATE – Design, Automation and Test in Europe
DIG – DL Implementation Group.
DL – Description Logic(s).
EC – Existential Conjunctive subset of logic.
ECAI – European Conference on Artificial Intelligence.
ECBS – . (IEEE International Conference and Workshop on the**)** Engineering of Computer Based Systems
ECOOP – European Conference on Object-Oriented Programming.
ECSTASY – Environment for Control System Theory and Synthesis.
EDIProD – Engineering Design in Integrated Product Development.
EJS – Easy Java Simulations.
EKAW – European Knowledge Acquisition Workshop – conferencia ahora denominada "International Conference on Knowledge Engineering and Knowledge Management".
EON – Evaluation of ONtologies for the Web, congreso de carácter internacional.
ESA – European Space Agency.
FMA – Foundational Model of Anatomy.

FOIS – Formal Ontology in Information Systems (congreso internacional).
FOL – First Order Logic.
FOPL – First Order Predicate Logic (igual que FOL).
GE-MEAD – General Electric – Multidisciplinary Expert-aided Analysis and Design.
GFP – Generic Frame Protocol.
HTML – HyperText Markup Language
IA – Inteligencia Artificial
IBCE – Internet Based Control Education.
ICCAD – (IEEE) International Conference on Computer Aided Design.
ICBGM – International Conference on Bond Graph Modeling and Simulation.
ICOTEC – International Committee for the History of Technology.
IEE – Institution of Electrical Engineers.
IEEE – Institute of Electrical and Electronics Engineers
IEEE CSS – IEEE Control Systems Society.
IFAC – Internacional Federation on Automatic Control.
IFIP – International Federation for Information Processing.
IJCAI – International Joint Conferences on Artificial Intelligence.
IMACS – International Association for Mathematics and Computers in Simulation.
INCO – Ingeniería del Conocimiento
INCOSE – International Council of Systems Engineering.
IS – Ingeniería del Software.
ISO – International Organization for Standarization.
ISWC – International Semantic Web Conference.
ITS – Intelligen Tutoring System.
JESS – Java Expert System Shell.
KACTUS – modelling Knowledge About Complex Technical systems for multiple Use.
KAW – Knowledge Acquisition (Modeling, and Management) Workshop.
KE – Knowledge Engineering.
KES – Knowledge based and Intelligent Engineering Systems.
KEML – Knowledge Engineering Methods and Languages.
KIF – Knowledge Interchange Format.
KQML – Knowledge Query and Manipulation Language.
KRL – Knowledge Representation Language.
KSE – (ARPA) Knowledge Sharing Effort.
KSL – Knowledge Systems Laboratory. Grupo de investigación y departamento de la universidad de Stanford
LISP – LISt Processor.
MathML – Mathematics Markup Language.
MATLAB – MATrix LABoratory.
MDA – Model Driven Architectures.
MDSE – Model-Driven Systems Engineering.
MIT – Massachusetts Institute of Technology.
NAF – Negation As Failure.
NASA – National Aeronautics and Space Administration.
NSF – National Science Foundation.

OCML – Operational Conceptual Modelling Language.
OKBC – Open Knowledge-Base Connectivity.
ODM – Ontology Definition Metamodel
OIL – Ontology Inference Layer.
OLMECO – Open Library for Models of MEchatronical Components.
OMG – Object Management Group.
OPS – Official Production System.
OWA – Open World Assumption.
OWL – Web Ontology Language.
OWLED – OWL: Experiences and Directions (Congreso internacional).
PACT – Palo Alto Collaborative Testbed.
PAL – Protégé Axiom Language.
PDTAG – Product Data Technology Advisory Group.
PDE – Product Data Exchange (y congreso del mismo nombre patrocinado por la NASA y la ESA).
PGEF – Pan Galactic Engineering Framework.
PSM – Problem Solving Method.
PSIG – Programming Special Interest Group (OMG).
RDF – Resource Description Framework.
RDFS – RDF Schema.
SCADA – Supervisory Control and Data Adquisition.
SE – Software Engineering.
SETF – Software Engineering Task Force (W3C).
SGBD – Sistema de Gestión de Bases de Datos.
SIGART – Special Interest Group on ARTificial intelligence (dentro de ACM).
SIGMOD – Special Interest Group on Management Of Data (dentro de ACM).
SIGSEMIS – Special Interest Group on Semantic Web and Information Systems (dentro de AIS).
SISO – Single input – single output.
SKG – (IEEE International Conference on) Semantics, Knowledge, and Grid.
SoSyM – Software and Systems Modeling (Journal).
SSEE – Sistemas Expertos.
STEP – Standard for the Exchange of Product data model.
SUMO – Suggested Upper Merged Ontology.
SUO – Standard Upper Ontology.
SysML – Systems Modeling Language
SWEET – Semantic Web for Earth and Environmental Terminology.
SWRL – Semantic Web Rule Language
TIC – Tecnologías de la Información y la Comunicación.
TM – Topic Maps.
UML – Unified Modeling Language.
UNA – Unique Name Assumption.
XML – Extensible Markup Language.
XSW – Workshop XML Technologies for the Semantic Web.
W3C – World Wide Web Consortium.
WiSME – Workshop in Software Model Engineering.

Índice de figuras

Prefacio

La presente tesis doctoral se ha realizado en el periodo 2002-2008 dentro de los proyectos "La ingeniería de control en la sociedad del conocimiento: representación del contenido científico técnico en el ordenador", subvencionado por la Junta de Castilla y León (con código LEA024) y “Herramienta software para la ayuda al diseño de controladores basada en modelos de conocimiento”, subvencionado por el Plan Nacional de Diseño y Producción Industrial dentro del Plan I+D del Ministerio de Ciencia y Tecnología (con código DPI2007-64408).

El tema abordado en esta tesis es la representación del conocimiento del dominio de la ingeniería de control en las aplicaciones informáticas. En concreto se presenta y estudia el uso de las técnicas de modelado del conocimiento provenientes del campo de la inteligencia artificial como forma de hacer frente a alguna de las necesidades que presenta el software en esta disciplina. Para comprobar la validez de esta aproximación se estudia y lleva a cabo la construcción de una estructura conceptual (una ontología) que recoge el conocimiento existente en un subdominio de esa disciplina.

¿Se lo pueden imaginar? ¡Tenían bibliotecas en las que los libros no hablaban entre sí!

Marvin Minsky, hablando del presente desde un futuro imaginado.

Introducción

La segunda mitad del siglo XX ha estado marcada por la creciente informatización de los procesos y actividades en todos los ámbitos de la sociedad. Este hecho es provocado, en parte, por la necesidad de automatización del tratamiento de la información, cuyo volumen e importancia estratégica no ha dejado de aumentar durante esta etapa, razón por la cual se acuñó el término "sociedad de la información" para la misma. El comienzo del siglo XXI ha visto cómo el término "sociedad del conocimiento"[1] ha venido a sustituir al de "sociedad de la información". Este nuevo término hace referencia a la necesidad de manejar mayor cantidad de información y a unos niveles de complejidad más altos. Asimismo, los procesos de toma de decisión basados en esta información son también más complejos y sensibles.

Estos hechos hacen que sea necesario traspasar cada vez más procesamiento de la información desde el ser humano a los ordenadores, lo cual ha provocado que, actualmente, la informática se encuentre en una etapa de evolución para hacer frente a estos nuevos retos. Desde todos los ámbitos de la disciplina se postula que la solución pasa por encontrar mecanismos que permitan pasar de almacenar y manejar estructuras de datos a poder trabajar con la semántica y el conocimiento.

El nuevo paradigma de creación de software postulado para lograr los objetivos mencionados es conocido bajo el calificativo de “basado en modelos del conocimiento”. Este paradigma surge de los trabajos desarrollados dentro de las disciplinas de la ingeniería del software y de la inteligencia artificial, aunque es en esta última en la que comenzó la preocupación por el problema de la "representación del conocimiento" y la que ha tratado en mayor profundidad el tema.

[1] El término “sociedad del conocimiento” surge en el ámbito de las organizaciones empresariales a principios de los años 1970 aunque la idea no fue desarrollada hasta los años 1990, haciendo referencia a la importancia del conocimiento o “valor intangible” que radica en cada una de las personas que forman una organización. Bajo este punto de vista el término se relaciona directamente con el de “gestión del conocimiento”. El término también aparece algunas veces citado en relación con el denominado proceso de globalización de las economías; de este modo la sociedad del conocimiento será aquella en la que el acceso al mismo sea universal. Ambos puntos de vista son compatibles con la acepción más general expresada aquí.

El problema de la representación del conocimiento ha sido la principal preocupación de la ingeniería del conocimiento, disciplina nacida en los años 1980s dentro del campo de la inteligencia artificial. El origen de la ingeniería del conocimiento se remonta a las experiencias en la construcción de los sistemas expertos y ha evolucionado hasta el denominado "paradigma del modelado del conocimiento". Los "modelos de dominio"[2], las estructuras que constituyen el núcleo de este paradigma, se recogen en esquemas conceptuales conocidos con el nombre de "ontologías", donde se representa y almacena el conocimiento existente en dicho dominio. Durante los años 1990s y el principio del presente siglo se ha producido el nacimiento y desarrollo de la investigación sobre ontologías (Musen, 2004).

En el campo de la ingeniería del software también se ha planteado recientemente el uso de modelos de conocimiento como las estructuras básicas sobre las que construir las aplicaciones informáticas. El paradigma de las arquitecturas basadas en modelos (Model Driven Architectures, MDA) (Brown, 2004) es el mejor ejemplo de esta aproximación.

Numerosos estudios postulan, y la experiencia práctica demuestra, el progresivo acercamiento y colaboración entre las ideas de ambos campos (Djuric et. al., 2007). La llegada de Internet ha supuesto un escenario en el que esta convergencia se ha visto acelerada y estimulada. La denominada Web Semántica[3] brinda una oportunidad para avanzar en el proceso de interacción entre las dos disciplinas así como para profundizar en el estudio y desarrollo práctico de ontologías.

Esta tesis se centra en la aplicación del paradigma del modelado del conocimiento (de las ideas provenientes del campo de la ingeniería del conocimiento) a la creación de software para la disciplina del control automático, estudiando los aspectos relacionados con la construcción y uso en una aplicación informática de modelos del conocimiento de este dominio expresados en ontologías.

[2] Por *dominio* debe entenderse un ámbito de conocimiento en una determinada materia, incluyendo los conceptos y los mecanismos de razonamiento empleados para resolver los problemas propios de ese ámbito. Ejemplos de dominio pueden ser el diagnóstico de enfermedades infecciosas o el sistema de concesiones de préstamos de un banco.

[3] La idea de Web Semántica fue introducida en 1998 por el propio creador del servicio Web, Tim Berneers-Lee (Berneers-Lee, 1998), como el objetivo final del tratamiento de información en Internet: que las máquinas pudiesen comprender el contenido de los documentos que manejan, descargando así de la tarea a los seres humanos. Ver http://www.w3.org/2001/sw/

Capítulo

1

Justificación, objetivos y estructura de la tesis

Para cualquier cosa que quieras decir sólo hay una palabra para expresarla, sólo un verbo para darle movimiento, sólo un adjetivo para calificarla. Debes buscar esa palabra, ese verbo, ese adjetivo, y nunca contentarte con una aproximación.

Consejo de Gustave Flaubert a Guy de Maupassant

La naturaleza de la investigación a llevar a cabo, así como las aportaciones esperadas en la misma, justifican su realización. El objetivo general de esta tesis es estudiar y comprobar las posibilidades de las técnicas y tecnologías de la ingeniería del conocimiento en la creación de software para ingeniería de control. Se abordará la conceptualización de los diferentes tipos de conocimiento existentes en un subdominio concreto de este campo, construyendo un modelo conceptual del mismo reflejado en una ontología. Para comprobar las posibilidades de esta estructura conceptual se creará una aplicación que permita realizar una interacción (entre el usuario, la aplicación y el conocimiento del dominio) a un nivel que no existe en la actualidad en el software de control.

1.1. Justificación

La ingeniería de control incorpora rápidamente los avances en el campo del software debido a la importancia que éste tiene en todas las disciplinas técnicas y especialmente en ésta. Así ocurrió con los lenguajes de programación, el paradigma de orientación a objetos, las arquitecturas basadas en componentes, los sistemas multiagente, los lenguajes de modelado y simulación, etc. Hoy en día la relación entre software y control es si cabe más estrecha: los sistemas embebidos, la complejidad creciente o la mezcla de dinámicas continuas y simbólicas en los

sistemas son algunas de las causas que hacen que el ingeniero de control deba conocer a fondo el campo del software y los ordenadores (Murray et. al., 2003), (Sanz y Årzen, 2003).

La ingeniería de control también se enfrenta a los problemas y desafíos de la sociedad del conocimiento: si en un principio las aplicaciones informáticas para ingeniería fueron programadas ad-hoc y funcionaban de forma aislada, hoy en día es necesario que todas ellas estén integradas y puedan comunicarse entre sí, para lo cual es necesaria la creación de mecanismos de representación del conocimiento comunes o de formas de automatizar la traducción entre mecanismos diferentes. Además, cada vez se exige un mayor nivel de abstracción en la representación de la información de forma que el ordenador pueda realizar procesos de toma de decisión más complejos, descargando al ser humano de parte de ese trabajo. La utilización del ordenador principalmente como máquina de procesamiento numérico está dando paso al uso del mismo como máquina manipuladora y representadora de símbolos.

Las particularidades que presenta el conocimiento propio de la ingeniería de control hacen que la aplicación de las técnicas de modelado del conocimiento mencionadas en la introducción suponga una oportunidad para la mejora del software en este campo de aplicación (denominado genéricamente CACE – Computer Aided Control Engineering). La relevancia del lenguaje y la semántica en el campo del control automático es primordial y juega un papel más importante que en otras disciplinas técnicas. Mediante el uso de ontologías y tecnologías relacionadas se puede mejorar la capacidad de las aplicaciones informáticas, permitiendo una representación del conocimiento del dominio de más alto nivel, y ofreciendo grandes posibilidades para la reutilización de las estructuras software creadas, lo que permitirá construir aplicaciones modulares fácilmente integrables que, además, lleven a cabo procesamientos de mayor complejidad.

1.1.1. Sobre la construcción de ontologías

La construcción de modelos de conocimiento reflejados en ontologías es un trabajo que exige multidisciplinariedad, al encontrarse la complejidad de este proceso distribuida en dos ámbitos bien localizados: por una parte, el conocimiento necesario para la creación de ontologías está en manos del ingeniero del conocimiento; por otra, el conocimiento que ha de ser modelado (habitualmente de gran complejidad) está en posesión del experto del dominio cuyo conocimiento se pretende modelar. Esta dicotomía ha dado lugar a discusiones sobre en quién debe recaer la mayor parte del trabajo de construcción de ontologías, es decir, si el ingeniero del conocimiento debe adquirir nociones del dominio de aplicación o si el experto en el dominio debe conocer las técnicas de modelado del conocimiento. Este hecho, junto con el carácter de investigación

que todavía caracteriza al campo de las ontologías, hace que el desarrollo de representaciones conceptuales de dominios complejos no sea una tarea sencilla.

Si se observan los proyectos relacionados con la creación de ontologías en dominios complejos se pueden distinguir los siguientes roles o actividades relacionadas con el modelado del conocimiento:

- Desarrolladores de lenguajes para la representación del conocimiento en ontologías.
- Desarrolladores de herramientas para la creación, edición y mantenimiento de ontologías.
- Desarrolladores de ontologías, rol que puede tener varios niveles dependiendo del tipo de conocimiento abordado (ontologías más generales o más específicas).
- Desarrolladores de bases de conocimiento[4].
- Desarrolladores de aplicaciones informáticas (sistemas basados en el conocimiento) que utilizan las estructuras de conocimiento almacenadas en la ontología y en las bases de conocimiento.
- Otros, menos relevantes para esta discusión, como los desarrolladores de algoritmos de razonamiento, métodos de resolución de problemas (PSMs –Problem Solving Methods–), etc.

El primer nivel, los desarrolladores de lenguajes, corresponde a las investigaciones realizadas en el campo de la inteligencia artificial, encaminadas a generar lenguajes que den soporte a los diferentes formalismos de representación que existen. Los estudios realizados en este ámbito están en su mayor parte relacionados con la lógica formal. Los aspectos clave en este campo son los de la expresividad del lenguaje y el balance respecto a su complejidad computacional (o su computabilidad).

El desarrollo de herramientas para la construcción de ontologías es un aspecto crucial, ya que éstas sirven para la creación, edición y mantenimiento de las mismas. Además, son también, en muchos casos, las mismas herramientas que se utilizan para la evaluación y reutilización de estas estructuras. Estas mismas herramientas pueden servir también para que, una vez construida la ontología, ésta sea complementada con la introducción de datos sobre problemas concretos que, junto con la ontología original, formarán una "base de conocimiento". Las herramientas serán muchas veces la interfaz entre las construcciones del lenguaje subyacente y los esquemas de representación del conocimiento, expresados a más alto nivel, que utilizará el constructor de la ontología en primer lugar y el

[4] De forma genérica se puede decir que la ontología es una representación de los conceptos, relaciones y tareas presentes en un determinado dominio, mientras que la base de conocimiento está compuesta por esa representación genérica más información relativa a un problema concreto que pretende abordarse.

encargado de introducir los datos concretos posteriormente. Mientras más facilidades proporcione la herramienta más se aislará al usuario (ya sea experto en ingeniería del conocimiento o en el dominio modelado) de la complejidad del lenguaje subyacente y, por tanto, existirá una mayor comunidad potencial de desarrolladores y usuarios de ontologías.

En el rol de los desarrolladores de ontologías se puede encontrar a investigadores de muy diversos ámbitos dependiendo del tipo de ontología que se pretenda construir. Existen ontologías que modelan conceptos generales presentes en múltiples dominios como el tiempo, el espacio, las relaciones parte-todo, etc., y existen ontologías dedicadas a un dominio concreto de conocimiento como el diagnóstico y tratamiento de enfermedades infecciosas, el comercio electrónico, la descripción de aparatos físicos, etc.

Mediante el término desarrolladores de bases de conocimiento se recoge a los usuarios "primarios" de las ontologías, habitualmente expertos del dominio conceptualizado en esas ontologías, que utilizan sus estructuras para crear una base de conocimiento correspondiente a un problema determinado.

El rol de los desarrolladores de aplicaciones consiste en la realización de aplicaciones informáticas que tengan como elemento central, en tiempo de ejecución, los modelos de conocimiento expresados en la/s ontología/s. Entre estas aplicaciones están aquellas que realizan procesos de razonamiento complejos, similares a los realizados por los seres humanos, sobre la estructura de conocimiento (éstas son aplicaciones herederas de los sistemas expertos que suelen denominarse sistemas basados en el conocimiento). También se encuentran aplicaciones que utilizan la estructura de conocimiento de forma más "tradicional", sustituyendo o complementando a los sistemas basados en bases de datos (e incluyendo, probablemente, alguna funcionalidad extra). En cualquier caso, se necesita que existan facilidades para el acceso a las estructuras de las ontologías y a toda la semántica expresada en las mismas desde la aplicación informática.

A medida que el campo de investigación en ontologías se ha ido aplicando en dominios prácticos reales la evolución del mismo en lo correspondiente a los dos primeros roles (desarrolladores de lenguajes y de herramientas) está siendo influenciada por el tercer rol: el de los desarrolladores de ontologías. Esto quiere decir que los lenguajes para representación de ontologías se desarrollan para que sean lo suficientemente expresivos como para recoger las estructuras de conocimiento que los constructores de ontologías en diversos dominios demandan. La práctica demuestra que sólo a partir de las experiencias concretas se pueden obtener patrones de conceptualización que posteriormente se trasladan a la expresividad del lenguaje. Aparte de las construcciones típicas, presentes en muchos campos y fácilmente identificables como la jerarquía de conceptos

(relación is-a o de subsunción), la herencia de propiedades, ciertas restricciones como la cardinalidad de las propiedades, etc., existen muchas otras estructuras que sólo aparecen al estudiar campos concretos.

En este sentido, las aplicaciones que más influencia han tenido en el desarrollo de lenguajes y herramientas para ontologías son las pertenecientes al ámbito de la biomedicina. En este campo, la representación del conocimiento ha tenido una gran importancia a lo largo del tiempo. Las estructuras de datos utilizadas en este dominio, desde los diccionarios y tesauros de términos hasta las ontologías más grandes que se han desarrollado, presentan una continua evolución. En muchas ocasiones se han creado estructuras de conocimiento y ontologías cuya semántica se implementa, al menos en parte, en el código de las aplicaciones, al no existir suficiente expresividad en el lenguaje de representación para recoger esas estructuras.

La ingeniería ha sido un campo en el que la investigación en representación del conocimiento se ha aplicado también de forma generalizada. En este caso, más que mediante equipos multidisciplinares (como es el caso de la biomedicina), la iniciativa de esta utilización ha sido llevada a cabo por los propios ingenieros. Lo mismo puede decirse del campo de la ingeniería de control, donde la importancia del software en la misma hace que el propio ingeniero sienta como necesidad el conocer las técnicas y herramientas software necesarias (Sanz y Årzen, 2003)

Los hechos mencionados en este apartado permiten concluir que, en el caso del dominio de la ingeniería de control, la investigación y desarrollo en torno a la creación de ontologías para el conocimiento de este dominio puede partir y llevarse a cabo desde el propio campo, al menos en lo concerniente a los roles tercero, cuarto y quinto de los presentados anteriormente[5]. Además, los aspectos presentados en la introducción y en los párrafos anteriores de este apartado permiten concluir que los objetivos de esta tesis, presentados en la siguiente sección, son relevantes tanto para el campo de la ingeniería de control como para el del desarrollo de ontologías.

1.2. Objetivos de la tesis

El objetivo general de esta tesis es estudiar y comprobar las posibilidades de las técnicas de la ingeniería del conocimiento en la creación de modelos de conocimiento y aplicaciones software basadas en esos modelos, para el dominio de la ingeniería de control. La tesis parte de la premisa de que el software en el campo de la ingeniería de control se encuentra en un momento de evolución, similar al resto de las aplicaciones de la informática, y presenta una serie de

[5] Similar conclusión aparece en algunas investigaciones previas llevadas a cabo en los años 1990s (Butz et. al., 1990).

necesidades para adaptarse a las exigencias de la disciplina en la coyuntura social y tecnológica actual. Algunas de estas necesidades pueden ser satisfechas mediante la creación de estructuras de representación del conocimiento con un alto nivel de abstracción que permitan que los ordenadores procesen parte de la semántica del dominio que hasta ahora es manejada de forma exclusiva por el ser humano.

Será necesario identificar y estudiar los diferentes tipos de conocimiento que existen en la disciplina del control automático (en un subdominio o problema acotado de esta disciplina), localizando cómo se han representado previamente estructuras conceptuales similares en otros dominios y/o creando nuevos esquemas cuando no se haya abordado antes la representación de ese tipo de conocimiento. El producto final de este proceso será una ontología que refleje la semántica (el conocimiento existente) del dominio elegido.

El subdominio dentro de la ingeniería de control que se ha elegido para crear el modelo conceptual es el diseño de compensadores de adelanto-retraso de fase con las técnicas del lugar de las raíces. Esta elección se ha tomado debido al gran contenido semántico que presenta el conocimiento de los métodos de análisis y diseño dentro la teoría clásica de control y en particular en el del lugar de las raíces.

La presente tesis aborda la conceptualización del conocimiento necesario para reflejar la estructura estática de conceptos y relaciones existentes en el dominio mencionado. No se realizará una conceptualización del propio proceso de diseño de compensadores, tarea ésta que se está llevando a cabo en otra tesis doctoral. Lo que sí constituye un objetivo es que la ontología resultante sea la base para facilitar el proceso de adquisición de conocimiento para la tarea de diseño de compensadores.

1.2.1. Comprobación de los resultados

El éxito de los objetivos planteados para la presente tesis consistirá en generar una estructura conceptual que recoja el modelo del conocimiento, comprobando que la funcionalidad alcanzada por una aplicación informática utilizando una ontología no puede alcanzarse con las aplicaciones CACE existentes en la actualidad, ni se puede lograr de forma más sencilla con otras tecnologías existentes en el campo del desarrollo de software.

Para comprobar la validez y posibilidades del conocimiento representado en la ontología se construirá una aplicación en la que se muestre cómo se accede a esta estructura de conocimiento y cómo se produce la interacción con el usuario. La aplicación expondrá al usuario el conocimiento existente en la ontología,

ofreciendo definiciones y explicaciones sobre el mismo. Esta aproximación puede ser asimilada a un tutor inteligente en el que sólo se ha implementado el "modelo del dominio" de los que forman la estructura típica de estos sistemas[6] (Psyché et. al., 2003). Existirá también un mecanismo explicativo básico que permita obtener una expresión del conocimiento interno almacenado en relación con cada concepto.

La elección de este tipo de aplicación, orientada a la educación, para comprobar las estructuras de conocimiento creadas se ha basado en la importancia que el campo de la educación tiene en el área de la ingeniería de control y en las posibilidades que las tecnologías investigadas pueden tener en el software educativo para este campo.

1.3. Estructura de la tesis

La estructura de la tesis es la siguiente: en el capítulo 2 se estudian las características y evolución del software para ingeniería de control desde el punto de vista de la representación de la información en el mismo. Se presentan los diferentes ámbitos en los que se ha utilizado el software en este campo y los hitos más relevantes en su evolución, así como los problemas a afrontar en el futuro (todo respecto al punto de vista mencionado). Este capítulo, además de poner en contexto el problema de la representación del conocimiento en el software para control, sirve también para completar el razonamiento sobre la justificación de la aproximación defendida en esta tesis.

La creación de software basada en modelos puede hacerse por medio de diferentes técnicas, unas provenientes de la ingeniería del software y otras de la ingeniería del conocimiento. En esta tesis el estudio se centra en estas últimas ya que son más adecuadas al tipo de conocimiento que pretende representarse (conocimiento de naturaleza humano o conocimiento experto) y al uso que se le pretende dar (la estructura conceptual es la base para la construcción y funcionamiento, en tiempo de ejecución, de las aplicaciones). En el capítulo 3 se presenta el campo de la ingeniería del conocimiento y la evolución que el mismo ha tenido hasta llegar al paradigma del modelado del conocimiento y al concepto de ontología. También se introducen los diferentes formalismos que pueden ser utilizados para la construcción de ontologías, presentando las claves que deben tenerse en cuenta en la elección del mismo. Se dedica un apartado a la comparación de las técnicas descritas con las empleadas en el campo de la ingeniería del software, pretendiendo demostrar el grado de colaboración y mutua influencia que ambas

[6] Un tutor inteligente basado en modelos de conocimiento suele presentar al menos tres de estos modelos: el del dominio, que recoge el conocimiento sobre la disciplina a aprender, el del proceso de aprendizaje, que recoge la secuenciación de temas y relación entre conceptos del modelo del dominio de acuerdo al itinerario de aprendizaje de esa materia, y el modelo del alumno, que recoge lo que el estudiante conoce (y también lo que no conoce) en cada momento.

disciplinas presentan en la actualidad. En este mismo capítulo, después de introducidos estos temas y con la referencia de la evolución presentada en el capítulo 2, se describen las investigaciones y proyectos más relevantes relacionados con la aplicación de las ontologías al dominio de la ingeniería de sistemas y control.

El capítulo 4 presenta, en primer lugar, el subdominio elegido para ser conceptualizado, las razones para su elección, así como una caracterización general del conocimiento existente en el mismo. Posteriormente se muestra cómo se ha abordado la conceptualización de los diferentes conceptos, presentando primero el proceso de decisión sobre el formalismo de representación a utilizar y exponiendo las principales estructuras de conocimiento que forman la ontología desarrollada.

El capítulo 5 describe brevemente las dos aplicaciones desarrolladas en relación a la ontología. La primera de ellas es un procesador de las estructuras semánticas que se han construido pero que no pueden ser recogidas de forma "nativa" por el formalismo de representación elegido (aproximación basada en marcos u objetos estructurados). Esta aplicación es, por tanto, primordial para el funcionamiento de la propia ontología. En segundo lugar se describe la forma en la que se ha desarrollado una aplicación cuyo principal objetivo es mostrar cómo la ontología puede ser utilizada para construir un software en el que la estructura principal es el modelo de conocimiento reflejado en la misma, así como para comprobar la forma en la que las construcciones de la ontología pueden ser presentadas al usuario, ofreciendo una interacción de mayor nivel que la habitualmente encontrada.

Todos los capítulos incluyen como último apartado una breve discusión de los temas tratados y conclusiones obtenidas.

Capítulo

2

El software en ingeniería de control

There is always a danger in using computers because there is a temptation to just plug in numbers until a certain result is obtained without thinking. Recall the words from Hamming[7] *the purpose of computing is insight not numbers* should sit high on any screen.

Karl Johan Åström. Conferencia plenaria en el congreso Advances in Control Education, 2006

El componente software en ingeniería de control es cada vez más importante. De forma general, y desde el punto de vista de la representación de la información en el ordenador, puede decirse que la evolución del software en este campo de la ingeniería ha ido dirigida a conseguir cada vez estructuras de representación con mayor nivel de abstracción como forma de hacer frente al aumento de la información a procesar, la representación de la complejidad, la interacción con el usuario y la integración de aplicaciones. La utilización del paradigma de orientación a objetos está presente en todos los ámbitos de aplicación, observándose que la tendencia, en los últimos años, es el uso de modelos formales de información y conocimiento.

2.1 Introducción

El componente software en la disciplina del control automático es cada vez más significativo en todos los ámbitos de la misma: desde las tareas de modelado, simulación, diseño o implementación "on-line" de controladores hasta la necesidad de nuevas herramientas informáticas que hagan frente a la aparición de

[7] R. W. Hamming. Numerical methods for scientists and engineers. Dover, New York, 1986

nuevos usos del control en ámbitos diferentes a los tradicionalmente conocidos. En un informe de principios de los años 2000s sobre las direcciones de la disciplina del control en el futuro se pone de manifiesto este hecho (Murray et. al., 2003). En este informe se refleja la creciente importancia de la informática en las tareas de la ingeniería de control así como la necesidad de hacer frente a sistemas con mezcla de dinámicas continuas y simbólicas a todos los niveles de aplicación, aunando las técnicas de control con las de la informática y las comunicaciones.

Otro hecho importante, que condiciona las características necesarias del software en control, es la necesidad de integrar todas las aplicaciones informáticas que intervienen en el ciclo de vida de un producto. Así, las aplicaciones de control deben estar integradas con las dedicadas al campo más genérico de la ingeniería de sistemas y, a su vez, deben trabajar conjuntamente con las demás aplicaciones empresariales, de producción y gestión. La necesidad de integración plantea grandes retos debido al gran número y heterogeneidad de aplicaciones y fabricantes existente.

Las nuevas tecnologías de la información y la comunicación suponen también una gran oportunidad en el campo de la educación en control. En este campo, el uso del ordenador como herramienta de aprendizaje ha sido muy limitado, basándose en la ejecución de ejercicios sobre herramientas utilizadas en el diseño de sistemas asistido por ordenador pero sin plantear aplicaciones dedicadas específicamente para el ámbito de la educación. La mejora de las capacidades multimedia de los ordenadores y la expansión de la red Internet han supuesto una oportunidad para empezar a crear aplicaciones dirigidas únicamente al aprendizaje de los conceptos de control, como herramientas con un alto nivel de interactividad o laboratorios remotos en red.

La mejora del software para ingeniería de control se puede abordar desde numerosas vertientes. En los siguientes puntos se realiza una categorización de las aplicaciones dedicadas al campo del control, estudiando cual ha sido su evolución y estado actual desde el punto de vista de la representación de la información empleada en las mismas. Seguidamente se obtienen una serie de conclusiones, haciendo hincapié en la situación actual y en su posible evolución futura en este terreno.

El capítulo está estructurado como sigue: La sección 2.2 está dedicada a estudiar el campo de la ingeniería de control asistida por ordenador desde el punto de vista de la representación del conocimiento en las aplicaciones informáticas de este campo. Después de realizar una caracterización de este tipo de software se muestra la evolución del mismo a lo largo de los años. Finalmente, en la sección 2.3 se presentan las conclusiones que pueden obtenerse de ese estudio previo.

2.2 El software CACE

El uso del ordenador en el campo de la ingeniería de control comenzó prácticamente con la propia disciplina y a lo largo de los años ha encontrado diferentes ámbitos de aplicación. El conjunto de técnicas, procedimientos y herramientas software utilizadas en el campo del control automático se recoge bajo la denominación CACE (Computer-Aided Control Engineering) (Bilqees, 1996). El término abarca el uso del ordenador en todas las actividades de modelado, análisis, diseño, implementación, testeo y documentación de productos y sistemas relacionados con el control automático.

El campo CACE suele dividirse en tres grandes áreas de acuerdo al tipo de uso al que se dedica el ordenador desde el punto de vista del campo del control (Maciejowski, 2006):

- Simulación y modelado.
- Diseño (off-line) de sistemas de control.
- Implementación de controladores.

El software dedicado a las actividades propias de los dos primeros apartados citados suele recogerse bajo el término CACSD[8] (Computer Aided Control System Design).

A los efectos de centrar la categorización de este software de acuerdo al tema tratado en la presente tesis se puede realizar una clasificación atendiendo al tipo de actividad o procesamiento que realiza el ordenador y no al tipo de uso al que se dedica en el campo del control. Así, se tiene:

- Uso del ordenador como herramienta de procesamiento y cálculo numérico. Dentro de esta categoría se incluirían aspectos como el control por computador (la implementación de acciones de control), y aspectos del software CACSD como la simulación (a nivel de algoritmos); también aspectos relacionados con adquisición de datos y el procesamiento digital de señales en la monitorización de sistemas y las herramientas SCADA (Supervisory Control and Data Acquisition), etc. En este tipo de uso la importancia radica en la velocidad de procesamiento de datos para garantizar el funcionamiento en tiempo real.

[8] La caracterización del software utilizado en el campo del control automático ha dado lugar a diferentes clasificaciones y nomenclaturas a lo largo de la historia. En un primer momento se realizó una distinción entre aplicaciones "off-line" y "on-line", separando el software dedicado a la implementación de acciones de control del utilizado para las etapas de diseño no relacionadas con la implementación. De forma más reciente se ha acuñado el término CACSD que ha sido definido como "El uso de computadoras digitales como herramienta primaria durante las fases de modelado, identificación, análisis y diseño de ingeniería de control" (Rimvall y Jobling, 2006). Pero a menudo se usa este nombre para englobar también herramientas y tareas de simulación e incluso implementación de acciones de control. Por otro lado, también el software para educación en control es muchas veces incluido como una aplicación CACSD. El término CACE es el de más reciente aparición y ha venido a englobar todas las tareas involucradas en el campo del software para control, siendo considerado como una especialización del término CAE (Computer Aided Engineering). Ver también http://www.laas.fr/cacsd/scope.html para una discusión sobre el significado y evolución del término CACSD.

- Uso del ordenador como herramienta de representación simbólica. En este caso se incluirían los usos de las herramientas CACSD no relacionadas directamente con el procesamiento numérico sino con la representación de la información, la interacción con el usuario y la comunicación con otras herramientas. También se incluyen en esta categoría las emergentes herramientas en el campo de la educación en control. En este tipo de uso lo importante es la capacidad del software para representar la información de la forma más adecuada y flexible posible, así como la mejora de la interacción hombre-máquina y máquina-máquina.

La diferencia entre los dos usos del ordenador radica en el nivel de representación de la información en el mismo: en el primer caso se manejan datos, en el segundo, información y conocimiento[9]. De esta forma, la evolución de las herramientas, en el primer caso, se produce buscando algoritmos eficientes, estudiando los sistemas operativos en tiempo real, mecanismos para asegurar tiempos de respuesta, etc. En el segundo caso la evolución viene determinada por la mejora en la representación de la información y el conocimiento, y la interacción entre las aplicaciones y entre los usuarios y las aplicaciones. Es este segundo punto de vista el que se pretende estudiar.

De forma general, y atendiendo al ámbito de estudio que se ha mencionado, la evolución de las herramientas CACSD ha estado dirigida por cuatro objetivos principales:

- Mejorar el bajo nivel de representación de los lenguajes de programación, sobre todo en el campo del modelado y la simulación, con el fin de crear herramientas más flexibles que pudiesen facilitar la tarea de introducción y modificación del modelo del sistema.
- Mejorar el nivel de representación de la información datos para facilitar la interacción entre el usuario y las aplicaciones.
- Integrar las diferentes aplicaciones que intervienen en el ciclo de creación de un producto, de forma que diferentes herramientas informáticas puedan trabajar de forma conjunta intercambiando datos de forma transparente.
- Mejorar la presentación de la información en la interfaz con el usuario. Este aspecto se hace patente en el desarrollo de las herramientas de monitorización o las de ayuda a la educación en control.

Seguidamente se ofrece un breve recorrido para comprobar cómo estos cuatro objetivos han influido en la evolución del software en control. En todos los casos

[9] La distinción entre datos, información y conocimiento es a menudo difusa y se presta a interpretaciones. De forma simple puede decirse que los datos serían valores numéricos (2.3; 23.4), la información serían esos valores asociados a alguna magnitud (distancia focal, temperatura), mientras que el conocimiento sería esa información asociada a acciones o toma de decisiones (por ejemplo si esa información sobre distancia focal es adecuada para captar cierta imagen o si esa información sobre temperatura en cierto escenario constituye un problema)

se presentará la evolución de estas herramientas en cuanto a la forma en la que implementan la representación de la información, de modo que se puedan comprobar las diferentes aproximaciones al problema, así como las tendencias actuales. Prácticamente en todos los casos existen experiencias relacionadas con la aplicación de las ideas provenientes del campo de la ingeniería del conocimiento y las ontologías, pero se ha preferido dejar estas exposiciones para la sección 3.5, una vez que se haya introducido el concepto de ontología.

2.2.1 Los lenguajes de modelado y simulación

La evolución del software CACSD en su componente de representación simbólica ha estado marcada en gran medida por los aspectos relacionados con las tareas de modelado y simulación, en definitiva por la necesidad de reducir tiempos y costes en esas tareas con la ayuda del ordenador. Esta evolución puede resumirse con la de los lenguajes de programación dedicados a estos fines. Se puede citar CSSL o ACSL (Michel y Gauthier Associates, 1996) como la herramienta y lenguaje estándar para representar sistemas de ecuaciones diferenciales ordinarias que describen la dinámica de sistemas continuos. La aparición de herramientas como DYMOLA (Elmqvist, 1978), supusieron la posibilidad de representar estas ecuaciones diferenciales sin establecer la causalidad a priori, lo que dio opción a que un mismo modelo pudiese ser usado en problemas diferentes. OMOLA (Mattson et. al., 1993) surgió como un sucesor de DYMOLA en el que se utilizan las técnicas de creación de software orientado a objetos para crear un lenguaje de modelado gráfico y modular. Esto permitió, por primera vez, la reutilización de un modelo de un problema en otro diferente. La posibilidad de crear librerías de objetos y componentes permitió construir herramientas que pudiesen ser utilizadas para diferentes dominios físicos y sistemas en los que se mezclan varios de esos dominios. MODELICA (Mattson y Elmqvist, 1997) es un ejemplo de herramienta que pretende unificar los aspectos de modelado de sistemas que contengan componentes mecánicos, eléctricos, electrónicos, hidráulicos, térmicos, de control, de potencia o relacionados con procesos industriales.

Esta evolución en las herramientas y lenguajes de modelado y simulación dio lugar a la conclusión de que, para manejar la complejidad en un modelo, es imprescindible contar con mecanismos que permitan reutilizar las estructuras de modelado existentes. Para que esto sea posible es necesario, por un lado, contar con un mecanismo de representación de alto nivel (los objetos y componentes en este caso) y, por otro, crear formatos de representación estandarizados a los que todos los desarrolladores se puedan remitir al crear sus modelos.

2.2.2 La interacción en las aplicaciones CACSD

La forma de interacción entre el usuario y las aplicaciones CACSD ha sido otra de las preocupaciones que han guiado la evolución de estas herramientas. El tipo de interacción que se pretenda obtener condiciona la representación interna de los datos en la aplicación informática. Tradicionalmente en el campo CACSD han existido dos aproximaciones enfrentadas: el uso de estructuras y tipos de datos de alto nivel, incluyendo interfaces gráficas de usuario, frente a una interacción basada en la introducción de comandos o en la creación de scripts con estos comandos.

Cuando la aplicación se dedica a un problema o dominio concreto se necesita una interacción con el software de alto nivel, de forma que el usuario no deba ser un experto programando los comandos de una determinada aplicación. Por otro lado, debido al carácter horizontal de la disciplina del control automático, es también necesario que las herramientas presenten la flexibilidad suficiente para que puedan ser extendidas a otros campos de aplicación o puedan ser utilizadas en tareas de investigación. En este segundo caso es más conveniente el uso de comandos de bajo nivel.

La controversia sobre la idoneidad de estas dos aproximaciones deja de ser tal si se tiene en cuenta la horizontalidad mencionada de la disciplina y la existencia de diferentes tipos de usuario (con diferentes perfiles y necesidades) de las herramientas CACSD.

La categorización de los usuarios del software CACSD ha sido tema de debate. En un principio se puede hacer una división entre usuarios expertos e inexpertos, de forma que la interacción basada en comandos estaría dedicada al usuario experto y la de más alto nivel al inexperto. Otra clasificación utilizada es la división entre usuarios de la industria y del ámbito académico, donde los primeros necesitan herramientas de más alto nivel de representación dirigidas a los procesos productivos e integradas con otras aplicaciones, mientras que los segundos tienen necesidades orientadas a experimentar nuevas técnicas y algoritmos, con lo que necesitan una interacción más flexible, de más bajo nivel (lo mismo puede decirse para la realización de prácticas en el campo de la educación). Una clasificación más compleja es la mostrada en (Bilqees, 1996), donde se dividen los usuarios en diseñadores de algoritmos (extienden herramientas), desarrolladores de software (implementan herramientas) y usuarios finales (utilizan herramientas). Todas las divisiones presentadas hacen ver que la existencia de diferentes tipos de interacción con las herramientas CACSD responde a diferentes necesidades de los usuarios y, por lo tanto, todos los tipos de interacción tienen su razón de existir.

En el ámbito de las herramientas basadas en comandos, MATLAB (The Mathworks, 2007) es el ejemplo paradigmático, y se ha convertido en estándar de-

facto en el campo CACSD. La decisión de mantener la interacción basada en comandos y la simplificación de los tipos de datos (todo en MATLAB son matrices de complejos) supuso el gran éxito de esta herramienta, ya que puede ser utilizada para desarrollar software complementario (toolboxes) orientado a determinados campos de aplicación.

En cuanto a la interacción y representación de datos de alto nivel existen numerosos ejemplos, muchos de los cuales se desarrollan como interfaces de usuario basadas en representaciones implementadas mediante el paradigma de orientación a objetos y/o componentes que, en las capas bajas, contienen comandos de MATLAB. El propio software MATLAB incorporó SIMULINK como un software orientado a la representación de los datos a un nivel de abstracción mayor para el campo del control automático. Otros ejemplos de aplicaciones que utilizan representaciones de alto nivel son MATRIXx (National Instruments, 2007a) o LabView (National Instruments, 2007b).

2.2.2.1 Influencia de UML en el software CACSD

El principio de los años 1990s estuvo marcado, en el campo de la ingeniería del software, por la introducción de los aspectos relacionados con el modelado de problemas para el desarrollo de software. El denominado paradigma de la orientación a objetos se vio impulsado por la aparición, a mediados de la década, del lenguaje de modelado UML (Unified Modeling Language) (OMG, 1997) que rápidamente pasó a convertirse en un estándar dentro del ámbito de esta disciplina debido al apoyo y difusión obtenido por parte de la organización OMG (Object Management Group)[10]. La evolución del UML en el campo de la ingeniería del software fue rápida debido a su adopción como estándar y a su uso extendido entre los desarrolladores, dando lugar a una nueva versión (UML 2.0) en 2004. En esta nueva versión se incluyeron nuevas funcionalidades al lenguaje, como la posibilidad de representar estructuras que constan de diferentes componentes o especificar diagramas representando actividades y procesos dinámicos.

El campo de la ingeniería de sistemas comenzó pronto a tener en cuenta el estándar UML al ir cobrando en la disciplina cada vez más importancia los aspectos relacionados con el software. Herramientas como Artisan Studio (Artisan Software, 2007) incorporaron este lenguaje de modelado en sus aplicaciones.

Desde finales de los años 1990s la adopción de UML como estándar de modelado comenzó a traspasar el dominio del desarrollo de software, para el que estaba concebido, y a alcanzar otros campos de aplicación, incluso relacionados con el

[10] La OMG es una organización formada por numerosas empresas privadas que tiene como objetivo el aunar las metodologías y arquitecturas utilizadas en el desarrollo del software, creando especificaciones y estándares que sean aplicados por toda la comunidad.

hardware. Las nuevas características de UML 2.0 hicieron que éste lenguaje se viese como una oportunidad también en el campo de la ingeniería de sistemas. En marzo de 2003 la OMG lanzó una iniciativa para desarrollar un lenguaje específico para este campo, tomando como base la versión 2 del UML. Este estándar es SysML (Systems Modeling Language) (OMG, 2006) (Willard, 2007). SysML introduce nuevos diagramas y estructuras, eliminando alguna de las construcciones de UML y siendo por tanto una extensión de UML 2.0 y no sólo una aplicación del mismo[11]. A pesar de su reciente aparición, SysML ya ha sido propuesto en el campo del control automático como un lenguaje de modelado de alto nivel sobre herramientas como MATLAB (Mueller et. al., 2006) (Vanderperren y Dehaene, 2006).

2.2.3 La integración de aplicaciones

Uno de los hechos que más ha influido en la evolución de la representación de la información en el software para ingeniería de control (y para ingeniería de sistemas en general) es la necesidad de integración de aplicaciones.

La integración de las herramientas informáticas participantes en el ciclo de vida de un producto es una preocupación que surgió de la necesidad de automatizar la comunicación entre todas las aplicaciones dedicadas a diferentes etapas de la producción. Al introducirse la informatización en todos los ámbitos de la producción el paso de los datos de una aplicación a otra empieza a ser problemático, ya que muchas veces supone que los ingenieros expertos en esas aplicaciones y procesos tengan que comunicarse y adaptar los formatos de esos datos para que puedan ser introducidos en la siguiente herramienta. Hoy en día esta preocupación va más allá todavía, y se pretende que la integración de herramientas llegue al máximo, abarcando también las aplicaciones no relacionadas directamente con los procesos de fabricación (aplicaciones de gestión, económicas, de relación con el cliente, etc.).

Los primeros intentos de crear plataformas software de integración de aplicaciones en los campos de la ingeniería de sistemas y control datan de mediados de los años 1980s (más allá de la traducción de datos a bajo nivel para las herramientas de simulación). Ejemplos de estas plataformas fueron ECSTASY (Environment for Control System Theory and Synthesis) (Munro, 1990), GE-MEAD (General Electric - Multidisciplinary Expert-aided Analysis and Design) (Taylor y Frederick, 1984) o ANDECS (Grübel, 1995). Estos tres sistemas están basados en la existencia de un repositorio central donde los datos utilizados en el proceso de diseño de un sistema son almacenados utilizando un formato estándar.

[11] El diagrama más importante que se ha introducido es el de requerimientos mientras que, desde el punto de vista del lenguaje, es relevante mencionar la casi nula utilización del diagrama de clases y la importancia de la representación de la información estructural.

Existe un mecanismo de traducción entre los datos de una aplicación informática de un cierto fabricante y este formato estándar, de forma que se logra la interoperabilidad e integración deseada. El repositorio central suele consistir en una base de datos. Además, en ANDECS también existe un completo marco de referencia que establece un modelo de objetos dedicado a la representación de la información en el campo del control automático (Grübel, 1994) (el modelo de datos se guarda en una base de datos orientada a objetos).

Las experiencias con estas herramientas mostraron que la utilización de un repositorio central común para la información no era suficiente para asegurar la integración de aplicaciones de diferentes fabricantes (Varsamidis et. al., 1996). El mayor problema de estas aproximaciones radica en que el mecanismo de traducción a una representación común de los datos no es viable, al no poder hacer frente a la multitud de fabricantes de software que pueden cambiar sus modelos particulares de datos de una versión a otra. Esto dio lugar a que, durante la década de los 1990s, los esfuerzos se centraran en buscar estándares para la representación de datos a los que todos los fabricantes pudiesen adherirse y arquitecturas de integración basadas en entornos abiertos que permitiesen la implementación e integración sencilla de aplicaciones heterogéneas.

En (Barker, 1994) se introduce el uso de arquitecturas abiertas para definir el modo de integración de las herramientas CACSD basándose en ideas similares provenientes del campo de la Ingeniería del Software, en concreto del campo del CASE (Computer-Aided Software Engineering). Estas arquitecturas definen un modelo en capas donde cada capa implementa una serie de servicios (interfaz de usuario, gestión de tareas, modelado y almacenamiento en base de datos). Las diferentes herramientas se integran en esa arquitectura y un servicio de mensajes permite la comunicación entre las diferentes capas.

Desde el punto de vista de la representación de la información, esta aproximación utiliza la orientación a objetos incluso en el ámbito del almacenamiento permanente, donde se prioriza la utilización de bases de datos orientadas a objetos partiendo de la idea de que una base de datos relacional no posee la riqueza suficiente para representar las estructuras de información presentes en la ingeniería de control (Joos y Otter, 1991)

Durante los años 1990s se buscan nuevas vías para expresar la riqueza de las estructuras de datos más allá de lo que ofrecen las técnicas de orientación a objetos. Un ejemplo es el lenguaje EXPRESS (ISO, 1992), desarrollado por la ISO (Internacional Organization for Standarization) dentro del estándar ISO 10303 (Trapp, 1993) (conocido también como STEP – Standard for the Exchange of Product data model –) e introducido en el campo CACSD como un lenguaje de representación común de alto nivel para facilitar la integración de aplicaciones (Varsamidis et. al., 1994). También se realizaron experiencias utilizando

EXPRESS como una capa de abstracción sobre MATLAB (Varsamidis et. al., 1999).

La evolución del estándar ISO 10303, inicialmente dirigido a representar información relacionada con la ingeniería mecánica, ha dado lugar recientemente (años 2000s) a la definición de un protocolo de aplicación[12] orientado al intercambio de datos en el campo de la ingeniería de sistemas: el AP-233 "Systems Engineering Data Representation" (Herzog y Törne, 2001). Este protocolo pretende describir las estructuras de datos genéricas (independientes del ámbito en particular) que se encuentran en el proceso de creación de un producto. La publicación como estándar internacional está prevista para enero de 2008. AP-233 utiliza EXPRESS como lenguaje para especificar los modelos debido al uso extensivo que este lenguaje tiene en el desarrollo del estándar ISO 10303, aunque también se estudian otros lenguajes, como UML, para este cometido (Lubell et. al., 2004).

Como puede comprobarse, desde finales de los años 1990s la representación de datos parte de la creación de estándares creados por organizaciones internacionales con la ayuda de diferentes instituciones privadas, educativas, profesionales, etc., en vez de provenir de ideas generadas en grupos aislados en instituciones académicas. En el campo de la ingeniería de sistemas la representación de información tiene como principal herramienta al lenguaje SysML, mientras que para la comunicación de datos el protocolo AP233 está en pleno proceso de desarrollo. Ambos pretenden representar básicamente el mismo tipo de información, presentando cada aproximación puntos fuertes y débiles (Herzog, 2005). La convergencia de ambas aproximaciones en un modelo común de representación/intercambio de datos sería técnicamente viable pero es difícilmente alcanzable por razones de política de las organizaciones que los promueven.

2.2.4 El software para educación en control

La educación en el campo del control es una preocupación a la que se da gran importancia en esta área de la ingeniería, baste citar que las dos organizaciones internacionales más importantes (la IEEE CSS - IEEE Control Systems Society - y la IFAC - Internacional Federation of Automatic Control -) tienen comités dedicados exclusivamente al campo de la educación. Existen congresos dedicados a este asunto en los que cada vez se produce más participación y aportaciones (por ejemplo el simposio IFAC ACE - Advances in Control Education – celebrado con

[12] Los protocolos de aplicación (AP – Application Protocols –) son módulos que se desarrollan para permitir la representación de la información de un determinado campo de aplicación. La mayoría de los estándares y protocolos de aplicación de STEP se dedican a la ingeniería mecánica y a la representación de artefactos de natruraleza mecánica.

carácter trienal a partir del año 1988 o el IFAC Workshop on Internet Based Control Education -IBCE-, cuya primera reunión se produjo en 2001).

Los aspectos sociales y tecnológicos de la educación en control se han puesto de manifiesto en diversos estudios sobre el tema (Dormido, 2002), (Antsaklis et. al., 1999), donde se introducen aspectos como la necesidad de implantar el aprendizaje a lo largo de la vida en esta materia o la necesidad de aprovechar la oportunidad que las Tecnologías de la Información y la Comunicación (TICs) suponen para este campo.

Hoy en día los libros de texto más utilizado en educación en control suelen incluir en cada capítulo una serie de ejercicios desarrollados en MATLAB (y/o SIMULINK) como apoyo para afianzar los contenidos teóricos. Sin embargo, la integración entre los libros de texto y las herramientas CACSD no se ha producido en absoluto hasta la fecha, tal como se refleja en las conclusiones de la mesa de trabajo **"**Rethinking Control Education in The Modern World" del 7º simposio en Avances en Educación en Control (IFAC ACE 2006).

La preocupación por herramientas software específicamente pensadas para educación en el campo del control es un hecho relativamente reciente. A partir de finales del siglo XX han ido apareciendo diversas aplicaciones, como (Johanson et. al., 1998) o (Guzmán et. al., 2006), que hacen uso de las técnicas de orientación a objetos, capacidades multimedia y las comunicaciones a través de Internet para crear aplicaciones con gran interactividad y uso extensivo de representaciones gráficas así como, más recientemente, para construir laboratorios virtuales y/o remotos a través de Internet (Reguera et. al., 2004), (García et. al., 2006). Estas aplicaciones han supuesto un gran avance comparadas con las existentes hasta el momento, de las que usualmente se obtenían resultados numéricos sin ningún tipo de interactividad.

La interacción del ser humano con el ordenador es un elemento crucial en este tipo de aplicaciones, ya que la educación es un proceso de transmisión de conocimiento. Por esta razón, además de la importancia de la introducción de datos e información en el ordenador, es capital la presentación de información por parte del ordenador al usuario. Uno de los principales problemas que las modernas herramientas mencionadas afrontan es la separación entre la información presentada al usuario y los conceptos teóricos subyacentes. Las interfaces de usuario, aunque muy avanzadas en cuanto al empleo de capacidades multimedia, siguen dejando la interpretación de los datos al usuario de forma que éste puede perderse en lo profuso de las gráficas y datos que se le presentan y no establecer las relaciones con los conceptos teóricos importantes. Las aplicaciones no cuentan con modelos que recojan el conocimiento del dominio ni el conocimiento del proceso de aprendizaje de la materia. De esta forma, no es posible ofrecer contenidos conceptuales relacionados con los datos presentados, a

no ser de forma totalmente separada. Tampoco es posible, por tanto, utilizar estas herramientas como ayuda para localizar los errores del alumno y guiarlo de forma individual en su proceso de aprendizaje.

2.3 Discusión y conclusiones parciales

De forma general, y desde el punto de vista de la representación de la información en el ordenador, puede decirse que la evolución del software en ingeniería de control, y en el campo más amplio de la ingeniería de sistemas, está caracterizada por la búsqueda de estructuras de representación con mayor nivel de abstracción como forma de hacer frente al aumento de la información a procesar. La utilización del paradigma de orientación a objetos está presente en la actualidad en todos los ámbitos de aplicación, observándose que la tendencia en los últimos años es el uso de modelos formales de información y conocimiento.

La evolución del software CACE ha estado guiada por la necesidad de reutilizar las estructuras de código, representar la complejidad, ofrecer una mejor interacción hombre-máquina e integrar las aplicaciones participantes en los procesos de ingeniería.

Los lenguajes de modelado y simulación basados en objetos y componentes han logrado que la reutilización de código sea una realidad, consiguiendo que la construcción del modelo de un sistema complejo sea una tarea más sencilla y además que ese mismo modelo (o una de sus partes) pueda ser utilizado en otros problemas.

La mejora en la interacción hombre-máquina ha venido también del uso de técnicas de la ingeniería del software adaptadas al campo del control. En este aspecto es muy interesante fijarse en la evolución de la aplicación del lenguaje UML en el campo de la ingeniería de sistemas. Como se ha visto, UML pasó de ser un lenguaje para crear modelos de aplicaciones software a ser un lenguaje para crear un modelo de un sistema que mezcle tanto componentes software como hardware, siendo SysML la evolución final de este proceso.

La gran ventaja de las aproximaciones basadas en UML/UML2.0/SysML radica en que, debajo de los diagramas utilizados, hay modelos formales que sirven para comprobar la integridad de los datos, así como automatizar la creación ya sea de software o hardware, mejorar la legibilidad e interpretación de los modelos o permitir la traducción automática de los datos a otros lenguajes para facilitar la comunicación e integración con otras aplicaciones.

La estandarización y la creación de modelos de información basados en arquitecturas abiertas son las soluciones encontradas para lograr la integración de

aplicaciones. Dentro del ámbito de la creación de modelos de información para ingeniería se puede comprobar cómo, en un principio, la iniciativa de generación de estos modelos partía del ámbito de la investigación dentro de instituciones académicas mientras que hoy en día parte de iniciativas de grupos de organizaciones y empresas que buscan crear un estándar. La necesidad de integración real de productos ha hecho que las empresas tomen conciencia de la necesidad de modelos comúnmente aceptados, única forma de que la integración se produzca a niveles industriales. Además, dentro de los estudios relacionados con esta problemática, se comprobó también que la representación de la información existente en el campo de la ingeniería de control requería de estructuras complejas basadas en la orientación a objetos y más allá de estas.

En cuanto a la educación en control, es un hecho admitido que las nuevas tecnologías de la información y las comunicaciones permiten una aplicación directa de los ordenadores en este campo. Aunque el ordenador se ha utilizado siempre como una herramienta importante en la ingeniería de control sólo cuando éste ha tenido la capacidad para ofrecer una interacción gráfica compleja con el usuario se han desarrollado aplicaciones específicas para el ámbito de la educación. Sin embargo existen dudas sobre cual es la mejor forma de utilizar estas nuevas tecnologías. Parece claro que el utilizar el ordenador y las redes de comunicación para implementar el mismo tipo de enseñanza que se imparte en una clase presencial no es lo adecuado (Dormido, 2004). Los laboratorios virtuales en Internet son un ejemplo de aplicación novedosa. La interactividad es otro de los puntos fuertes de las herramientas implementadas hasta el momento, pero presentan el problema de perder el vínculo con los conceptos teóricos subyacentes.

De acuerdo a todas estas consideraciones puede concluirse que las técnicas y herramientas de la ingeniería del conocimiento y las ontologías son unos elementos con grandes posibilidades de aplicación dentro del dominio. Las ontologías son, por definición, estructuras conceptuales que recogen el modelo de conocimiento de un dominio. Son por tanto estructuras creadas para representar conocimiento y tienen cierto paralelismo con los objetivos perseguidos con el uso de EXPRESS, UML o SysML pero presentan, como se verá, mejores características que éstas herramientas y lenguajes.
Tal como se ha comprobado, la complejidad del conocimiento de ingeniería de control demanda estrategias de modelado del conocimiento más potentes para construir aplicaciones CACE, sea cual sea el ámbito de aplicación. La ingeniería de conocimiento y las ontologías son una oportunidad para avanzar y hacer frente a la creciente informatización del dominio a todos los niveles.

Quizás en el ámbito en el que mayor aplicación puede tener las ontologías es en el de la educación en control con ayuda del ordenador. Los modelos de conocimiento del dominio, expresados en ontologías, permitirán que los

elementos presentados al usuario en la interfaz de las aplicaciones educativas no pierdan la relación con los conceptos teóricos. Con el software educativo basado en ontologías el ordenador puede ser una herramienta importante para implementar la educación en conceptos de control desde una perspectiva más cualitativa que cuantitativa, aspecto relevante a la hora de lograr una buena comprensión de los temas (Bissell, 1999). En definitiva, se pasaría de intentar integrar las aplicaciones CACSD en los libros a "incluir los libros" en la aplicación informática. Y es más, esta unión no sería mediante enlaces a transcripciones digitalizadas de textos teóricos sino que la propia teoría estará incluida en el modelo conceptual.

Capítulo

3

Representación del conocimiento y ontologías

- ... Trabajé un poco en este proyecto, para ser exacto.
- ¿Qué tipo de trabajo?
- Oh, asuntos de P.I. en su mayoría – dijo Hackworth. Asumió que Finkle-McGraw se mantenía al tanto y reconocería la abreviatura de pseudo-inteligencia, y quizás incluso apreciase que Hackworth hubiese hecho esa suposición.
Finkle-McGraw se iluminó un poco.
- Sabe, cuando era joven lo llamaban I.A., Inteligencia Artificial.
Hackworth se permitió una leve y fina sonrisa.
- Bueno, supongo que podríamos hablar de caraduras.

Neal Stephenson – La era del diamante: Manual ilustrado para jovencitas

---

En el presente capítulo se hace un recorrido por la representación del conocimiento, término nacido en el campo de la inteligencia artificial. El objetivo de este recorrido es conocer las raíces y la evolución del paradigma del modelado del conocimiento, poniendo en contexto el concepto de ontología y presentando la interacción de estas técnicas con las existentes en el campo de la ingeniería del software. También se presentan los problemas existentes en la formalización de una ontología con el objetivo de implementarla y utilizarla en una aplicación informática y, finalmente, se repasan las investigaciones sobre la aplicación de estas estructuras conceptuales en el campo de la ingeniería de sistemas y control.

3.1 Introducción

Desde el momento de la invención del ordenador las posibilidades de aplicación del mismo se agruparon en dos categorías: aquellas que aprovechaban la

capacidad de esta máquina para realizar cálculos numéricos complejos con extremada rapidez y aquellas que, apoyándose en el ordenador como máquina capaz de manejar símbolos, perseguía emular los procesos de razonamiento del cerebro humano. Esta segunda tendencia es la que dio lugar a la disciplina de la inteligencia artificial, mientras que la primera puede considerarse que originó la rama del "desarrollo del software tradicional".

Los primeros estudios sobre los problemas de representación del conocimiento[13] datan de los años 1970s, alcanzando una gran importancia dentro del campo de la inteligencia artificial y siendo el punto de partida en la creación de una importante sub-disciplina dentro de ésta: la ingeniería del conocimiento[14] (Feigenbaum, 1980).

La evolución de la ingeniería del conocimiento puede dividirse en dos etapas: la primera de ellas, que abarca los años 1980s y el principio de la década de los 1990s estuvo caracterizada por las aplicaciones denominadas "sistemas expertos", aplicaciones construidas mediante el paradigma de la "extracción del conocimiento" de la mente del experto por medio de diversas técnicas. A partir de finales de los 1980s se produjo una crisis en estos sistemas expertos, cobrando mayor relevancia los estudios teóricos sobre las realizaciones prácticas. Surge entonces, a partir de mediados de los años 1990s, la segunda etapa de la ingeniería del conocimiento, basada en el paradigma del "modelado del conocimiento" que, en oposición a la etapa de la extracción del conocimiento, trata el problema de forma global, tratando de generar un modelo conceptual del mismo (Palma et. al., 2000). La explicitación de este modelo conceptual en una estructura concreta recibe el nombre de ontología (Gruber, 1993).

Una ontología es, pues, una estructura conceptual que almacena la representación del conocimiento de un dominio. La ontología existe de forma independiente de cualquier implementación, se dice que está expresada en el denominado "nivel del conocimiento" (Newell, 1982), sin importar el agente (humano o no) que la va a utilizar. Para utilizar la ontología en un ordenador (es decir, para expresarla en el nivel simbólico y de código) es necesario explicitarla mediante algún formalismo[15] o lenguaje. La elección de este formalismo es un punto crucial en el

[13] Todo software conlleva una representación del conocimiento más o menos explícita, por lo que el término no sería privativo del campo de la inteligencia artificial. Sin embargo, aquí se empleará la acepción acuñada dentro de esta disciplina, que es donde con más profundidad se ha tratado el tema.

[14] El término ingeniería del conocimiento se creó para agrupar todos los aspectos relacionados con el desarrollo de sistemas expertos, en este sentido abarca los estudios en representación del conocimiento pero también las técnicas utilizadas en el diseño, creación y evolución de esos sistemas.

[15] El término "formalismo" o "formalización" hace referencia a la expresión del conocimiento en una forma tal que sea directamente implementable en un ordenador. Suele hablarse también de formalismos más o menos "formales", lo cual crea cierta confusión. Este hecho se produce porque las palabras "formal" y "formalización" se utilizan con dos significados diferentes. En el proceso de creación de sistemas expertos, formalización es el proceso de especificar el conocimiento en una estructura manejable por un ordenador. Desde el punto de vista de la representación del conocimiento, el grado de formalización (de un formalismo) hace referencia al empleo de un lenguaje basado en mecanismos semánticos

desarrollo de una aplicación basada en ontologías, ya que de esa elección depende la capacidad de representar en el ordenador todas las estructuras existentes en la ontología, así como la posibilidad de que el esquema resultante sea computable y tenga una complejidad computacional abordable[16].

El paradigma del modelado del conocimiento es aplicable en la realización de cualquier tipo de software, aunque es más eficiente en aquellos problemas en los que interviene conocimiento humano difícilmente explicitable (y posiblemente incompleto, por tanto) o en dominios complejos cuya estructura pretende utilizarse en toda su complejidad durante la ejecución de la aplicación.

Las ideas de la ingeniería del conocimiento están ganando terreno en el campo de la creación de aplicaciones prácticas. La interacción entre ingeniería del conocimiento e ingeniería del software es hoy en día elevada, siendo el paradigma del modelado del conocimiento el punto de unión de ambas. La madurez de la ingeniería del conocimiento se puede comprobar en el hecho de que algunos lenguajes y técnicas relacionadas con las ontologías están siendo objeto de procesos de estandarización por parte de organismos como la OMG o el World Wide Web Consortium (W3C)[17]. En este último caso, el lenguaje OWL (Ontology Web Language), de representación de conocimiento en ontologías, es el que se está utilizando como base para la construcción de lo que se postula como el nuevo servicio Web: la denominada Web Semántica[18].

El capítulo está organizado como sigue: El apartado 3.2 está dedicado a la historia de la representación del conocimiento, centrándose sólo en los aspectos más relevantes mediante los que se pueden presentar las ideas que dieron lugar a la situación actual de la disciplina y ofrecer un contexto para introducir el concepto de ontología. En el apartado 3.3, una vez definido el concepto de ontología, se

automatizables. En este sentido la mayor formalidad sería la conseguida mediante lenguajes cuya semántica esté basada en la lógica y la teoría de modelos formales de Tarsky.

16 Por computabilidad se entiende el estudio de si un determinado problema es resoluble utilizando un determinado modelo de computación. El término está relacionado con la decidibilidad (se estudia si el ordenador puede hacer frente a un determinado problema mediante cierto algoritmo y resolverlo en un tiempo finito). La complejidad computacional, por su parte, hace referencia a la eficiencia en la resolución de un problema mediante un determinado algoritmo, estando relacionada con el concepto de tratabilidad (se estudia si se puede asegurar que el ordenador ofrecerá una respuesta en un tiempo finito y asumible - tiempo polinomial - o no -tiempo exponencial -). Puede haber esquemas de representación cuyos algoritmos sean indecidibles, decidibles pero intratables o decidibles y también tratables. La expresividad del esquema de representación se ve reducida en cada uno de esos casos. Cabe mencionar también que los estudios sobre decidibilidad y tratabilidad se pueden aplicar a aquellos problemas que se pueden expresar como un problema de decisión. En el caso de los lenguajes de representación del conocimiento estos estudios se pueden realizar cuando el lenguaje utilizado está basado en una teoría lógica. Se dice que una teoría lógica es decidible si el conjunto de todas las fórmulas bien formadas válidas en el sistema es decidible, es decir, si existe un algoritmo tal que para cada fórmula en el sistema el algoritmo es capaz de decidir en un número finito de pasos si la fórmula es válida en el sistema o no. El problema de decisión es pues el de la validez o no de una fórmula. A modo de ejemplo, la lógica de proposiciones es decidible, la de predicados (ó lógica de primer orden) sólo lo es si se limita a predicados monádicos.

17 El World Wide Web Consortium "desarrolla tecnologías inter-operativas (especificaciones, líneas maestras, software y herramientas) para guiar la Red a su potencialidad máxima. El W3C es un foro de información, comercio, comunicación y conocimiento colectivo" http://www.w3.org

18 http://www.w3.org/TR/owl-features/

prestará atención a la evolución del mismo desde su aparición, a principios del siglo XX, hasta la actualidad, exponiendo las ideas principales, así como los formalismos mediante los que se representan estas estructuras conceptuales en un ordenador. Todas estas exposiciones son importantes para la elección de la aproximación a seguir al construir la ontología para el dominio de control automático. Aunque la disciplina de la representación del conocimiento tiene sus raíces en la inteligencia artificial, los problemas abordados por la misma aparecen también en el proceso de programación de cualquier aplicación informática. Hoy en día existe cada vez una preocupación mayor por este tema tal como se muestra en el apartado 3.4, dedicado a presentar las similitudes entre las aproximaciones basadas en el modelado conceptual provenientes del campo de la ingeniería del software y las del campo de la ingeniería del conocimiento. Este apartado pretende mostrar cómo el paradigma del modelado del conocimiento está ganando aceptación en todos los ámbitos de la construcción del software y que ambas disciplinas se encuentran en un proceso de intercambio de ideas, pudiéndose incluso hablar de convergencia. El capítulo finaliza, en el apartado 3.5, con un repaso a los trabajos e investigaciones en los que las técnicas de la ingeniería del conocimiento y las ontologías se han aplicado a dominios relacionados con la ingeniería de sistemas. Este punto es un complemento a los temas abordados en el capítulo 2. Finalmente, el punto 3.6 está dedicado a la discusión y conclusiones parciales.

3.2 Evolución de la representación del conocimiento

3.2.1 El origen de la representación del conocimiento.

La inteligencia artificial (IA) empezó su andadura en los años 1950s buscando mecanismos de razonamiento generales que emulasen a los del cerebro humano y que pudiesen ser utilizados para resolver cualquier tipo de problema de los habitualmente resueltos por las personas. El desarrollo más representativo de esta aproximación fue el denominado "Solucionador General de Problemas" (General Problem Solver) (Newell et. al., 1959). Sin embargo este objetivo es abandonado pronto ante los resultados obtenidos en las primeras experiencias realizadas, donde se ve que mecanismos genéricos de razonamiento sólo son capaces de resolver problemas matemáticos (por tanto ya muy formalizados desde su planteamiento) sencillos y que, además, las técnicas utilizadas no escalan bien, es decir, que sólo podían ser utilizadas para resolver problemas "de juguete", mientras que dejaban de ser válidas cuando el problema aumentaba de tamaño o consistía en una aplicación real.

En los años 1960s y 1970s se plantea que lo realmente importante para que un ordenador pueda manejar y resolver problemas habitualmente solucionados por seres humanos es tener tanta información como sea posible sobre el dominio específico del problema y no confiar en mecanismos generales de razonamiento que puedan ser aplicados de forma universal. De esta forma se produce un doble cambio en la orientación de la investigación en inteligencia artificial: por un lado se pasa de buscar soluciones para cualquier problema a restringir mucho el dominio en el cual la aplicación de IA funcionará como tal, por otro lado se pasa de plantear la búsqueda de mecanismos generales y universales de razonamiento a la idea de que lo realmente importante es que el sistema tenga tanto conocimiento interno como sea posible acerca del dominio de aplicación.

De este modo surge el problema de la "representación del conocimiento", es decir, cómo llevar el conocimiento (almacenado en una mente humana) de un determinado dominio a un ordenador. Uno de los primeros proyectos donde se pone de manifiesto este problema y sus posibles soluciones es SHRDLU (Winograd, 1972), nombre de un robot virtual que debía manipular bloques de diferente tipo y color a partir de las indicaciones que un usuario introducía en el sistema en forma de sentencias en lenguaje natural controlado[19]. En este proyecto la representación del conocimiento se hacía mediante una base de datos que contenía los diferentes bloques, sus colores y las posiciones relativas de los mismos, es decir, definiendo un "micromundo" que constituía el dominio de conocimiento a manejar.

3.2.2 Del paradigma de la extracción al del modelado del conocimiento.

A partir del planteamiento y estudio del problema de la representación del conocimiento proliferan, al final de los años 1970s y durante los 1980s, los denominados “sistemas expertos”: aplicaciones software capaces de solucionar problemas dentro de un dominio reducido emulando los métodos utilizados por personas expertas (de ahí su nombre) en ese dominio. Algunos de estos sistemas expertos tuvieron bastante éxito e incluso fueron utilizados por empresas generando sustanciosos beneficios (en (McDermott, 1982) se describe el caso más paradigmático de éxito en el aspecto económico: el sistema R1/XCON).

Los sistemas expertos contaban con una representación explícita del conocimiento del dominio del problema, constando ésta habitualmente de varios módulos más o menos independientes entre sí:

[19] Este trabajo contiene el germen muchas de las técnicas que en los años 80 serían utilizadas en la programación de los denominados “sistemas expertos” y es también uno de los primeros en tratar el problema del procesamiento del lenguaje natural en los ordenadores.

- Una base de hechos, que contiene los datos o información sobre la que trabaja el sistema.
- Una base de reglas, formada por reglas del tipo SI…ENTONCES que capturan el conocimiento acerca de la resolución del problema llevada a cabo por el experto.
- Un mecanismo de inferencia, que se encarga de hacer que las reglas se ejecuten y lleven al programa desde su punto de partida a la solución.

A finales de los 1980s y principios de los 1990s la investigación y desarrollo de sistemas expertos cayó en crisis debido a los problemas inherentes a la forma en la que estaban concebidos y construidos y que puede resumirse en los siguientes factores:

- Conocimiento disperso. A pesar de que existe una representación explícita del conocimiento, éste se encuentra disperso a lo largo de una serie de hechos sin organización y reglas que, en algún caso, llegan a un número de decenas de miles.
- Mantenimiento imposible. La dispersión del conocimiento hace que, a la hora de modificar éste, sea casi imposible saber cuántas reglas habría que cambiar o qué implicaciones tendría el cambio de cierta regla sobre otras.
- Construcción incremental. El paradigma para la construcción de los sistemas expertos es la "extracción del conocimiento": Mediante entrevistas al experto, técnicas de análisis de protocolos, etc. se iban obteniendo reglas que expresaban su conocimiento, pero estas reglas se iban introduciendo en el sistema según se conseguían, sin tener en cuenta la estrategia de resolución de problemas a un mayor nivel de abstracción.
- Construcción artesana. La característica que mejor define a la construcción de un sistema experto es la de "artesanía" o, si se quiere, la consideración de este proceso como "un arte", lo que supone la inexistencia de estructuras de conocimiento reutilizables y la poca definición en las metodologías de construcción, verificación y validación.

De esta crisis de los sistemas expertos y debido a las características de los mismos se toma conciencia de la importancia de un segundo problema (aparte del de la representación del conocimiento): el problema de la "adquisición del conocimiento", expresado por primera vez por Feigenbaum en (Feigenbaum, 1977) [20]. Este término hace referencia al proceso por el cual se introduce nuevo conocimiento en el sistema y su aparición venía a poner de manifiesto el problema de la creación y mantenimiento de los sistemas expertos. De hecho, se llegó a

[20] En realidad Feigenbaum introduce tres términos, dos de ellos de su propio cuño: "representación del conocimiento" (ya conocido) que hace referencia a "las actividades involucradas con las bases de conocimiento", "utilización del conocimiento", que hace referencia a "los procesos de razonamiento que usan el conocimiento para resolver problemas" y la "adquisición del conocimiento", que hace referencia a "las actividades involucradas en introducir el conocimiento en las bases de conocimiento".

acuñar el término "cuello de botella de la adquisición del conocimiento" para hacer referencia a la dificultad de este proceso.

El estudio de los problemas de la representación y adquisición del conocimiento son la base para el desarrollo de nuevas investigaciones en torno a los sistemas expertos y para la aparición, como consecuencia de ese proceso, del nuevo paradigma del modelado del conocimiento frente al de la extracción, que dará lugar a utilizar el término más genérico de "sistemas basados en el conocimiento" frente al de "sistemas expertos". El término "adquisición", referido al conocimiento, se sustituye por "modelado". La componente de arte, asociada a la disciplina, pretende sustituirse por un proceso de ingeniería verdadera. La ingeniería del conocimiento surge, entre los años 1980s y 1990s, como un amplio campo de estudio que pretende la creación de artefactos software utilizando las técnicas de la inteligencia artificial. En (Feigenbaum, 1992) Feigenbaum ofrece una visión personal de la era de los sistemas expertos justo en el momento de cambio al paradigma del modelado.

El problema de la representación del conocimiento, bajo la nueva concepción, no recoge sólo la necesidad de introducir el conocimiento de los expertos en las aplicaciones informáticas, sino que incluye cómo debe hacerse este proceso y la búsqueda de mecanismos formales que permitan realizarlo. Además, la estructura que refleje el conocimiento explícito del dominio tratado debe tener la suficiente capacidad como para recoger la complejidad de ese conocimiento.

Dentro de las ideas del nuevo paradigma del modelado se delimita y redefine el concepto "representación del conocimiento" que, de acuerdo a (Randall et. al., 1993), es a la vez:

- Un sucedáneo, una serie de símbolos que representan objetos y relaciones existentes en el mundo exterior.
- Una serie de acuerdos ontológicos (el término ontológico se utiliza aquí en el sentido filosófico) sobre lo que existe, y su significado dentro de un determinado dominio de aplicación.
- Una teoría fragmentaria de razonamiento inteligente. El mero hecho de utilizar una cierta representación del conocimiento conlleva suponer un cierto mecanismo que produce el comportamiento inteligente.
- Un medio para una computación más eficiente, al codificar conocimiento de forma explícita en el ordenador (en contraposición a la programación denominada "procedural").
- Un mecanismo de comunicación entre humanos, en concreto entre el ingeniero del conocimiento y el experto del dominio.

A continuación se ofrece una visión general de los formalismos de representación del conocimiento más relevantes que influenciaron la evolución de esta disciplina y la aparición y desarrollo del concepto de ontología.

3.2.3 Evolución de los formalismos de representación del conocimiento.

Desde mediados de los 1980s la componente teórica en la investigación sobre representación del conocimiento va cobrando mayor importancia. En una de las primeras revisiones del estado del arte en representación del conocimiento (Brachman y Levesque, 1985) se hace la siguiente división sobre los diferentes mecanismos utilizados para este cometido:

- Representaciones asociativas.
- Representaciones basadas en objetos estructurados.
- Representaciones basadas en lógica formal.
- Representaciones procedurales y sistemas de producción.
- Otras aproximaciones.

En esta división, que data del año 1985, se observa cuáles eran las ideas más habituales en el campo de la representación del conocimiento hasta ese momento. A grandes rasgos, los diferentes mecanismos pueden agruparse en dos grandes bloques: aquellos que se basaban en el uso de la lógica y aquellos con una representación más informal.

Por un lado, las representaciones asociativas hacen referencia a formalismos basados en grafos del tipo redes semánticas (Sowa, 1991), redes de dependencia conceptual (Shank y Abelson, 1977) y similares, donde la importancia radica en la interconexión de conceptos mediante enlaces. Los objetos estructurados hacen referencia a mecanismos de representación consistentes en estructuras complejas que pretenden recoger el conocimiento de forma más localizada. Ejemplo de esta segunda aproximación son los marcos (Minsky, 1975) o los guiones (Shank, 1975). Ambas aproximaciones están influenciadas y surgen dentro de las investigaciones en psicología cognitiva sobre las estructuras de conocimiento existentes en el cerebro humano.

La lógica formal ofrece un mecanismo de representación del conocimiento basado en teorías lógicas, explicitando la sintaxis y la semántica de una forma totalmente rigurosa, lo que permite que se puedan estudiar las características de expresividad y computabilidad de las diferentes representaciones creadas. Las ideas sobre estos formalismos parten de los estudios de Aristóteles sobre los mecanismos del razonamiento humano. La lógica de predicados de primer orden es el punto de partida más habitual para las diferentes variedades de lenguajes lógicos que se han creado a lo largo de los años para representar estructuras de conocimiento. Se siguen dos estrategias principales para crear un lenguaje de representación basado

en la lógica: reducir la expresividad de la misma en favor de la computabilidad y/o añadir expresividad para hacer frente a tipos de conocimiento fuera del alcance de la lógica de predicados (así se tienen diferentes variantes, como la lógica modal, multivaluada, temporal, etc).

Los sistemas de producción, basados en las reglas de producción, son un mecanismo de representación del conocimiento que surgió del desarrollo práctico de los sistemas expertos y tiene vigencia todavía en la actualidad. La expresión del conocimiento experto en forma de reglas del tipo "SI condición ENTONCES consecuencia" se reveló como una forma muy adecuada de reflejar el conocimiento que un ser humano posee sobre una determinada estrategia para resolver un problema. De hecho, el campo de las "reglas de negocio" (The Bussiness Rules Group, 2000) es cada vez más importante en las organizaciones como forma de establecer su modelo de funcionamiento[21]. Habitualmente las reglas de producción aparecen acompañadas de una serie de hechos que expresan el conocimiento básico de la estructura de conceptos sobre los que se construyen esas reglas.

Los formalismos citados anteriormente son los que más han influido en la evolución y situación actual del campo de la representación del conocimiento. La explicitación del conocimiento en ontologías se realiza en la actualidad con formalismos que han surgido al combinar y ampliar las ideas introducidas por estos esquemas iniciales. A continuación se exponen brevemente las ideas principales y la evolución de estos diferentes formalismos.

3.2.3.1 Las redes semánticas

El formalismo de redes semánticas fue introducido en los años 1960s, siendo el trabajo de Quillian (Quillian, 1967) el considerado como punto de partida. La idea de Quillian era representar las estructuras mediante las cuales se organizaban los conceptos que forman el conocimiento en una manera similar a la postulada en la teoría de memoria asociativa humana. De esta forma, la red semántica podría simular el comportamiento del lenguaje humano.

Las redes semánticas son grafos orientados que cuentan con dos estructuras conceptuales: los nodos y los arcos. Los nodos representan a los conceptos mientras que los arcos, que van de uno a otro nodo, representan relaciones (ya sean lingüísticas, físicas, conceptuales, etc.) entre esos conceptos.

[21] El campo de las reglas de negocio, aunque surgió a partir de las ideas de la inteligencia artificial, se desarrolla hoy en día de forma independiente de esta disciplina.

El éxito y aceptación de este formalismo fue considerable gracias a lo intuitivo de su formulación y a la facilidad para la implementación del mismo en un ordenador por medio de listas enlazadas.
La mayor desventaja de las redes semánticas radica, paradójicamente, en la falta de contenido semántico que presenta la estructura tal como se planteaba inicialmente. Por citar algunos problemas se puede mencionar:

- No se distingue entre nodos que representan a clases de individuos y nodos que representan a individuos particulares.
- No se distingue entre arcos que representan conocimiento estructural y los que representan conocimiento asertivo (sobre afirmaciones).
- El significado de los arcos depende totalmente de la interpretación que haga el usuario, solamente aparecen denotados por una etiqueta textual.

En general se reclamaba una mayor base formal y una mayor descripción de la semántica de nodos y arcos. Muchos de estos problemas fueron puestos de manifiesto en (Woods, 1975) y tuvieron una amplia repercusión en la aparición y desarrollo de las denominadas lógicas descriptivas en años posteriores y, por tanto, también en la evolución hacia el concepto de ontología.

3.2.3.2 Los marcos

Los marcos son una idea coetánea de las redes semánticas y, al igual que éstas, perseguían encontrar la estructura más adecuada para describir formalismo dedicado a la representación del conocimiento humano. La idea es original de Minsky (Minsky, 1975) que, inicialmente, describió a los marcos como trozos de conocimiento que representan situaciones estereotípicas, como por ejemplo ir a una fiesta o estar en una habitación (la hipótesis de Minsky es que cuando un ser humano enfrenta una situación recupera de la memoria uno de estos marcos, o una serie de ellos, que se adaptan a esa situación). El marco recoge toda la información relevante necesaria para distinguir esa situación de otras. Esta información se recoge en "slots" que reflejan las propiedades o características definitorias de la situación descrita por el marco.

El artículo de Minsky es bastante impreciso a la hora de describir de forma explícita lo que son los marcos y su funcionamiento, aunque se ofrece una visión general de cómo funcionan. Los marcos también se pueden agrupar en redes (al estilo, y con la influencia, de las redes semánticas) mediante relaciones que se establecen entre los mismos, pero ofrecen como característica muy relevante y novedosa la posibilidad de ser agrupados en jerarquías de herencia que hacen que los marcos "hijo" hereden los slots de los marcos "padre". El trabajo de Minsky fue posteriormente ampliado y mejor definido desde el punto de vista computacional y de la representación del conocimiento en los ordenadores (los

trabajos de Minsky, al igual que los de Quillian, se desarrollaban en el campo de la psicología congnitiva).

La idea de los marcos fue criticada por Hayes (Hayes, 1979) entre otros, haciendo especial hincapié en la necesidad de consistencia en la representación del conocimiento como forma para asegurar un buen proceso de razonamiento. Hayes también postula que la idea de marcos no es más que una nueva sintaxis para la lógica de primer orden[22]. De cualquier forma, la idea de representación de la complejidad, incluyendo la jerarquía de herencia y la idea de razonamiento sobre las estructuras de conocimiento, son dos importantes aportaciones de esta propuesta que influenciaron los desarrollos posteriores.

Otra de las aportaciones más relevantes de los marcos es el concepto de asignación de procedimientos (procedural attachment), que consiste en que algunos de los slots pueden tener asignados procedimientos para responder ante ciertos eventos de forma que puedan calcular o modificar los valores que contienen (de hecho el concepto de asignación de procedimientos a los slots es un precedente de la encapsulación de métodos en el paradigma software de orientación a objetos).

La idea de los marcos influyó decisivamente en el desarrollo de los lenguajes de programación orientados a objetos, como el SmallTalk y posteriores y es también la base de algunos formalismos de representación de ontologías, como se verá más adelante. Los marcos pretenden representar la complejidad de las situaciones; de acuerdo a Minsky, el conocimiento debe estar recogido de forma localizada en estructuras complejas y no en muchos pequeños axiomas lógicos. El ordenador, en este sentido, es una herramienta de representación muy útil para tratar la complejidad.

3.2.3.3 La lógica formal

La lógica ha jugado diferentes papeles a lo largo de los años en el campo de la representación del conocimiento. Como mecanismo genérico de representación del razonamiento humano fue la primera propuesta de la inteligencia artificial cuando se buscaba un formalismo genérico para emular el comportamiento del cerebro. Pronto esta idea se abandonó, llegando a la conclusión de que la lógica puede representar ciertos tipos de razonamiento, decidiendo sobre su validez formal, pero no cualquier tipo de razonamiento humano.

[22] Este hecho, el de la lógica como formalismo universal al que puede ser resumido cualquier otra aproximación, es un aspecto recurrente en la discusión sobre diferentes esquemas de representación del conocimiento.

Los métodos de representación del conocimiento como redes semánticas y marcos surgen como alternativas a la lógica, influenciados por investigaciones en el área de la psicología cognitiva y sin basarse en ningún formalismo lógico. Sin embargo, como se ha visto, aparecen críticas que postulan que estas aportaciones se reducen al dominio de la sintaxis. Estas críticas hicieron que la lógica se recuperase como lenguaje de representación del conocimiento, argumentando que sólo mediante esta formalización se podrían estudiar sistemáticamente las propiedades de un determinado formalismo.

Se asume que mediante la lógica no se puede representar todo el razonamiento humano pero también que sí puede representarse todo aquel que puede ser implementado en un ordenador. De hecho, la capacidad expresiva de la lógica de primer orden sobrepasa la capacidad computacional de los ordenadores. Los dos aspectos que definen la adecuación computacional de un lenguaje de representación basado en lógica son la computabilidad y la complejidad computacional. Mientras más expresividad permita un determinado lenguaje lógico más elevada será su complejidad computacional, hasta el punto de hacerlo intratable o indecidible. Esta dicotomía entre expresividad y tratabilidad, bien conocida desde las primeras etapas de la ingeniería del conocimiento (Levesque y Brachman, 1985), es el punto de partida para crear esquemas y lenguajes de representación del conocimiento basados en versiones reducidas (en cuanto a expresividad) de la lógica de primer orden que mantengan buenas características en cuanto a complejidad computacional.

La potencia expresiva de la lógica para representar diferentes estructuras de representación del conocimiento ha hecho que, a lo largo de los años, la lógica haya sido utilizada como formalismo de representación primario, como formalismo "neutro" para traducir entre diferentes esquemas de representación (donde KIF - Knowledge Interchange Format (Genesereth y Fikes, 1992) es el ejemplo más claro) o para ofrecer una base formal a ideas menos formales como las de los marcos y objetos (como en el caso de la denominada F-Logic o Frame Logic (Kifer et. al., 1995) o en Ontolingua (Farquhar et. al., 1996)).

La combinación de la lógica con las ideas surgidas de los campos de las redes semánticas, marcos y redes de herencia estructural (que se verán a continuación) dieron lugar a la aparición de las denominadas lógicas descriptivas (description logics - DL - introducidas también en el siguiente apartado) que fueron el punto de partida para una familia de lenguajes de representación del conocimiento que ha evolucionado hasta la actualidad, donde el lenguaje OWL se ha convertido en el paradigma de la representación del conocimiento mediante formalismos lógicos (el proceso de evolución hasta la aparición de OWL se tratará en el apartado 3.3.5).

Por último cabe mencionar que, recientemente, se ha retomado la idea con la que se desarrolló el lenguaje KIF, a la luz de la evolución e influencia de las lógicas descriptivas y la explosión en la investigación y desarrollo en el ámbito de la Web Semántica. El lenguaje Common Logic (CL)[23] (ISO, 2007) pretende ser un marco general para recoger las construcciones de una familia de lenguajes basados en lógica, incluyendo las semánticas de todos ellos y funcionando, por tanto, como una herramienta para la interoperabilidad entre los mismos.

3.2.4 De las redes de herencia estructural a las lógicas descriptivas

La idea de introducir claridad en el formalismo de las redes semánticas data de finales de los años 1970s y se inicia con el trabajo de Brachman (Brachman, 1977). En este trabajo se plantea el uso de estas redes para representar solamente relaciones estructurales entre conceptos, restringiendo y explicitando así el significado de los arcos. Los elementos con los que se construye este formalismo son:

- Conceptos
- Superconceptos
- Roles (posibles relaciones con otros conceptos).
- Descripciones estructurales (relaciones entre roles).

Los conceptos no se ven como entidades atómicas sino como descripciones complejas. Existen conceptos genéricos, que pretenden denotar clases de individuos, y conceptos individuales, que pretenden denotar a individuos particulares.

A su vez, los conceptos genéricos pueden ser primitivos (aquellos que sólo pueden ser determinados de forma parcial con su descripción, es decir, que no se pueden describir exhaustivamente) y conceptos definidos (aquellos cuyo significado es completamente determinado por su descripción). El significado de un concepto solo dependerá del significado de sus superconceptos asociados y, en su caso, de su grado de primitividad.

Las principales relaciones que se establecen son:

- De especialización (subclase o subtipo).
- De individualización.
- Arbitrarias, según establecidas por los roles creados.

[23] El grupo de trabajo inicial para el desarrollo de Common Logic está compuesto, entre otros, por Pat Hayes, John Sowa y Michael Gruninger.

Lo más destacable de esta aproximación es que las redes semánticas se usan para representar un conocimiento muy específico y acotado: la relación estructural entre conceptos.

El éxito de los sistemas de este tipo radica en que no trata de proporcionar una solución a cada problema de representación, sino que sólo se encarga de estudiar la forma de cómo *definir conceptos*, dejando de lado cualquier tipo de conocimiento que no esté relacionado con la definición. Por esta razón a este tipo de formalismos se les llama también representación terminológica de conocimiento ("terminological knowledge representation (or language)"), representación definicional de conocimiento ("definitional knowledge representation (or language)") o lenguaje conceptual ("concept language"). Hoy en día este tipo de formalismo se expresa mediante una base lógica formal que se conoce como "lógicas descriptivas" (DL – Description Logics) (Baader et. al. eds., 2007). Este tipo de lógicas tienen como punto fuerte la capacidad de expresar condiciones necesarias y/o suficientes para definir unos conceptos a partir de otros en base a restricciones en sus propiedades (roles). Las lógicas descriptivas son subconjuntos de la lógica de primer orden con expresividad reducida (con la excepción de algunos operadores extra en algunas variedades).

Aparte de la red estructural de conceptos, que parte de la idea de las redes semánticas, esta aproximación tiene también relación con los marcos, ya que los roles tienen relación con el concepto de slots surgido de aquellos. Las lógicas descriptivas son, pues, la evolución de las redes de herencia estructural que, a su vez, son el resultado de la combinación de ideas de los tres ámbitos mencionados (marcos, redes semánticas y lógica).

La primera implementación relevante de las redes de herencia estructural basadas en lógicas descriptivas fue el sistema KLONE[24] (posteriormente denominado KL-ONE) (Brachman, 1978), (Brachman y Schmolze, 1985). Este sistema dio paso a otros que han evolucionado a lo largo de los años hasta la actualidad. Entre ellos se pueden citar KRIPTON (sobre el año 1983), KANDOR (sobre el año 1984) (Patel-Schneider, 1984), NIKL (sobre el año 1983), BACK (sobre el año 1985) (von Luck et. al., 1987), LOOM (sobre el año 1987) (McGregor, 1988) o CLASSIC (sobre el año 1989) (Borgida et. al., 1989). Cada uno de estos sistemas tiene sus propias características respecto a corrección, completitud, decidibilidad y tratabilidad[25], aunque en un principio los estudios respecto a estas características no se realizaban o se hacían respecto a alguna de ellas solamente (Baader y Sattler, 2001). En general, mientras mayor expresividad tiene un sistema lógico

[24] Inicialmente estaba basado en representaciones gráficas no formalizadas con lógica.

[25] Corrección hace referencia a la capacidad de los mecanismos de inferencia que pueden implementarse sobre ese esquema de representación para hacer que todos y cada uno de las conclusiones obtenidas sea correcta. El mecanismo de inferencia se dice que es completo si todas las conclusiones correctas que puedan obtenerse de la representación del conocimiento pueden ser obtenidas con ese mecanismo.

más difícil será lograr que sea correcto y completo, pudiendo llegar a afectar a la tratabilidad y la decidibilidad (Tobies, 2001), (Donini et. al., 1996).

Con la mejor comprensión de las técnicas para estudiar las características de computabilidad y complejidad computacional de estos sistemas, algunas aproximaciones más modernas como FACT (Horrocks, 1998) intentan conservar la expresividad en algún aspecto importante como la definición de conceptos y también implementar razonadores completos con una eficiencia computacional asumible. Dentro de esta categoría se encuentra también OWL-DL (Ontology Web Language - Description Logic), de cuya aparición y evolución se hablará en el siguiente apartado, ya que se ha convertido en el lenguaje más utilizado para la formalización de ontologías mediante un formalismo lógico.

Todos estos sistemas cuentan con lenguajes capaces de expresar la estructura terminológica construida con descripciones, especificando los roles de cada concepto, así como las características de los mismos, como la cardinalidad, el tipo de datos o conceptos que pueden "rellenar" estos roles, el número de datos que puede aparecer en cada rol, etc. El mecanismo básico de razonamiento automatizado que los sistemas basados en lógicas descriptivas pueden realizar es la clasificación automática de conceptos, es decir, dado un concepto el sistema es capaz de clasificarlo debajo de los súper-conceptos padre adecuados. La principal relación que se establece entre los conceptos en la red terminológica, y sobre la que actúa el mecanismo de inferencia, es la de subsunción o especialización. Este tipo de relación va más allá de la jerarquía de herencia (también denominada relación IS-A) que se encuentra en la orientación a objetos, ya que no se limita a especificar las propiedades que se heredan sino que los conceptos se definen según estas propiedades. Dicho de otro modo, se ofrece una definición del concepto en vez de una descripción del mismo.

Otra particularidad propia de este tipo de lenguajes es la clara separación entre la red estructural de definición de conceptos, llamada "Terminological Box" o T-Box, y la de instancias particulares y sus relaciones, denominada "Assertional Box" o A-Box, aunque no todos los sistemas citados implementan estas dos construcciones. En el caso del A-Box, el mecanismo de inferencia automatizable más habitual es la comprobación de instancias (instance checking), es decir, determinar si un determinado individuo en el A-Box es una instancia de un determinado concepto en la T-Box[26].

[26] Para ver una lista completa y detallada de razonamientos automatizados posibles sobre un sistema de representación del concocimiento basado en lógicas descriptivas ver (Donini et. al., 1996).

3.3 Las ontologías

3.3.1 Concepto de ontología

El término "ontología"[27] comienza a ser empleado de forma generalizada en la comunidad de la ingeniería del conocimiento a partir del trabajo de Gruber (Gruber, 1993)[28] en el que se ofrecía una definición de dicho término como "una especificación de una conceptualización". Esta definición es bastante vaga y se ha intentado en varias ocasiones ofrecer alguna más descriptiva. Entre ellas se pueden citar:

- Una ontología es una especificación explícita de una conceptualización (Gruber, 1995).
- Una ontología es una representación de un sistema conceptual por medio de una teoría lógica (Guarino y Giaretta, 1995).
- Una ontología es una teoría de qué entidades pueden existir en la mente de un agente conocedor (que posee conocimiento) (Wielinga y Schreiber, 1993).
- Una ontología es una serie de acuerdos acerca de una conceptualización compartida (Gruber, 1993).

En estas definiciones se expresan importantes características que todas las ontologías comparten.

Una ontología refleja una *conceptualización*. Este término, introducido por Genesereth y Nilsson en (Genesereth y Nilsson, 1987) es la base de la definición de ontología. Sin embargo, desde su primera aparición, ha suscitado controversias sobre qué es exactamente una conceptualización y cómo se debe definir (Guarino y Giaretta, 1995). De forma genérica puede decirse que una conceptualización es una visión del mundo respecto a un cierto dominio, estando formada por un conjunto de conceptos (entidades, procesos, atributos, etc.) junto con sus definiciones e interrelaciones (Uschold y Grüninger, 1996).

La representación del conocimiento recogida en una ontología es *explícita*, lo que quiere decir que tiene entidad como tal, de forma aislada de cualquier uso que se haga de ella. En una aplicación informática esta explicitación sirve para separar el

[27] El término ontología se adopta del campo de la filosofía, donde fue definido por primera vez por Aristóteles con el significado de "estudio de lo que existe". Tiene, por lo tanto, relación con la metafísica.

[28] Existen documentos anteriores que recogen este término con un significado similar, por ejemplo (McCarthy, 1980) o (Hayes, 1985), aunque su uso generalizado se produce a partir del trabajo de Gruber. Para una descripción detallada del origen y uso del término ver (Guizzardi, 2005)

conocimiento y el código del programa: el modelo existe aunque no haya aplicación que lo utilice.

La idea de *teoría lógica* subyace en la mayoría de los sistemas de representación del conocimiento y también en las ontologías. Las ontologías contienen términos y axiomas que restringen la interpretación de esos términos (Sowa, 2000).

El conocimiento representado en una ontología puede pertenecer y ser usado por un *agente conocedor*, una forma genérica de referirse a cualquier elemento que puede manejar conocimiento, ya sea un ser humano o un agente software. Aunque habitualmente el uso que se hace del término está dirigido al procesamiento automático en un ordenador, la ontología es una estructura perteneciente al denominado nivel del conocimiento (Newell, 1982). En este sentido la ontología recoge los diferentes tipos de conceptos y sus relaciones, desde un punto de vista neutro respecto al formalismo de representación. Los sistemas de representación del conocimiento deben proporcionar el mecanismo para formalizar la ontología e introducirla en el ordenador para su procesamiento automático.

La idea de *acuerdo* también está incluida en el concepto de ontología. La existencia de un grupo de agentes que contraigan el compromiso de utilizar la conceptualización explicitada en la ontología para intercambiar conocimiento es primordial[29].

La idea de acuerdo también es central en el concepto de *reutilización*. Si una ontología es un acuerdo sobre una representación de un cierto dominio de conocimiento ésta podrá utilizarse para implementar otras aplicaciones o para construir otras ontologías que empleen o extiendan a la primera. El concepto de reutilización se considera clave hoy en día para la adopción de las ideas de modelado del conocimiento (también en el ámbito de la ingeniería del software) y ya estaba contemplado en los primeros estudios sobre ontologías (Gruber, 1995). Para que las ontologías puedan ser reutilizadas de forma sencilla deben existir métodos para que puedan ser evaluadas, alineadas, insertadas en otras ontologías, etc.

3.3.2 Estructuras que se consideran ontologías

La definición del término ontología no es un asunto sobre el que haya un acuerdo total, existiendo desde los inicios una controversia en cuanto al significado de dicho término y de aquellos que suelen aparecer en su definición (Guarino y Giaretta, 1995). También existen trabajos y autores que recogen bajo el término

[29] Se dice que el conjunto de agentes que forman parte del consenso sobre una determinada ontología tienen un compromiso (commitment) con esa ontología.

ontología diferentes esquemas de representación de datos e información (Lassila y McGuinness, 2001).

Los diccionarios, tesauros, etc, son considerados ontologías en algunos ámbitos. Se han acuñado términos como ontología ligera (lightweight ontology) para definir una jerarquía simple de conceptos (por citar un ejemplo). De este modo, atendiendo al tipo de construcciones conceptuales que contienen, se podrían encontrar diferentes tipos de "estructuras parecidas a ontologías" (McGuinness, 2003). La noción o versión más simple de ontología sería un vocabulario controlado (una lista de términos), por ejemplo un catálogo. El siguiente paso en complejidad sería el glosario (una lista de términos con su significado expresado en lenguaje natural). Otro paso más serían los tesauros, que contienen información adicional de tipo semántico (por ejemplo una taxonomía de la especificidad de los términos). Todos estos sistemas están pensados para ser usados por un agente humano, no computacional. Para que un ordenador pueda utilizarlas, las definiciones deben ser inambiguas y estar expresadas por medio de algún formalismo. El tipo de ontología procesable por ordenador más simple consistiría así en una jerarquía que recoja la relación de generalización/especialización entre los términos[30]. La figura 3.1 recoge dos clasificaciones sobre sistemas de representación que pueden ser considerados ontologías según sus autores. La figura 3.1a está tomada de (Uschold, 2006), la 3.1b de (McGuinness, 2003)

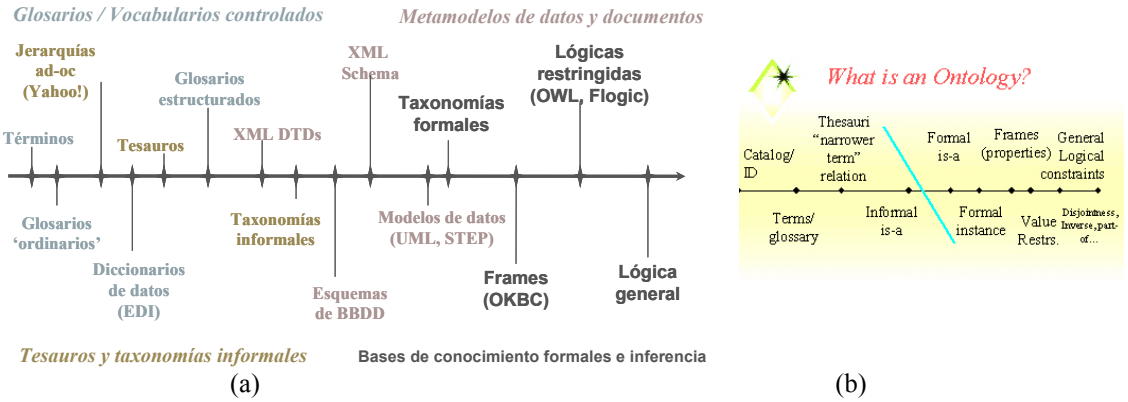


Figura 3.1 Estructuras que pueden ser consideradas ontologías.

En cualquier caso, la idea original de ontología en el campo de la inteligencia artificial no se limita a un vocabulario más o menos estructurado en una jerarquía, sino que incluye también la idea de conceptualización de un determinado dominio por medio de ese vocabulario. El término ontología, según su definición original,

30 En (OMG, 2007) se da una definición de ontología de acuerdo a estas consideraciones: "Una ontología define los términos y conceptos (significado) utilizados para describir y representar un área de conocimiento. Una ontología puede ir, en cuanto a expresividad, desde una taxonomía (conocimiento con mínima jerarquía o estructura padre/hijo), a un tesauro (palabras y sinónimos), a un modelo conceptual (con conocimiento más complejo), a una teoría lógica (con conocimiento muy rico, complejo, consistente y significativo)"

abarcaría también relaciones entre conceptos, restricciones en propiedades, definiciones, axiomas, etc.

Puede resumirse la diferenciación entre los esquemas vistos atendiendo al hecho de que los vocabularios controlados, taxonomías y tesauros están dirigidos a recoger y organizar términos, los modelos de datos y modelos de objetos están dirigidos a representar la información en ordenadores, mientras que las ontologías están dirigidas a representar el conocimiento de forma independiente del agente que lo utilice.

3.3.3 Tipos de ontologías

La caracterización de las ontologías se puede realizar a diferentes niveles y con diferentes criterios. A continuación se presentan dos de las clasificaciones más relevantes y utilizadas.

Desde el punto de vista del grado de formalidad en la expresión de la ontología se pueden tener (Uschold y Grüninger, 1996):

- Ontologías expresadas informalmente. Para su explicitación se utiliza el lenguaje natural.
- Ontologías expresadas semi-informalmente. Se expresan en algún tipo de lenguaje natural controlado (CNL -Controlled Natural Language)
- Ontologías expresadas semi-formalmente. Expresadas por medio de un lenguaje artificial.
- Ontologías expresadas de forma rigurosamente formal. Expresadas por medio de semántica formal basada en teorías lógicas, con teoremas y demostraciones.

Según el nivel de generalidad del conocimiento recogido en la ontología (o del nivel de dependencia de respecto a una tarea o punto de vista particular) se pueden tener (Guizzardi, 2005):

- Ontologías de alto nivel (o nivel superior, top-level), que describen conceptos muy generales, aplicables casi universalmente. Espacio, tiempo, materia, acción, etc.
- Ontologías de dominio y de tarea que describen los conceptos que aparecen en un determinado dominio (como la medicina) ó tareas genéricas (como diagnóstico).
- Ontologías de aplicación. Describen conceptos que dependen tanto de un dominio como de una tarea particular.

3.3.4 Componentes de una ontología

Se considera que una ontología está formada por diferentes elementos que, conjuntamente, forman el modelo conceptual de un determinado dominio de conocimiento. De acuerdo a (Gruber, 1993) los elementos que constituyen una ontología son:

- Conceptos.
- Relaciones entre conceptos.
- Funciones.
- Axiomas.
- Instancias.

Los conceptos son descripciones y abstracciones de conjuntos de individuos que comparten ciertas propiedades o atributos. Las relaciones representan interacciones entre conceptos (las propiedades o atributos que reflejan las características de los conceptos se tratan como relaciones). Las funciones son un tipo de relación especial. Los axiomas son sentencias, expresadas sobre los conceptos y relaciones, que siempre son verdaderas sin necesidad de prueba. Las instancias son representaciones concretas de los conceptos, de individuos que existen en un determinado momento y contexto[31].

Respecto a la caracterización del conocimiento contenido en una ontología se puede hacer una división en dos grandes bloques (Chandrasekaran et. al., 1999):

- Conocimiento factual del dominio. Formado por el conocimiento estático del dominio.
- Conocimiento sobre resolución de problemas, describe cómo lograr ciertos objetivos manejando los conceptos de la parte estática. Es el conocimiento de carácter dinámico del dominio.

La separación entre estos dos tipos de conocimiento es un logro de la investigación en ontologías, ya que permite desarrollar un modelo de conocimiento factual de forma independiente al conocimiento sobre resolución de problemas y, aun más, permite que el conocimiento sobre resolución de problemas pueda ser generalizado y no estar totalmente condicionado por una tarea concreta. Sin embargo, en la práctica, esta separación no es fácilmente realizable.

La representación del conocimiento mediante ontologías consiste en la reutilización de ontologías y existentes y/o creación de otra u otras nuevas para lograr un esquema conceptual que recoja el conocimiento de un determinado

[31] La nomenclatura acerca de los diferentes elementos que componen una ontología es muchas veces confusa, debido a las diferentes nominaciones que aparecen relacionadas con los diferentes formalismos y también a la imprecisión en el uso de las mismas. Por ejemplo, todos los elementos citados (conceptos, relaciones e instancias) podrían ser denominados "conceptos", ya que forman parte de la conceptualización.

dominio. Por esta razón, a veces se habla de ontología en singular ó de ontologías en plural. En este trabajo se utiliza el término en singular, ontología, para hacer referencia a la estructura total que recoge el conocimiento que quiere representarse, sin tener en cuenta si esta estructura está formada por una o varias ontologías separadas. Además, se habla de la "parte estática" y la "parte dinámica" de la ontología para referirse al conocimiento factual y de resolución de problemas respectivamente.

La creación de ontologías comenzó siendo, y todavía lo es en parte, un arte. Sin embargo, desde el comienzo se pretendió que la disciplina tuviese un tratamiento de ingeniería en todos sus aspectos y por lo tanto se llevaron a cabo trabajos que intentan clarificar los aspectos constructivos (Guarino y Welty, 2004), definir metodologías de construcción (Fernández et. al., 1997) y métodos de evaluación (Sure et. al., 2004) así como mecanismos para la reutilización (Pinto y Martins, 2004).

3.3.5 Formalismos para la construcción de ontologías

Las ontologías son un mecanismo de representación del conocimiento que pertenecen al denominado "nivel del conocimiento" (ver apartado 3.3.1), es decir, una ontología puede existir en la mente de un ser humano sin que haya reflejo de la misma en ningún soporte físico externo; sin embargo, en el contexto de la computación, estas estructuras se introdujeron con la idea de ser implementadas y automatizadas en un ordenador. La implementación de las estructuras de una ontología en un lenguaje reconocible por un ordenador se denomina formalización. En un proceso de formalización siempre se perderá algo de la expresividad que contiene una ontología expresada en el nivel del conocimiento porque el ordenador (los lenguajes que el ordenador puede manejar) tiene un límite expresivo (de ahí surgen los problemas de computabilidad y complejidad computacional mencionados en apartados anteriores). A lo largo de los años se han utilizado diferentes formalismos y lenguajes para representar ontologías. Desde las aproximaciones provenientes de los lenguajes utilizados para la construcción de los sistemas expertos en los años 1980s, hasta las lógicas descriptivas y los lenguajes creados para la Web Semántica en Internet, la evolución ha sido constante. En (Corcho et. al., 2003), (Gómez-Pérez y Corcho, 2002), (Pulido et. al., 2006) se pueden encontrar buenas monografías sobre este tema. A continuación se ofrece una visión personal y resumida sobre esta evolución.

Los formalismos de representación del conocimiento aparecen muchas veces ligados a herramientas o sistemas que forman un entorno completo para la creación de modelos conceptuales. Pueden poseer una base muy formal, basándose en una teoría lógica y, por lo tanto, tener una semántica independiente

de interpretación externa o estar expresados por medio de lenguajes menos formalizados, sin expresar la semántica por medio de teorías lógicas. Ejemplo de los primeros son los lenguajes y sistemas basados en KIF, en lógicas descriptivas o cualquier otro formalismo lógico. Se puede citar entre ellos Ontolingua (Farquhar et. al., 1996), OCML (Operational Conceptual Modelling Language) (Motta, 1998), F-Logic (Kifer et. al., 1995) y todos los sistemas basados en lógicas descriptivas vistos en el apartado 3.2.4. Ejemplos de los segundos son KEE (Knowledge Engineering Environment) (Fikes y Kehler, 1985), CLIPS[32], Protégé (Genari et. al., 2002) (sólo en su versión denominada Protégé-frames), etc, pudiéndose incluir aquí también lenguajes provenientes de la ingeniería del software, que han sido propuestos para la realización de ontologías, como UML (Kogut et. al., 2002), (Cranefield y Purvis, 1999).

Dentro de los formalismos que cuentan con una teoría lógica formal, muchos de ellos contienen también estructuras de alto nivel basadas en los marcos para facilitar la creación y edición de ontologías. Ontolingua es el lenguaje paradigmático del principio de la década de de los 1990s, ofreciendo la posibilidad de desarrollar las ontologías a través de un servidor central conectado a Internet. La semántica de Ontolingua está basada en KIF. Ontolingua permite la expresión de las estructuras de conocimiento propias de los marcos mediante un formalismo lógico, por lo que puede verse como una combinación de las representaciones basadas en marcos y en lógica. La finalidad de Ontolingua es ofrecer un formalismo neutro para el intercambio de ontologías. De este modo, la propia herramienta ofrece la posibilidad de traducir la representación a otros sistemas como CLIPS, LOOM o KIF.

Dentro de los formalismos basados en marcos merece hacer especial mención a la herramienta Protégé. Esta herramienta, cuyo origen data de los años 1980s, implementó, a mediados de los años 1990s, una interfaz gráfica para la adquisición del conocimiento basada en el formalismo de los marcos. En su evolución se utilizaron, a modo de mecanismo de representación del conocimiento, objetos de Lisp y, posteriormente, el lenguaje CLIPS. Desde mediados de los 1990s el formalismo cumple la mayoría de las construcciones de conocmiento especificadas en el protocolo OKBC[33,34] (Chaudhri et. al., 1998a y b) (Open Knowledge-Base Connectivity, protocolo desarrollado en parte a partir de la experiencia con Ontolingua). La evolución de la herramienta Protégé (desde

[32] CLIPS - C-Language Integrated Production System, una shell de sistema experto creada en la NASA en los años 1980, desarrollado a partir de OPS5 (Official Production System versión 5), un lenguaje para la expresión de sistemas expertos basados en reglas que fue utilizado para la programación de diversos sistemas en los años 1980s.

[33] OKBC es una especificación sobre una serie de operaciones genéricas para interactuar con una base de conocimiento construida con un formalismo que puede describirse por medio de las construcciones típicas de los marcos y la orientación a objetos -OKBC es el sucesor de GFP, Generic Frame Protocol-. En cierto sentido puede considerarse complementario a un lenguaje de representación del conocimiento como KIF.

[34] El nombre OKBC se escogió por analogía con ODBC (Open DataBase Conectivity), una tecnología para el acceso a bases datos de forma independiente del fabricante de la misma.

su origen hasta el año 2000) puede verse en (Noy et. al., 2000). Hoy en día Protégé es un editor de ontologías tanto para el formalismo de los marcos como para el de las lógicas descriptivas (en concreto para OWL).

La inclusión de una API para acceder a las estructuras de la ontología desde el código de una aplicación fue uno de los factores clave para la aceptación de Protégé por parte de una amplia comunidad de usuarios, permitiendo tanto la expansión de las funcionalidades de la propia herramienta como la creación de aplicaciones que acceden a la estructura de conocimiento almacenada en la ontología. La herramienta Protégé, al contrario que Ontolingua, es una aplicación de escritorio (no de acceso remoto a un servidor) distribuida bajo licencia de código abierto. La mayor dificultad de uso de Ontolingua y los problemas a la hora de emplear las ontologías desarrolladas en aplicaciones informáticas prácticas hizo que esta aproximación no tuviese una aceptación general, siendo Protégé la elección más habitual durante el final de los 1990s e incluso lo sigue siendo en la actualidad (dentro de las distribuidas bajo licencia de código abierto).

Las lógicas descriptivas, con estudios ya avanzados en el momento de asentamiento y expansión del concepto de ontología, se postularon también como un formalismo adecuado para ofrecer una base lógica formal que podría permitir la expresión, evaluación y reutilización de ontologías de forma más sencilla y con posibilidad de automatización. El uso de este formalismo como lenguaje para la representación del conocimiento en ontologías se vio impulsado sobre todo a partir de finales de los 1990s gracias a su aplicación en el emergente campo de la Web Semántica, tal como se expone a continuación.

3.3.5.1 El papel de Internet y el servicio Web en la evolución de la representación del conocimiento

Los últimos años de la década de los 1990s marcaron un hito en la aplicación práctica y en la evolución de las ideas del campo de la representación del conocimiento. En esta época, el servicio Web basado en la creación de documentos de texto plano, está dando paso a la Web dinámica, generada a partir de información almacenada en bases de datos transparentes para el usuario. La idea original de Tim Berneers Lee[35] (Berneers-Lee, 1998) para la Web está sin embargo todavía lejos de ser alcanzada. Cada vez hay más información, tanto en documentos de texto HTML como en bases de datos empresariales. La búsqueda de documentos HTML es eficiente y rápida, gracias a los robots buscadores basados en palabras clave, pero la interpretación del contenido de los documentos encontrados y la relevancia de los mismos queda totalmente a cargo del usuario.

[35] El "padre" del servicio Web lo describió como "un universo de información accessible en red" que fuese, en primer lugar, un medio de comunicación entre humanos y, posteriormente, un espacio en el que los agentes software puedan convertirse en herramientas que trabajen conjuntamente con los seres humanos.

Se buscan entonces mecanismos para conseguir trasladar a los ordenadores parte de este procesamiento sobre el contenido y la relevancia de los documentos, añadiendo para este fin contenido semántico a las palabras que aparecen en los mismos.

Inicialmente se propuso RDF (Resource Description Framework) (Klyne y Carroll eds., 2004) (publicado como recomendación por el W3C en 1999) como marco general para la descripción semántica de recursos en Internet, usando sintaxis XML (Extensible Markup Language) para incluir esa información en los documentos Web. A continuación RDFS (RDF Schema) (Brickley y Guha eds., 2004) se introdujo (en el año 2000 como candidato para recomendación) para dotar de estructura a los conceptos y atributos utilizados en la descripción de recursos con RDF. Finalmente, el uso de ontologías para definir los términos empleados para etiquetar los documentos se plantea desde principios del siglo XXI como una extensión a las limitaciones expresivas de RDF(S)[36].

OIL (Ontology Inference Layer) (Bechhofer et. al., 2000) y DAML (Darpa Agent Markup Language) (Hendler y McGuinness, 2000) son dos proyectos coetáneos que estaban dedicados a este fin. OIL afronta la tarea postulando la aplicación del campo de las lógicas descriptivas al conjunto RDF/XML. Entre tanto DAML también se desarrolla, pretendiendo crear un lenguaje formal para definir las estructuras típicas de la orientación a objetos y los marcos. Sobre el año 2001 ambas iniciativas convergen e inician un camino común, denominándose DAML+OIL (McGuinness et. al., 2002). En esta misma época el W3C comienza a estudiar la posibilidad de lanzar un lenguaje formal para la especificación de ontologías en la Web. El proyecto comienza planteando una serie de casos de uso para definir las propiedades requeridas del lenguaje (Heflin ed., 2004), trabajando a partir de los desarrollos y experiencias obtenidas con DAML+OIL y pretendiendo que exista total compatibilidad con la idea de RDF (Horrocks et. al., 2003). El resultado final es OWL (Web Ontology Language) (W3C, 2004), promovido como recomendación por parte del W3C en febrero de 2004[37].

La Web Semántica es la principal causa que ha hecho salir la investigación en lógicas descriptivas del ámbito académico al de la aplicación práctica (McGuinness, 2001) y OWL es el lenguaje elegido como estándar para la representación del conocimiento en ontologías en ese ámbito. En concreto, el W3C ha planteado tres variantes de este lenguaje que poseen diferentes grados de expresividad: OWL-Lite, con una expresividad que permite la definición de jerarquías simples de conceptos; OWL-DL, que presenta una expresividad dada por un lenguaje perteneciente a las lógicas descriptivas (con mayor expresividad que OWL-Lite) y por último OWL-Full, que presenta la mayor expresividad y

[36] El término RDF(S) se utiliza para recoger a RDF junto a RDFS.

[37] La recomendación final para RDF, RDFS y OWL se produjo conjuntamente en febrero de 2004

compatibilidad total con RDF(S) pero con serios problemas en cuanto tratabilidad e incluso decidibilidad (Antoniou y van Harmelen, 2004a).

El impacto de la Web Semántica en el concepto de ontología ha hecho que, en la actualidad, muchos de los desarrollos prácticos de ontologías, estén dedicadas o no al ámbito del servicio Web, se lleven a cabo con OWL[38], llegándose incluso a recoger las ideas y técnicas relacionadas con las ontologías bajo el nombre genérico de "herramientas de la Web Semántica".

3.3.6 Metodologías para la construcción y evaluación de ontologías

Como se ha mencionado, el proceso de desarrollo de una ontología tiene todavía gran componente de arte aunque, desde comienzos del siglo XX, se han estudiado y definido metodologías para su creación y evaluación[39]. Muchas de estas metodologías surgieron a partir de proyectos de investigación en los que se desarrollaba alguna ontología concreta, teniendo en cuenta las metodologías del campo de la ingeniería del software. En (Fernández, 1999) puede consultarse un estudio comparativo de algunas metodologías para la creación de ontologías.

Habitualmente la metodología recoge las fases a llevar a cabo en el proceso de construcción, planteando etapas similares a las existentes en el desarrollo de software tradicional. La particularidad del conocimiento representado en una ontología hace que, para identificar los conceptos, se planteen técnicas como el esbozo de escenarios de uso, preguntas de competencia, recopilación de información, etc. Otro de los aspectos que algunas metodologías plantean es la estrategia en la creación de los conceptos. Ésta puede ser de abajo hacia arriba (abstracción o generalización a partir de los conceptos más específicos), de arriba hacia abajo (especialización a partir de conceptos de alto nivel de abstracción), o del medio hacia arriba y hacia abajo (middle-out).

La madurez de estas metodologías de construcción todavía no se ha alcanzado de forma total, siendo METHONTOLOGY (López et. al., 1999), (Fernández et. al., 1997) la más completa en cuanto a partes del proceso cubiertas y nivel de detalle en la especificación de la metodología.

La evaluación de ontologías (Gómez-Pérez, 2004), (Sure et. al., 2004) es útil tanto durante el desarrollo de las mismas (para comprobar que el conocimiento almacenado es correcto y consistente) como a la hora de plantearse la reutilización

[38] Cuando se mencione el término OWL sin especificar la variante se estará haciendo referencia a la versión OWL-DL, ya que es la que más se utiliza para la descripción de ontologías tanto en la Web como fuera de ella.

[39] El término evaluación de ontologías comprende la verificación y la validación.

de una ontología existente para utilizarla en un problema nuevo o en una nueva ontología que pueda aprovechar parte de ésta.

Las técnicas y metodologías de evaluación están muy relacionadas con los lenguajes de representación del conocimiento en los que se formalizan las ontologías. La mayor parte de los estudios sobre evaluación de ontologías mediante métodos formales se dedica al conocimiento organizado en una jerarquía de especialización (conocimiento taxonómico), mientras que aspectos como el estudio de las relaciones, atributos, axiomas, individuos, etc., está menos desarrollado.

Los formalismos basados en teorías lógicas facilitan el proceso de evaluación en algunos aspectos. Así, las lógicas descriptivas permiten detectar clasificaciones erróneas o no especificadas en la jerarquía de conceptos. De forma general se pueden abordar estudios sobre la satisfacibilidad y consistencia de los conceptos. En el formalismo de los marcos muchas herramientas proporcionan mecanismos para la comprobación de restricciones establecidas en el modelo de conocimiento, aunque las posibilidades son menos amplias que en el caso anterior.

La metodología OntoClean (Guarino y Welty, 2004) proporciona una serie de metapropiedades (rigidez, identidad y unidad) que pueden ser empleadas para comprobar y encontrar conceptualizaciones incorrectas (de forma manual) aunque, una vez más, limitándose al ámbito del estudio de la jerarquía de generalización/especialización de conceptos.

Aparte de las comprobaciones en tiempo de construcción, como las mencionadas anteriormente, hay opiniones respecto a que la evaluación de una ontología debe hacerse midiendo el rendimiento de las aplicaciones que se construyan en base a ella, lo cual es un proceso que presenta numerosas dificultades (Sabou et. al., 2006). También se ha mencionado la posibilidad de crear una comunidad de usuarios en las que las experiencias adquiridas por los mismos al utilizar diferentes ontologías puedan ser una medida evaluativa de las mismas en el proceso de selección de ontologías para su posible reutilización (Lewen et. al., 2006). Este hecho supone la existencia de métodos y estudios para la evaluación de ontologías que van desde los puramente manuales hasta los automatizados.

Además de para el proceso de creación y evaluación, existen metodologías para el proceso de reingeniería, aprendizaje, evolución, mezcla de ontologías, etc. En (Corcho et. al., 2003) se pueden encontrar referencias a estos temas.

3.4 Relación entre la ingeniería del conocimiento y la ingeniería del software.

La ingeniería del conocimiento surge dentro de la disciplina de la inteligencia artificial y, en cierta medida, ha estado enfrentada con la disciplina más práctica (en cuanto a desarrollo de software de uso comercial) de la ingeniería del software. Sin embargo, la finalidad de la ingeniería del conocimiento es la misma que la de la ingeniería del software, esto es, estudiar el proceso de creación de aplicaciones informáticas (denominadas en este campo sistemas basados en el conocimiento) creando lenguajes, herramientas y metodologías para las tareas de diseño, construcción, evaluación, verificación y mantenimiento de esas aplicaciones. La diferencia radica en el tipo de aplicación que se pretende construir y la aproximación que se usa para construirla. Esta diferencia, sin embargo, se está diluyendo en la actualidad.

Desde la época de los sistemas expertos las técnicas de la ingeniería del conocimiento han ido ganando terreno en la realización, completa o parcial, de software comercial. La elección de ingeniería del software o ingeniería del conocimiento a la hora de crear una aplicación es una cuestión que debe decidirse evaluando las características del problema que se pretende resolver. Muchas aplicaciones actuales emplean técnicas y lenguajes que provienen de ambos campos y este hecho es cada vez más patente, de forma que ambas disciplinas intercambian ideas e influyen mutuamente en su respectivo desarrollo (Juristo y Acuña, 2002). Muestra de esta interacción son las iniciativas, planteadas desde el ámbito académico y de investigación, orientadas a la creación de currícula combinados para materias que incorporen ambas disciplinas (Alonso et. al., 1996), (Juristo et. al., 1999).

El encuentro de ambas disciplinas se produce en torno a la creciente importancia que los modelos de conocimiento tienen en el desarrollo del software. Muchas ideas de la ingeniería del software como los patrones de diseño (Gamma et. al., 1995), los componentes, etc., persiguen los mismos fines de reutilización que las ontologías en la ingeniería del conocimiento (Kalfoglou et. al., 2000). El concepto de "arquitecturas basadas en modelos" (Model Driven Architectures – MDA –) (Brown, 2004) es la aproximación de la ingeniería del software al desarrollo de aplicaciones basadas en modelos, postulando la definición del problema a resolver en diferentes niveles de abstracción y usando de forma generalizada modelos de conocimiento de manera que, idealmente, fuese posible partir de una especificación de un modelo independiente de la computación (Computation-Independent Model, CIM) y, pasando por otros modelos intermedios, llegar a producir un código informático para cualquier plataforma, todo ello de forma automática.

Con anterioridad a la aparición del concepto MDA, en la ingeniería del software se venían utilizando modelos en la fase de diseño de las aplicaciones informáticas, siendo el lenguaje UML el estándar de facto para esta tarea. Sin embargo, estos modelos no son equivalentes a los planteados actualmente en MDA, ni a los utilizados en la ingeniería del conocimiento ya que tienen características y objetivos diferentes:

- Los modelos UML están pensados para modelar aplicaciones software, no dominios de conocimiento.
- Los modelos UML son herramientas de la fase de diseño, dejan de tener utilidad al crear la aplicación, es decir, no se usan en tiempo de ejecución.
- Los modelos UML son sobre todo herramientas de comunicación entre ingenieros para comprender y comunicar la estructura de la aplicación.
- Están basados en la orientación a objetos del software, es decir, reflejan no objetos del mundo real sino objetos en el sentido software (es decir, encapsulando propiedades y métodos). No se representan las propiedades que definen el concepto sino aquellas más adecuadas para generar la aplicación informática.

Sin embargo, las ideas que subyacen en las arquitecturas basadas en modelos surgen a partir de la experiencia en UML, así como de la influencia de la ingeniería del conocimiento. Desde la ingeniería del software surge interés por las ontologías (Baclawski et. al., 2002) como estructuras para albergar los modelos de conocimiento definidos en la arquitectura MDA, existiendo un grupo de interés dentro del OMG dedicado a este campo[40] desde 2002. El lenguaje UML, pese a no haberse creado con estos fines, como se ha visto, ha sido también postulado como adecuado para la definición de ontologías (Kogut et. al., 2002), (Cranefield y Purvis, 1999) aunque la falta de una base formalizada con semántica de teorías lógicas ha hecho que se defina ex profeso un metamodelo para la definición de ontologías: el denominado Ontology Definition Metamodel (ODM)[41,42] (OMG, 2007). Otros trabajos toman UML como punto de partida para, mediante la formalización lógica de su metamodelo, crear un sistema de representación del conocimiento completo y funcional (Guizzardi, 2005).

Por su parte, también la comunidad de ingeniería del conocimiento estudia las aplicaciones de las ontologías en el campo del desarrollo del software "tradicional", existiendo en el W3C un grupo de interés sobre ingeniería del software[43]. Diversos estudios relacionan y acercan las ideas sobre estructuras

[40] OMG Ontology Programming Special Interest Group (PSIG) - http://www.omg.org/ontology/

[41] Las ontologías creadas con ODM se pretende que sean compatibles con las representadas mediante lógicas descriptivas (OWL), Common Logic (CL) o Topic Maps (TM) y también con los razonadores utilizados por estas herramientas.

[42] UML sería utilizado como una herramienta para el desarrollo y visualización de las ontologías (definiendo perfiles apropiados para ello), pero su semántica estaría en último término construida según el metamodelo ODM.

[43] Semantic Web Best Practices and Deployment Working Group: Software Engineering Task Force (SETF) - http://www.w3.org/2001/sw/BestPractices/SE/

conceptuales y el concepto de ontología en ambos campos (Gaševic et. al., 2004), (Djuric et. al., 2005), (Cranefield y Pan, 2007).

3.5 Ontologías para representación del conocimiento en el campo de la ingeniería

Dos de las áreas en las que con más asiduidad han sido aplicadas las técnicas de la representación del conocimiento (y de forma más genérica las técnicas de inteligencia artificial) son la biomedicina (biología, genética, medicina,…) y la ingeniería (mecánica, eléctrica, electrónica, de sistemas,…). Las características de estos campos hacen que sean terrenos útiles para comprobar y aplicar las técnicas desarrolladas, ya desde los tiempos de los primeros sistemas expertos. Respecto a la investigación en ontologías se puede afirmar algo similar. Seguidamente se ofrece una visión de algunas de las aplicaciones que las ontologías han tenido en el campo de la ingeniería, observándose cómo este campo de aplicación fue y es importante, no sólo en la aplicación de resultados sino en el curso de la propia investigación en representación del conocimiento.

La aplicación de las ideas de la ingeniería del conocimiento y los sistemas expertos en el ámbito del software para ingeniería de control se produjo desde los primeros sistemas surgidos en los años 1980s. La doble vertiente de existencia de conocimiento complejo y procesos de resolución de problemas que involucran conocimiento humano (heurístico y de difícil explicitación) hace que este campo sea apropiado para la aplicación de estas técnicas.

Algunos de estos estudios (dejando aparte las aplicaciones dedicadas a la implementación de controladores bajo la denominación de "control inteligente") abordaban, en mayor o menor medida y de forma más o menos explícita, el problema de la representación del conocimiento. Entre los proyectos desarrollados se pueden mencionar GE-MEAD (Taylor y Frederick, 1984), (Palumbo y Butz, 1991), SROOT (Muha, 1991), MEDAL (Pang, 1992), ANDECS (Grübel, 1995), etc.

La diferencia principal entre estas experiencias y las que se llevaron a cabo a partir de los años 1990s es que éstas primeras aproximaciones están basadas en el paradigma de la extracción del conocimiento, en el que lo importante es reproducir el comportamiento del experto y no crear un modelo del conocimiento. Los sistemas expertos, construidos con este paradigma, hacen más hincapié en el proceso de reproducción de la resolución de problemas, implementado mediante reglas, que en construir una estructura compleja de representación del conocimiento.

La descripción de los proyectos e ideas sobre la aplicación de las ontologías en el dominio de la ingeniería se presenta, a continuación, dividida en dos etapas. La primera de ellas describe brevemente algunos de los desarrollos realizados desde la adopción del concepto de ontología a principios de los 1990s, estando caracterizada por las investigaciones en el ámbito académico. La segunda etapa arranca al principio de los años 2000s y se caracteriza por el paso a una aplicación más práctica de las investigaciones de la década precedente.

3.5.1 Aplicaciones tempranas de las ontologías en el ámbito de la ingeniería (años 1990s)

Con la llegada del paradigma del modelado del conocimiento y la aparición del concepto de ontología, surgió una nueva oportunidad de aplicación de estas técnicas al dominio de la ingeniería. Al igual que con los sistemas expertos, esta aplicación se produjo desde los propios trabajos sobre ontologías de principios de los 1990s. Muchos de los trabajos de Gruber[44] sobre ontologías están intimamente relacionados con el dominio de la ingeniería. El proyecto "How Things Work" (Fikes et. al., 1991) sirvió de base para la propia introducción del concepto ontología, así como para las primeras definiciones y ejemplos de este término, que se llevaron a cabo en base a este proyecto (Gruber e Iwasaki, 1990). El proyecto tenía como objetivo la descripción de modelos de sistemas físicos por medio de formalismos que permitiesen implementar simulaciones y ofrecer explicaciones sobre la causalidad de su comportamiento. Es relevante destacar también que el proyecto estaba pensado como un complemento al desarrollo del estándar PDES/STEP de ISO.

También relacionada con el proyecto PDES/STEP surge la iniciativa PACT (Cutkosky et. al., 1993) (Palo Alto Collaborative Testbed), un proyecto que pretendía abordar las aplicaciones de ingeniería concurrente con técnicas de sistemas multiagente, representación del conocimiento y ontologías. La finalidad era integrar las diferentes actividades de los ingenieros que participan en el desarrollo de un producto (ingeniero de sistemas, mecánico, electrónico, de control, etc.) mediante la integración de las aplicaciones informáticas que suelen construirse de forma aislada y sin posibilidad de interacción entre ellas. El elemento de unión entre las diferentes aplicaciones serían los modelos de conocimiento expresados en ontologías. En este caso se utilizaba el lenguaje KIF y el protocolo de comunicación entre agentes KQML[45] (Knowledge Query and

[44] El investigador que generalizó y popularizó el uso del término ontología.

[45] KQML fue desarrollado como parte del consorcio DARPA Knowledge Sharing Effort (KSE) (Patil et. al., 1992), que también dio lugar, entre otros, al lenguaje KIF, Ontolingua y al propio sistema PACT - http://ksl-web.stanford.edu/knowledge-sharing/index.html

Manipulation Language) (Finin et. al., 1992)[46]. La mayor aportación del proyecto fue el proceso colaborativo de creación de los modelos de conocimiento. La representación contaba también con una conceptualización de diferentes conceptos temporales útiles para representar las etapas del proceso de diseño.

EngMath (Gruber y Olsen, 1994), una de las primeras ontologías que más difusión tuvo, se dedicó a conceptualizar el dominio de matemáticas en ingeniería, utilizándose además como ejemplo de ontología en uno de los primeros artículos seminales sobre el tema (Gruber, 1995). Esta ontología describe conceptos relacionados con las magnitudes y unidades físicas así como los conceptos "escalar", "vector", etc. El desarrollo de la ontología se realizó mediante Ontolingua, siendo también una de las primeras realizaciones en este lenguaje.

Los proyectos KACTUS (modelling Knowledge About Complex Technical systems for multiple USe) y OLMECO (Open Library for Models of MEchatronical COmponents) (Top y Akkermans, 1994), (Borst, 1997) fueron la base para una serie de investigaciones en cuanto a aplicaciones de las ontologías en el campo de la ingeniería de sistemas. Dentro de este proyecto se estudió la forma de construir una librería de ontologías para describir sistemas físicos tecnológicos y de usar esa librería para la construcción de bases de conocimiento sobre el tema. Algunas de las aportaciones de estos proyectos son la ontología PhysSys (Borst et. al., 1997), dedicada a la descripción de sistemas físicos a diferentes niveles. Se desarrollaron ontologías independientes sobre mereología (representación de las partes y el todo), topología (representación de la relación de interconexión entre las partes que permite el intercambio de energía entre ellas) y teoría de sistemas, así como ontologías dedicadas a representar las vistas posibles de un sistema físico: ontología sobre la vista de proceso (de la que se extrae el comportamiento), de componentes (de la que se extrae la distribución y conexión de componentes) y de relaciones matemáticas (que describen el modelo de ecuaciones diferenciales del sistema). Para esta última ontología se reutilizó la ontología EngMath (siendo uno de los primeros ejemplos prácticos de reutilización de estas estructuras de conocimiento). PhysSys ha sido formalizada en diferentes lenguajes, en (Borst, 1997) se recoge la formalización en Ontolingua versión 4.0. Por su parte OLMECO (Breunese et. al., 1997) es una librería de componentes para la construcción de diseños de sistemas físicos, organizada en forma de objetos. En conjunto estos proyectos supusieron una experiencia en la creación de ontologías aisladas y en la integración de las mismas en un sistema total, también supuso una experiencia en cuanto a reutilización, ya que se reutilizó EngMath, y de integración y modificación de una ontología existente en el caso de la dedicada a mereología. En cuanto a la ingeniería de sistemas, supuso una aportación en el área de modelado basado en el conocimiento de sistemas mecatrónicos descritos mediante grafos de ligaduras (bond graphs).

[46] La cita se refiere a la versión de KQML utilizada en el proyecto PACT

YMIR (Alberts, 1994) es la primera ontología dedicada exclusivamente a la representación de los conceptos propios de la ingeniería de sistemas. Esta ontología está orientada a la representación de los elementos utilizados en el proceso de diseño de sistemas y según (Alberts y Dikker, 1994) puede considerarse complementaria de los esfuerzos realizados en PDES/STEP y EXPRESS.

Una de las primeras aplicaciones del formalismo de las lógicas descriptivas a la definición de ontologías vino también de un campo relacionado con la ingeniería. La ontología Plinius (van der Vet et. al., 1994) se encuadra dentro del proyecto del mismo nombre cuya finalidad es la extracción semiautomática de conocimiento de textos cortos de carácter científico (van der Vet et. al., 1995). Estos textos consisten en resúmenes de artículos científicos en el área de ciencia de los materiales. El trabajo incluye además uno de los primeros estudios sistemáticos del problema de la computabilidad y la complejidad computacional de diferentes versiones de lógicas descriptivas (Speel, 1995). También constituye el ejemplo más paradigmático del proceso de construcción de ontologías "de abajo hacia arriba" (bottom-up) (van der Vet y Mars, 1998).

3.5.2 Aplicaciones contemporáneas (años 2000s)

A partir de comienzos del siglo XXI se produjo un creciente interés por la aplicación de las ontologías en el diseño y desarrollo de aplicaciones informáticas que, en el caso de los últimos años (a partir de 2004-2005) ha sido exponencial. En el campo de la ingeniería el uso de modelos de conocimiento representados en ontologías se aplica sobre todo a la tarea de describir sistemas complejos desde el punto de vista de su descripción física (partes, componentes, forma de conectarse, etc.) y funcional (comportamiento de los componentes, función de los mismos en el sistema). Estas descripciones se usan para aplicaciones de simulación y, sobre todo, diseño de sistemas (Kitamura y Mizoguchi, 2003), (Kitamura, 2006).

La industria aeronáutica es una de las primeras áreas donde la representación del conocimiento con ontologías es ampliamente explorada y utilizada. Algunos ejemplos que se pueden citar son (Valente et. al., 1999) o (Reiss et. al., 1999), aplicaciones dedicadas al campo de la planificación de campañas aéreas. Pero los ejemplos más paradigmáticos en esta área los constituyen el PGEF (Pan Galactic Engineering Framework) (Waterbury, 2007) (dedicada al campo del diseño de sistemas aeronáuticos en la NASA) y el conjunto de ontologías desarrolladas para la NASA descrito en (Hodgson y Coyne, 2006). El uso de las ontologías en estos casos permite abordar la elevada complejidad de los sistemas y ofrecer integración entre aplicaciones dedicadas a diferentes cometidos.

El tratamiento y representación de la complejidad es también la justificación para la aplicación de las ontologías en otros ámbitos de la industria como la fabricación de vehículos (Lukibanov, 2005), (Morgan et. al., 2005). Dentro de las ciencias de la tierra puede citarse el proyecto y ontología SWEET (The Semantic Web for Earth and Environmental Terminology), desarrollada dentro de la NASA, que incluye ontologías para describir el espacio, el tiempo, procesos y propiedades físicas, etc. (NASA, 2006).

El proyecto STEP es también, en esta década, uno de los campos de aplicación de las ontologías y tecnologías relacionadas. Después de que surgiesen críticas a la falta de claridad en los conceptos definidos por el estándar STEP se propuso la idea de ofrecer una base formal, basada en ontologías, para el mismo (Guarino et. al., 1997) y utilizar lenguajes dedicados a la formalización de ontologías en lugar del lenguaje EXPRESS[47] para describir los modelos de conocimiento (Price et. al., 2006), (Schevers y Drogemuller, 2005). El proyecto PGEF es un también un buen ejemplo de plataforma de modelado de sistemas basado en los estándares y protocolos STEP al que recientemente se han incorporado las ontologías y lenguajes como OWL (Waterbury, 2007). Entre los objetivos de este proyecto se cita "integrar los datos, herramientas y capacidades de STEP, UML, y OWL". Como se puede comprobar, se intenta integrar los aspectos de modelado de STEP/EXPRESS, el uso de ontologías con OWL y las técnicas de modelado del campo de la ingeniería del software incluyendo UML y SysML, así como las ideas provenientes de las arquitecturas basadas en modelos (MDA).

Otras aplicaciones que pueden citarse sobre ontologías relacionadas con el campo de la ingeniería de sistemas y automática son las desarrolladas para la integración de modelos de datos y de computación en un sistema integrado de automatización (Hu et. al., 2003), la representación de entornos para robots móviles (Chella et. al., 2002), (Schlenoff y Uschold, 2004), la representación de datos en ingeniería de procesos (Morbach et. al., 2007) (OntoCAPE), la gestión de la cadena de suministro (supply chain management) (Chandra y Tumanyan, 2007), la representación de los procesos en ingeniería de fabricación (manufacturing systems engineering) (Lin y Harding, 2007), la verificación de sistemas de control empotrados (Kumar y Krogh, 2006), la representación del producto incluido en su

[47] Puede decirse que la preocupación por la creación de modelos formales de información es anterior en el campo de la ingeniería de sistemas que en la del software, siendo el lenguaje EXPRESS una prueba de esto. El desarrollo de este lenguaje comenzó en los años 1980s persiguiendo la representación de conocimiento complejo en ingeniería. Las características y construcción de EXPRESS hacen que incluso pueda ser considerado un lenguaje para la especificación de ontologías, aunque con algunas limitaciones. EXPRESS fue utilizado como lenguaje para especificar las estructuras de los protocolos de aplicación de ISO STEP, aunque nunca tuvo gran implantación de debido a que no logró la suficiente popularidad entre los usuarios. La expansión de UML es debida precisamente a su amplia aceptación en la industria del software de tal forma que, como se ha visto, se ha utilizado en ingeniería a pesar de ser menos potente y expresivo que EXPRESS. Algunas propuestas recientes llegan a proponer la utilización de UML en vez de EXPRESS para definir los estándares de ISO STEP (Lubell et. al., 2004) pero UML no tiene la capacidad y flexibilidad de EXPRESS.

línea temporal (Batres et. al., 2007), la representación de esquemas como las redes de Petri (Gaševic y Devedžić, 2006), etc.

Por último citar (Pop y Fritzson, 2004), donde se investigan las posibilidades de utilizar OWL como lenguaje de modelado para MODELICA, con el objetivo de aprovechar las mayores posibilidades de expresividad que ofrecen las ontologías (sobre otros esquemas de representación del conocimiento) para la descripción y reutilización de modelos de sistemas físicos.

El estudio presentado en esta sección sirve también para completar la exposición de la sección 2.2 sobre la evolución de la representación de la información en el software CACE.

3.6 Discusión y conclusiones parciales

La idea de modelado del conocimiento está actualmente presente en todos los campos que estudian la creación de software. Si bien este concepto apareció claramente definido por primera vez en el ámbito de la inteligencia artificial, también en la ingeniería del software se han producido aproximaciones con ideas similares en los últimos años, creadas a partir de los métodos de modelado de software utilizados en la fase de diseño de las aplicaciones informáticas.

La tendencia hacia el modelado del conocimiento en las aplicaciones informáticas se observa con claridad también en el campo de aplicación de la ingeniería de sistemas y control. La evolución del software CACSD vista en el capítulo anterior junto las investigaciones mencionadas en la sección 3.5.2 permiten concluir que el uso de las ontologías puede jugar un papel importante en el desarrollo de software para estas disciplinas.

Las investigaciones en ingeniería del conocimiento, junto con las de la ingeniería del software y las propias iniciativas de la ingeniería de sistemas, están dando paso a nuevas ideas y conceptos como la Model-Driven Systems Engineering (MDSE) (Ingeniería de Sistemas Dirigida por Modelos). En estos nuevos paradigmas de la ingeniería de sistemas se produce una mezcla de técnicas de la ingeniería del software (UML 2.0, la orientación a objetos, los componentes), ingeniería del conocimiento (ontologías, lenguajes y formalismos de representación) y de la propia ingeniería de sistemas (SysML, STEP/EXPRESS, métodos formales) que colaboran para construir el software del futuro (dos Santos y Vrancken, 2007). En este sentido es interesante comprobar cómo las organizaciones internacionales que se encargan de la estandarización de tecnologías relacionadas con el desarrollo del software ó la ingeniería de sistemas se dedican, mediante diferentes grupos de trabajo, al estudio y proposición de estándares para la representación de modelos de conocimiento: así, está la OMG

con MDA, ODM, UML 2, SysML…, el W3C con RDF, RDF(s), OWL y la ISO com CL, STEP/EXPRESS, etc.

Habiendo llegado a la conclusión de la importancia del modelado del conocimiento en la creación de software para ingeniería de control, se presenta a continuación una serie de comentarios sobre cómo se puede abordar este paradigma desde las técnicas de la ingeniería del conocimiento, teniendo en cuenta lo expuesto en el presente capítulo. El objetivo de la siguiente discusión es ofrecer una base para decidir cómo se llevará a cabo el desarrollo de la ontología que se pretende construir. Esta decisión se presentará en el próximo capítulo.

Una ontología es, en principio, una estructura conceptual independiente de cualquier agente o aplicación que la pueda utilizar y también del formalismo que se utilice para implementarla en un ordenador. Sin embargo, estas afirmaciones son, en la práctica, difíciles de mantener, ya que al formalizar la ontología mediante algún formalismo (cuando se quiera expresar en un ordenador) se perderá parte de la expresividad de la misma, debido a las limitaciones impuestas por los problemas de computabilidad y complejidad computacional de los lenguajes de representación del conocimiento. De este modo, en la práctica, el desarrollo de una ontología se realiza teniendo muy presente desde un primer momento el formalismo y las herramientas que serán utilizadas para su construcción en el ordenador.

Por otro lado, aunque OWL, como se ha visto, es el lenguaje estándar creado por el W3C para la definición de ontologías en el ámbito de Internet, no es la única alternativa para elegir un formalismo de representación del conocimiento, presentando algunas características que lo pueden hacer inapropiado en ciertas ocasiones. La idea inicial de los marcos pervive todavía[48] encontrando líneas de colaboración y similitudes en el campo de la ingeniería del software (el lenguaje UML puede verse, a pesar de que su origen y uso preferente es para modelado de software, como una herramienta dedicada a la representación del conocimiento mediante un formalismo similar al de los marcos). La elección entre uno u otro formalismo (basado en marcos o en lógicas descriptivas)[49] depende en gran parte del dominio que se está estudiando, sus características (en cuanto a tipos de conocimiento que será necesario representar) y el tipo de uso que se hará de la ontología construida.

48 Actualmente estas son las dos elecciones posibles que, en la práctica, suelen ser utilizadas como formalismo para representar una ontología: OWL o algún formalismo basado en marcos. Otros formalismos y lenguajes pueden ser utilizados para representar ciertos tipos de conocimiento, pero no ontologías completas.

49 La categorización de los formalismos de representación del conocimiento es un asunto complejo. Los formalismos basados en lógicas descriptivas, herederos de las redes de herencia estructural, tienen parte de sus raíces en la idea de marcos como se ha visto y a menudo a estos formalismos se les ha denominado incluso como "basados en marcos". Hoy en día se suele hacer la distinción aquí mencionada, entre aproximaciones basadas en lógicas descriptivas y aproximaciones basadas en marcos. Otra denominación para las primeras es "basadas en el paradigma relacional", mientras que a las segundas se las suele denominar "basadas en el paradigma de orientación a objetos" (Wang et. al., 2006) aunque, una vez más, surge la paradoja de que la idea de marco es anterior a la de orientación a objetos, influyendo de hecho aquélla en ésta.

A diferencia de lo que ocurrió con las lógicas descriptivas y OWL (gracias a la existencia de un organismo como el W3C y a la necesidad de un estándar para la Web Semántica), para el formalismo de marcos no existe un lenguaje estandarizado. El protocolo OKBC (Open Knowledge-Base Connectivity) suele utilizarse como guía para implementar las herramientas de desarrollo y adquisición de conocimiento basadas en marcos, pero no cuenta con semántica basada en teoría lógica de modelos (aunque sí con una axiomatización descrita por medio de KIF).

En (Wang et. al., 2006) se comparan los formalismos basados en lógicas descriptivas (las comparaciones se realizan en base a OWL-DL) y en marcos (haciendo referencia a la implementación de OKBC presente en la herramienta Protégé), presentando sus similitudes, diferencias y los casos en los que uno es más apropiado que el otro. Las ideas más importantes sobre las que se establecen las diferencias relevantes entre ambos formalismos son[50]:

- Grado de composición en la creación de conceptos. Si los conceptos del dominio pueden definirse a partir de otros el uso de un sistema basado en lógicas descriptivas puede resultar muy ventajoso ya que los sistemas basados en marcos no permiten la definición de conceptos sino sólo su descripción[51].
- Complejidad de la jerarquía de conceptos. El uso de razonadores basados en lógicas descriptivas permite normalizar las ontologías, es decir, mantener éstas formadas por múltiples árboles no solapados. Esto es una ventaja en jerarquías de conceptos grandes y complejas, donde la creación y mantenimiento manual es prácticamente imposible. La posibilidad de definir unos conceptos a partir de otros, tal como se ha mencionado en el punto anterior, permite que OWL sea capaz de efectuar clasificación automática de conceptos, dentro de esta jerarquía, a partir de su definición.
- Naturaleza de pre-coordinación o post-coordinación en el uso de la ontología ((Rector, 2004), (Rector, 2005)). La pre-coordinación hace referencia al uso en tiempo de ejecución de la ontología construida, sus conceptos y relaciones, pero sin variar su estructura. El uso de formalismos basados en lógicas descriptivas en este caso sería útil en tiempo de diseño (bajo alguna de las condiciones expresadas anteriormente) pero no en tiempo de ejecución. Por el contrario, si la

[50] En el artículo se citan otras diferencias entre ambos formalismos, aquí sólo se presentan las más relevantes a la hora de elegir la aproximación más adecuada a las particularidades del conocimiento existente en el dominio de estudio en esta tesis.

[51] La definición de un concepto en lógicas descriptivas es la explicitación de las propiedades necesarias y suficientes para que un individuo pertenezca a ese concepto, es una representación de tipo intensional (intensional) del concepto. En la aproximación basada en marcos los conceptos se describen, es decir, se enumera el conjunto de propiedades (slots) que tienen los individuos que pertenecen a ese concepto. Para hacer esta enumeración se tiene en mente el conjunto de todos los elementos que componen el concepto, es una representación de tipo extensional.

aplicación requiere que los conceptos no puedan ser fijados de antemano sino que es posible que sean definidos en tiempo de ejecución (post-coordinación), el uso de razonadores basados en lógicas descriptivas sí es rentable.

- Hipótesis de mundo abierto o mundo cerrado[52]. Reiter (Reiter, 1977) distinguió dos categorías de bases de datos (que también pueden ser extendidas al caso de bases de conocimiento):
    - Mundos cerrados. Es aquella base de datos en la que toda proposición verdadera está almacenada o es deducida de otras que lo están. Cualquier proposición que no está almacenada o que no es demostrable se considera falsa[53].
    - Mundos abiertos. Una base de datos en la que algunos hechos acerca del dominio son desconocidos o no se pueden probar se denomina un mundo abierto. En un mundo abierto algunas proposiciones se sabe que son verdaderas, y algunas se sabe que son falsas, pero hay una gran área intermedia de proposiciones cuyo valor de verdad es desconocido.

  Los esquemas de representación basados en marcos se basan sobre todo en la hipótesis de mundo cerrado mientras que en OWL se asume la hipótesis de mundo abierto. Esto supone una diferencia drástica: para la aproximación basada en marcos, lo que existe es lo que explícitamente está representado o lo que se puede deducir de lo representado, mientras que para OWL pueden existir más elementos que los representados[54]: que algo no pueda ser deducido no quiere decir que no sea verdadero.

Otro de los temas sobre los que suele debatirse es el grado de formalidad a utilizar a la hora de definir una ontología[55] (respecto a aspectos prácticos de implementación de aplicaciones). Mientras que algunos investigadores postulan que la formalidad debe proceder de la especificación de la semántica mediante teoría lógica de modelos (por ejemplo, ver el artículo de Enrico Franconi en (Brewster y O'Hara, 2004)), otros admiten la existencia de representaciones con

[52] Open World Assumption - OWA - vs. Closed World Assumption - CWA -

[53] Este método de razonamiento se denomina "negación como fallo" (Negation As Failure - NAF -). En contraste, en la hipótesis de mundo abierto el razonamiento equivalente es "negación como insatisfacibilidad" (Negation As Unsatisfiability).

[54] Para indicar que todo lo representado es lo que realmente existe y nada más que eso en un lenguaje que utiliza la hipótesis del mundo abierto deben utilizarse los denominados axiomas de clausura (closure axioms).

[55] La discusión sobre la necesidad de formalidad en los formalismos de representación del conocimiento se remonta a los trabajos de Minsky, donde se defiende la idea de informalidad que subyace en los marcos, apoyándose en varias razones: 1) El conocimiento está más concentrado y localizado cuando se usan los marcos que si se distribuye a lo largo de diferentes axiomas. 2) Los mecanismos de deducción no tienen un foco de interés en el caso de formalismos lógicos. El existir superestructuras de conocimiento ayuda a seguir la secuencia de deducción y el control del mecanismo de deducción. 3) La lógica es monotónica. 4) La consistencia es un hecho incluso no deseable en una base de conocimiento, el ser humano se las apaña muy bien sin la consistencia.

mayor o menor grado de formalidad en la construcción de ontologías[56] (Mizoguchi e Ikeda, 1996), (Gruber, 2004).

La ventaja de un formalismo lógico estricto (como el utilizado en las lógicas descriptivas y OWL) es la posibilidad de automatizar las tareas de evaluación, comparación, reutilización, etc., de las ontologías, así como poder estudiar los aspectos relacionados con la computabilidad y complejidad computacional de los algoritmos de razonamiento automático que se puedan implementar. La desventaja radica en que, al aumentar la expresividad de estos lenguajes, la tratabilidad de los algoritmos implementados puede verse comprometida. Estas aproximaciones tienen su capacidad expresiva restringida a la permitida por la lógica utilizada y por tanto suelen presentar también problemas (entre otros) de monotonicidad e imposibilidad para representar contextualizaciones.

Las aproximaciones basadas en estructuras complejas (marcos) tienen la desventaja de su más difícil estudio y comprensión teóricos, pero la ventaja de su mayor flexibilidad, posibilidades y facilidades de uso. El razonamiento automático al estilo de las aproximaciones basadas en lógica no es posible con la aproximación de los marcos, aunque este hecho no supone un problema para crear aplicaciones prácticas en dominios controlados, tal como ocurre por ejemplo con todas las técnicas y herramientas de modelado basadas en UML o en la mayoría de los sistemas que utilizan bases de datos. También es muy probable que algunas de las estructuras semánticas construidas mediante este formalismo de marcos necesiten de procesadores "ad-hoc" construidos en el lenguaje de programación. La aceptación de las estructuras de conocimiento más útiles junto con sus motores de procesamiento podría hacer que éstas se convirtiesen en patrones comúnmente adoptados al estilo de los encontrados en el diseño de software. De esta forma se produciría la reutilización en el caso de las estructuras basadas marcos, mientras que en las aproximaciones basadas en lógica la semántica está autocontenida en el sistema de representación[57].

Teniendo en cuenta las consideraciones realizadas en este capítulo y las características del dominio de la ingeniería de control a modelar se elegirá un tipo de formalismo u otro para realizar la ontología en este campo. Estas decisiones y su justificación aparecen en el siguiente capítulo.

[56] Esta discusión sobre el grado de formalidad es también culpable en parte de la dificultad de ofrecer una definición universalmente aceptada para el término ontología.

[57] Para una discusión más amplia sobre el significado del término “semántica” y las formas de construirla en las aplicaciones informáticas (y en particular en la Web Semántica) ver (Uschold, 2003)

Capítulo

4

Esquema de representación propuesto

Gordie: Mickey es un ratón, Donald es un pato, Pluto es un perro. Pero ¿Qué es Goofy…?
Teddy: Es un perro, definitivamente es un perro…
Chris: No puede ser un perro, lleva sombrero y conduce un coche…
Vern: Sí, eso es extraño. ¿Qué demonios será Goofy?

Cuenta conmigo (Stand by me), 1986

En el presente capítulo se presenta la ontología desarrollada, que recoge la conceptualización de un subdominio del control automático. En primer lugar se elegirá un subdominio dentro de la ingeniería de control y, a continuación, el formalismo de representación a utilizar, tomando como base las características del conocimiento existente en el subdominio elegido. Seguidamente se procede a la identifican las estructuras de conocimiento existentes y su posible expresión por medio del formalismo elegido, lo que conduce a la construcción incremental de la ontología. Por último se presenta también la aplicación informática construida a partir de la ontología creada.

4.1 Introducción

La creación de una ontología es un proceso complejo y trabajoso. La evolución de las metodologías y formalismos, así como la construcción de ontologías dedicadas a diversos campos del saber permitirán, en el futuro, que este proceso de creación se vea facilitado al poder reutilizar ontologías existentes o contar con experiencias de conceptualización similares a las abordadas. En la actualidad, y en el dominio que se está tratando en particular, esta situación todavía no se ha producido.

Debido al carácter experimental del trabajo que se lleva a cabo existe una total libertad sobre el subdominio[58] de conocimiento a partir del cual se creará la ontología propuesta. La elección de este subdominio será la primera tarea a realizar ya que una adecuada elección del mismo permitirá poner de manifiesto de forma más adecuada las bondades y ventajas del modelado del conocimiento en el campo del software para ingeniería de control.

Una vez elegido el subdominio de conocimiento el siguiente paso será la elección del formalismo de representación con el que reflejar la ontología en el ordenador. Las diferentes capacidades expresivas de los lenguajes y formalismos hacen que este sea un punto importante en el proceso desde el punto de vista práctico, aunque idealmente la ontología debería desarrollarse en el "nivel del conocimiento" de forma independiente de la formalización posterior. La decisión sobre el formalismo a emplear está condicionada por el tipo de conocimiento existente en el dominio a modelar, por lo que será necesario realizar un estudio inicial del mismo.

La última etapa del proceso será el desarrollo práctico de la ontología. En esta etapa se localizan las estructuras conceptuales y se buscan modelos de conocimiento que las representen en el formalismo elegido. Durante este proceso se puede considerar la reutilización total o parcial de otras ontologías existentes dedicadas a dominios similares.

Este capítulo está organizado del siguiente modo: En primer lugar se justificará la elección de un subdominio de la ingeniería de control sobre el que trabajar y crear un modelo de conocimiento. A continuación, en la sección 4.3, se introducirán las características generales del conocimiento en ingeniería de control y las de ese subdominio en particular. Este estudio será la base para la elección de un formalismo de representación mediante el cual construir la ontología. La sección 4.5 está dedicada a la descripción de las estructuras conceptuales encontradas y reflejadas mediante ese formalismo. Finalmente, en la sección 4.6 se ofrece una discusión y conclusiones parciales.

4.2 Elección de un subdominio de estudio

Aunque el desarrollo de un modelo de conocimiento puede ser llevado a cabo sobre cualquier domino, se ha intentado encontrar uno que presente una serie de características que le hagan especialmente adecuado para ser representado mediante técnicas de ingeniería del conocimiento y que permita comprobar las

[58] En tanto en cuanto se va a acotar una parte del dominio del control automático se habla de subdominio de estudio, aunque cuando ya se haya elegido definitivamente el mismo, éste se convierte en el dominio de estudio propiamente dicho.

ventajas de esta aproximación. Estas características pueden resumirse a grandes rasgos en:

- Complejidad en las estructuras de conocimiento, es decir, que presente un conjunto de conceptos con elevado nivel de interrelación.
- Presencia de conocimientos de diferente naturaleza.
- Elevada componente de razonamiento humano en la resolución de problemas dentro del dominio.

En definitiva, se puede decir que se pretende escoger un subdominio con "elevado contenido semántico".

Otra de las características buscadas a la hora de elegir el subdominio es que éste sea ampliamente conocido y estudiado en el campo del control automático. De esta forma el número de documentos, artículos y libros que traten el tema será mayor y por tanto habrá más fuentes para realizar la conceptualización, y también más puntos de vista diferentes que pueden ser útiles a la hora de construirla.

Con todas estas premisas el posible subdominio de estudio puede estar recogido en alguno de los siguientes campos:

- Análisis y diseño básico de compensadores con técnicas de la denominada teoría clásica de control (en el dominio de la frecuencia compleja).
- Análisis y diseño básico de compensadores con técnicas de la denominada teoría moderna de control (en el dominio del tiempo).

Un estudio comparativo del conocimiento en ambos subdominios permite concluir que la teoría clásica tiene mucho más contenido semántico y se ajusta mejor a las características requeridas anteriormente. De las técnicas relacionadas con la teoría clásica de control se dice que son de diseño (y análisis) puro, mientras que las de la teoría moderna se basan más en la síntesis directa. Las primeras presentan un proceso iterativo en el diseño, partiendo de una solución dada por la observación de diferentes gráficas que, a la luz de los resultados o simulaciones obtenidas, es modificada las veces que sean necesarias hasta alcanzar los objetivos de funcionamiento. Las segundas en cambio se basan en cálculos más analíticos de la solución (el controlador) a partir del modelo del proceso o planta y las especificaciones de diseño (Dutton et. al., 1997).

Dentro de la teoría clásica a su vez se pueden realizar la siguiente división para delimitar aún más el subdominio:

- Métodos de análisis y diseño de compensadores con técnicas basadas en la respuesta en el tiempo (lugar de las raíces).

- Métodos de análisis y diseño de compensadores con técnicas basadas en la respuesta en frecuencia (Bode, Nyquist, etc.).

Una vez más, se tomó una decisión teniendo en cuenta el contenido semántico de ambas posibilidades, llegando a la conclusión de que los métodos y técnicas relacionadas con el lugar de las raíces eran los más adecuados para ser estudiados. Esta decisión se basa en una serie de hechos:

- Los parámetros relacionados con la respuesta en el tiempo son más intuitivos para el diseñador de compensadores, es decir, contienen más semántica (más significado) para el ser humano.
- El método relaciona las posiciones de las raíces de los polinomios que forman los modelos matemáticos en función de transferencia, con las respuestas en el tiempo de los sistemas correspondientes. Hay por tanto conocimiento de diferente naturaleza y muy interrelacionado.
- El método de diseño involucra decisiones heurísticas, es decir, hay presencia de lo que se denomina "conocimiento experto".
- Desde el punto de vista didáctico estos métodos permiten obtener una comprensión profunda de importantes conceptos de control.

De esta forma, pese a estar superado en la práctica habitual de diseño en control por muchos otros métodos, el lugar de las raíces se presenta como el que más se adecua a las características introducidas al principio requeridas para el subdominio de estudio. De hecho por similares razones a las expuestas estas técnicas siguen siendo empleadas en la docencia de la materia.

Por último, con el fin de centrar el conocimiento del subdominio de estudio se han acotado las posibilidades de acuerdo a las siguientes consideraciones:

- Sólo se tratarán sistemas lineales o previamente linealizados. La conceptualización tendrá como base de partida la existencia de una función de transferencia del sistema a controlar.

- Se considerarán sistemas invariantes, es decir, cuyos parámetros del modelo en función de transferencia no cambian con el tiempo.

- Se considerarán sistemas de una entrada y una salida (SISO - Single Input, Single Output).

- Se estudiará la conceptualización para el análisis y diseño de compensadores para seguimiento de consigna. Aunque en la práctica las perturbaciones externas son de gran importancia y ocupan un lugar principal en el diseño de compensadores, el diseño para seguimiento de consigna permite poner de manifiesto de forma más directa la relación

entre la configuración del controlador elegido y la respuesta del sistema ante una entrada escalón. No se considerarán por tanto presencia de ruidos en el sensor ni perturbaciones en la carga ni en la entrada al sistema

- Se considerará un solo tipo de señal de test en la entrada: el escalón unitario. La respuesta de un sistema ante una entrada en escalón unitario es la que permite obtener mayor número de parámetros relevantes de rendimiento y con un mayor significado intuitivo para el usuario.

- La configuración o colocación del controlador se supondrá siempre en serie con la planta o proceso a controlar. La configuración o posición del controlador en el bucle de regulación es uno de los parámetros de diseño que habitualmente se tiene en cuenta a la hora de solucionar un problema de control. Sin embargo, para mantener acotada la complejidad del dominio, se partirá de una configuración del controlador en serie con el sistema a controlar.

- La realimentación será negativa y unitaria. La mayoría de los métodos de análisis y diseño basados en el lugar de las raíces encontrados en los libros de texto consideran este tipo de realimentación.

- El control será siempre de acción directa, es decir, un incremento en la señal consigna debe resultar en un incremento en la señal de salida.

- Los tipos de controlador que podrán existir serán: proporcional, adelanto de fase, retraso de fase y adelanto-retraso de fase. Además, el controlador constará de una sola etapa, es decir, sin posibilidad de configurar varios controladores en cascada. También se considerará que todos los polos y ceros del controlador serán números reales.

- Se considerarán sistemas sin retardo.

- Se supondrá la existencia de un área de diseño, es decir, las especificaciones se darán en forma de inecuaciones respecto al parámetro de rendimiento considerado.

La conceptualización del dominio así acotado se ha dividido en dos partes. La ontología presentada en este capítulo aborda la conceptualización del conocimiento necesario para reflejar la estructura estática de conceptos y relaciones existentes en el dominio mencionado. No se realizará una conceptualización del propio proceso de diseño de compensadores, tarea ésta que se está llevando a cabo en otra tesis doctoral.

A continuación se realiza un estudio de la naturaleza y características del conocimiento en el dominio elegido como paso previo para la elección del formalismo de representación y como base para comenzar a construir las estructuras de conocimiento.

4.3 El conocimiento en la tarea de análisis y diseño en el dominio de la frecuencia compleja

4.3.1 Características del conocimiento en ingeniería de control

El control automático es una disciplina científico-técnica y como tal comparte ciertas características con otras áreas de la ingeniería:

- La ingeniería es una actividad humana y por lo tanto la resolución de problemas en este ámbito conlleva conocimiento humano, muchas veces heurístico e incompleto.
- La ingeniería trata sobre la construcción de sistemas físicos, es decir, el principio y el resultado de un proceso de ingeniería es siempre un artefacto o sistema real.
- El estudio de los sistemas físicos mencionados se realiza mediante modelos (simplificaciones del sistema original), normalmente de naturaleza matemática. Además, habitualmente existe un proceso de simplificación en el que se generan diferentes modelos del mismo sistema a diferentes niveles de representación y sobre ellos se utilizan diferentes tipos de razonamiento.
- Las herramientas y técnicas utilizadas en la resolución de problemas son variadas: desde manipulaciones y razonamientos matemáticos hasta el uso de gráficas y razonamientos espaciales.
- Las técnicas empleadas en las áreas de análisis y diseño normalmente dan como resultado un proceso iterativo que implica rediseño, es decir, la repetición del proceso de diseño a la luz de resultados previos.

Por su parte, el control automático tiene una serie de particularidades que lo distinguen de otras disciplinas similares y que son el resultado de su propia naturaleza. La ingeniería de control es una disciplina horizontal, lo que supone que aparece aplicada a muchas áreas diferentes, tanto científicas como técnicas. De hecho, el nacimiento de la disciplina se produjo en los años 1930 de forma independiente en las áreas de electrónica, control de procesos químicos, economía

y homeostasis[59] (West, 1992) (traducción literal de la frase y referencia tomadas de (Bissell, 1993)).

En los años inmediatamente posteriores a la Segunda Guerra Mundial la disciplina emergió como un campo independiente de forma lenta y progresiva, estando su desarrollo basado fundamentalmente en la creación y evolución de un lenguaje propio (Bissell, 1993). El proceso de constitución del control como disciplina independiente no consistió en el descubrimiento de nuevos fenómenos físicos o la invención de nuevos dispositivos, ni se basó en ningún cambio en las creencias sobre el mundo sino que encontró usos nuevos y nombres nuevos para fenómenos ya descritos. Suele decirse que la matemática es el lenguaje de la ingeniería[60]. Sin embargo, el lenguaje de la matemática por sí solo no sirve para transmitir los conceptos relevantes en ingeniería de control ya que éstos recogen abstracciones de alto nivel e interpretaciones de aquellos en ciertos escenarios y por lo tanto poseen un contenido semántico añadido que es el esencial para que la disciplina pueda ser transmitida y utilizada[61].

4.3.1.1 Características del conocimiento en la teoría clásica de control

La denominada teoría clásica de control[62] comprende aquellos métodos y descubrimientos realizados durante los años anteriores e inmediatamente posteriores a la Segunda Guerra Mundial. El uso de un lenguaje propio en la disciplina fue primordial para la expansión de la misma y el asentamiento y comprensión de los términos importantes. Muchos de los avances en el campo del campo de las comunicaciones que hoy en día forman parte del control automático no pasaron inicialmente al mismo al estar expresados en términos propios de ese campo del conocimiento. El control automático necesitaba términos independientes de cualquier dominio de aplicación.

La teoría clásica de control abstrae el estudio de los sistemas de su naturaleza física e incluso de sus condiciones de funcionamiento. El uso de medidas caracterizantes de la respuesta dinámica, como las constantes de tiempo, el uso de respuestas normalizadas para permitir la comparación de comportamientos de forma independiente de las magnitudes de las señales físicas, la utilización de parámetros adimensionales, la linealización en torno a un punto de

[59] Procesos de control o autorregulación dentro de organismos vivos.

[60] También se dice que lo es de la ciencia y hasta de la propia naturaleza.

[61] En la aplicación práctica (en la industria) de la disciplina del control se observa una mayor componente de uso de este lenguaje propio y mecanismos menos formales (en el sentido matemático) mientras que en el ámbito académico el aparato matemático tiene todavía una gran prevalencia (Bissell y Dillon, 2000).

[62] La distinción entre teoría clásica y moderna que suele realizarse tiende a crear cierta confusión ya que ambas aproximaciones tienen muy pocos años de diferencia y por lo tanto la "teoría moderna" no lo es tanto desde la perspectiva actual.

funcionamiento o la pérdida de magnitud física en las señales son algunas de las características principales que distinguen a la ingeniería de control y su lenguaje.

La base fundamental de la ingeniería de control es la abstracción que proporciona la aproximación basada en la idea de sistema, sin importar de qué tipo de sistema físico se esté hablando. La abstracción del sistema necesita del uso de esos parámetros neutros, independientes del dominio, sin ellos no habría disciplina de control automático. El sistema se trata como una caja negra, definido mediante un modelo denominado de función de transferencia que contiene y refleja el comportamiento dinámico de todos los elementos físicos que lo conforman sin posibilidad de aislarlos[63].

La transformación matemática de Laplace permite convertir las ecuaciones diferenciales que describen el comportamiento dinámico del sistema en ecuaciones algebraicas y (asumiendo condiciones iniciales nulas) obtener una representación de la relación entre la salida y la entrada de un sistema en forma de un cociente de polinomios con coeficientes reales. Este cociente es la denominada función de transferencia, función que captura el comportamiento dinámico natural del sistema, es decir, la forma en la que el sistema gana y pierde energía, sin necesidad de hacer referencia a una determinada entrada al mismo.

La situación de las raíces del numerador (denominadas ceros) y las del denominador (denominadas polos) de esta función de transferencia en el diagrama de Argand (plano complejo) permite anticipar cómo será la respuesta en el tiempo del sistema ante variaciones en la señal de entrada.

Los métodos gráficos son primordiales en las tareas de análisis y diseño dentro de la teoría clásica de control. Gran parte del lenguaje de control tiene por tanto un carácter gráfico. Estos métodos y representaciones proporcionan una abstracción más sobre el aparato matemático y conllevan una serie de razonamientos para el análisis y diseño en los que entra en juego y es primordial la experiencia del ingeniero. Muchos de estos métodos estaban orientados a la realización manual de las gráficas y se han visto revitalizados con el aumento de la capacidad de procesamiento y evolución del software en los ordenadores.

Como conclusión, puede decirse que el conocimiento en la teoría clásica de control utiliza una base matemática para describir los modelos de los sistemas, pero utiliza un lenguaje propio y elaborado sobre esta base para transmitir los conocimientos de la disciplina. Cada concepto utilizado en este lenguaje contiene referencias a los elementos matemáticos subyacentes y presenta relaciones complejas con otros conceptos del dominio.

[63] De hecho si la función de transferencia se obtiene por identificación ni siquiera se llegan a considerar los elementos físicos existentes.

4.4 Elección del formalismo de representación y herramienta de desarrollo de la ontología

La difusa definición del término ontología y la variedad de lenguajes y herramientas disponibles hacen que existan ontologías construidas utilizando diferentes formalismos, desde las aproximaciones que parten de la ingeniería del software (Java, UML, MOF), los lenguajes y formalismos de representación del conocimiento (marcos, sistemas KL-ONE y basados en lógicas descriptivas, KIF, Ontolingua, etc.) hasta las aproximaciones propias de Internet y la Web Semántica (RDF(S), OWL).

En el campo de la ingeniería del conocimiento las dos aproximaciones que suelen utilizarse para construir ontologías de uso práctico en la actualidad son OWL (que recoge toda la investigación en el campo de las lógicas descriptivas y ha sido propuesto desde el W3C como un estándar para la creación de ontologías en Internet) o algún formalismo basado en la idea de marcos (más cercana a las técnicas de orientación a objetos del campo de la ingeniería del software). La elección de uno u otro formalismo deberá hacerse atendiendo a las consideraciones expresadas en el apartado 3.6, teniendo en cuenta las características del conocimiento del dominio elegido que se han introducido anteriormente.

El estudio y exposición de características del conocimiento en control automático realizado en las secciones anteriores da como resultado las reflexiones que se exponen en los siguientes párrafos.

4.4.1 Elección del formalismo para el desarrollo de la ontología

La idea de sistema, primordial en la disciplina de ingeniería de control es una aproximación de "mundo cerrado" respecto al conocimiento. Todas las estructuras que conforman el conocimiento del dominio están bien definidas desde la base que representa el modelo matemático en función de transferencia y por tanto todo lo que no aparezca en la conceptualización o pueda deducirse de ésta se considerará falso.

La abstracción del sistema en la teoría clásica de control conlleva que la estructura de los sistemas físicos no sea relevante para estudiar su comportamiento dinámico y realizar el diseño de un sistema de control. La naturaleza y composición de los sistemas físicos no tendrá que ser conceptualizada en el dominio de estudio elegido para poder representar el conocimiento de la ingeniería de control. La ontología para representar los conceptos de control será por tanto bastante diferente en cuanto a la naturaleza de sus conceptos con respecto a las que pueden

existir en otras ramas de la ingeniería de sistemas. Por ejemplo, no existirán jerarquías de componentes físicos, ni será necesario representar la composición física de los mismos para formar el sistema total.

La conceptualización y la ontología resultante serán utilizadas en tiempo de ejecución sin que existan modificaciones en su estructura conceptual, es decir, bajo el denominado paradigma de pre-coordinación, tal como se definió en la sección 3.6. Se puede crear nuevo conocimiento, pero este será siempre obtenido a partir de la aplicación de los conceptos existentes a los datos particulares de un problema concreto, es decir, sólo se verá afectado el componente asertivo y no el terminológico. En definitiva, no será necesario el uso de un clasificador automático en tiempo de ejecución.

El clasificador automático de conceptos tampoco será necesario en tiempo de diseño ya que las jerarquías de generalización no son demasiado complejas y pueden construirse y mantenerse de forma manual. De hecho esta relación de subsunción o "subtipo" no será tan relevante como se verá que lo es, por ejemplo, la composición de conceptos.

La conceptualización a realizar deberá reflejar la estructura y construcción del lenguaje de control encima del matemático de forma que este lenguaje sea expuesto para permitir la adquisición de conocimiento relativo a la tarea de diseño de compensadores. El diseño de la ontología conlleva la utilización generalizada de instancias como se verá en los siguientes puntos. Habitualmente se considera que la creación de una ontología consiste en la creación de una estructura de clases (con sus propiedades y axiomas), mientras que las instancias forman parte de una visión particular de aplicación de la ontología[64]. Las clases más las instancias formarían una base de conocimiento[65]. En este sentido puede decirse que lo que se ha desarrollado ha sido tanto una ontología como una base de conocimiento. Tal como se verá, bajo la definición estricta de ontología y base de conocimiento presentada antes, lo que se ha definido en la ontología es la estructura del lenguaje de control (aparte de los conceptos matemáticos subyacentes) mientras que los términos concretos se han modelado como instancias.

La naturaleza del conocimiento en la teoría clásica de control hace que algunas propiedades asignadas a los conceptos del dominio deban ser calculadas por procedimientos externos, habitualmente por medio de programas de cálculo numérico. Este hecho tiene gran semejanza con la denominada asignación de

64 Así se separó entre TBox (Terminological Box) y ABox (Assertional Box) desde los comienzos de las representaciones basadas en redes terminológicas. Todos los sistemas basados en esta idea implementaban la TBox, aunque no todos la ABox.

65 Aunque, como se menciona en (Noy y McGuinness, 2001) "en realidad, hay una fina línea donde la termina la ontología y empieza la base de conocimiento".

procedimientos (procedural attachment) introducida como uno de los mecanismos básicos del formalismo de los marcos. El lenguaje OWL no incluye mecanismos para explicitar la asignación de procedimientos.

Otra de las técnicas utilizadas en la conceptualización realizada implica el encadenamiento de propiedades (property chaining), una forma de representar conceptos que se definen mediante la referencia a una secuencia de propiedades. Este mecanismo fue explícitamente eliminado de las especificaciones de partida para el desarrollo del lenguaje OWL (Heflin ed., 2004).

En definitiva, el uso de OWL como lenguaje para la especificación de la ontología no ofrece soluciones para algunas de las necesidades existentes y, por otra parte, tiene funcionalidades que son contraproducentes para este caso (como la asunción de mundo abierto por ejemplo). Pese a que podrían utilizarse mecanismos ad-hoc programados en el lenguaje de la aplicación que implementasen las funcionalidades que OWL no ofrece, todavía quedaría solucionar las características que constituyen un inconveniente en el dominio de estudio, por ejemplo deberían utilizarse numerosos axiomas de clausura para "cerrar" el mundo descrito. Y aunque se llevase a cabo esto, al final resultaría una ontología cuya semántica se implementaría en su mayor parte en un procesamiento externo, fuera de la teoría lógica de modelos.

La decisión sobre el formalismo de representación para la ontología se decanta por tanto hacia el uso de la aproximación basada en los marcos. Con esta decisión, tal como se ha mencionado, se pierde parte del carácter estrictamente formal que caracteriza a las aproximaciones basadas en lógicas descriptivas (como OWL). Parte de la semántica de las estructuras a construir tendrá que ser implementada en una aplicación que se construya a tal efecto[66]. Con esta decisión se pretende dar prioridad a la búsqueda de las estructuras de conocimiento del dominio y su descripción de forma más libre que mediante el uso de un formalismo como las lógicas descriptivas. En cierto sentido se sigue la idea de simplicidad y flexibilidad en la representación de la ontología La naturaleza del trabajo, del conocimiento, y del uso que se pretende dar a la ontología aconsejan primar la simplicidad en aras a la comprobación de la aproximación, tal como se cita en algunos artículos dedicados a la realización práctica de software basado en ontologías (Knublauch, 2002).

[66] Como se verá más adelante, el uso de OWL también requeriría la implementación de parte de la semántica de forma separada de las estructuras definidas en este lenguaje.

4.4.2 Elección de la herramienta para el desarrollo de la ontología

Las especiales características de la representación del conocimiento hacen que la elección de una herramienta sea un punto crucial. La herramienta para la edición de una ontología no sólo es una interfaz de usuario sino que expone las estructuras de conocimiento que podrán utilizarse para construir la conceptualización bajo un determinado formalismo subyacente y para posteriormente poblarla, es decir, introducir las instancias que en conjunto con la ontología formarán la base de conocimiento. La misma herramienta es muchas veces tanto un editor de ontologías como una herramienta de adquisición del conocimiento completa. Y es más, algunas herramientas incorporan la posibilidad de crear aplicaciones con interfaces gráficas e implementar diversos tipos de razonamiento sobre las estructuras conceptuales. Por otro lado, muchas veces parte de la semántica de la representación es construida dentro de la propia herramienta, como en el caso de la comprobación de restricciones y axiomas impuestos en la ontología[67].

Existen diferentes revisiones sobre herramientas de desarrollo de ontologías (Mizoguchi, 2004), (Corcho et. al., 2003), (Duineveld et. al., 1999), (Damjanović et. al., 2004) donde éstas se estudian desde diferentes puntos de vista: desde la expresividad que son capaces de representar hasta aspectos de usabilidad. Sin embargo, existen aspectos relevantes desde el punto de vista pragmático a la hora de crear una ontología que no han sido demasiado estudiados.

Uno de estos aspectos es el grado de uso de la herramienta o, lo que es lo mismo, la comunidad de usuarios con que la herramienta cuenta y la actividad y proyectos desarrollados con la misma. Este hecho es relevante para la estabilidad y evolución de la herramienta así como para la existencia de una colección de ontologías que pueden ayudar al desarrollo de la propia.

Otro aspecto relevante es la influencia que sobre la elección de la herramienta tiene el tipo de aplicación que pretende desarrollarse. Entre aspectos a tener en cuenta en este sentido están la interacción entre la aplicación y la ontología (existencia de una interfaz de programación de aplicaciones - API - completa y flexible) o la integración con motores de inferencia (emparejamiento de patrones, lógica, etc.). Ambos aspectos son muy importantes a la hora de crear una ontología que posteriormente va a servir como base para la ejecución de una aplicación.

[67] Esto ocurre sobre todo en el paradigma de los marcos. La semántica en el caso de las lógicas descriptivas (OWL) está autocontenida en el lenguaje de representación en la forma de una teoría lógica.

Los dos aspectos anteriores son especialmente relevantes cuando el formalismo elegido es basado en marcos ya que, idealmente, las aproximaciones basadas en lógicas descriptivas contienen la semántica dentro de los axiomas del lenguaje y podrían ser llevadas de una a otra herramienta de forma totalmente transparente. En el caso de las aproximaciones basadas en marcos idealmente también debería poder exportarse e importarse cualquier ontología en cualquier herramienta que implementase el protocolo OKBC, pero en la práctica no todas las herramientas implementan todos los aspectos del protocolo.

La herramienta elegida finalmente fue Protégé (Genari et. al., 2002). La decisión fue tomada en base a los resultados de los estudios mencionados y a la propia experiencia de uso. Protégé nació del trabajo seminal de Edward Feigenbaum en el laboratorio de informática médica de Stanford[68,69] (Musen, 1999) y ha evolucionado hasta ser una de las herramientas más utilizadas entre los editores de ontologías no comerciales. Entre las características que hicieron tomar la decisión a favor de Protégé están:

- La madurez de la herramienta, con más de 20 años de evolución.
- La actualización semestral incorporando numerosas novedades que van desde la interfaz gráfica hasta la mejora del rendimiento.
- La posibilidad de construir la ontología mediante el formalismo de los marcos o con OWL/RDF con una misma apariencia en el interfaz y el funcionamiento.
- El número de usuarios creciente exponencialmente.
- Las herramientas de colaboración como foros, wiki y lista de distribución, todos activos y con elevado índice de uso.
- El carácter de distribución bajo licencia de código abierto.
- La arquitectura de la herramienta, abierta y basada en “plugins”.
- El número de ontologías y proyectos llevados a cabo.

4.4.3 Componentes básicos para la construcción de la ontología

Una vez elegido el formalismo y la herramienta es conveniente especificar los elementos constructivos con que se cuenta para crear la ontología basada en marcos. En el caso de Protégé la nomenclatura utilizada para estos elementos es la siguiente (Noy y McGuinness, 2001):

68 Feigenbaum fue el responsable del desarrollo de DENDRAL, uno de los primeros sistemas expertos productivos en un área de aplicación real. Posteriormente, con la colaboración de Bruce Buchanan y Ted Shortliffe se construyó MYCIN, el sistema experto quizás más conocido. Proyectos como INTERNIST, MOLGEN o EON dieron lugar al desarrollo de la herramienta Protégé

69 Es muy interesante ver la evolución paralela de dos departamentos de la Universidad de Stanford: El Knowledge Systems Laboratory (KSL) y el Stanford Medical Informatics (SMI).

- *Clases*, que representan conceptos que son agrupaciones de individuos que comparten un conjunto de propiedades (recogidas en los denominados slots).
- *Slots*, recogen las propiedades que distinguen a las clases. Sobre los tipos de valores que aparecerán en los slot se pueden establecer restricciones por medio de las *facetas* (cardinalidad, valores por defecto, dominio, rango, carácter de requerido, …)
- *Instancias*, representan individuos concretos de una clase, correspondiendo a elementos reales en el mundo representado. La distinción entre instancia como concepto de la ontología e individuo como elemento en el mundo externo es importante. En el formalismo de los marcos se sigue la asunción de nombre único (unique name assumption - UNA -) de forma que cada instancia representará a un individuo y no habrá más que una instancia para representarlo. En el caso de las lógicas descriptivas y OWL dos instancias en la ontología pueden referirse a un mismo individuo: es una de las consecuencias de la asunción de mundo abierto.
- *Axiomas*. Protégé ofrece un lenguaje para la imposición de restricciones sobre los slot de forma más flexible que por medio de las facetas predefinidas. Los axiomas se introducen en un lenguaje que es un subconjunto de la lógica de primer orden denominado PAL (Protégé Axiom Language)[70].

En cuanto a capacidad de razonamiento, el formalismo de marcos implementado en Protégé permite realizar de forma automática los siguientes tipos de inferencia: herencia, valores por defecto y comprobación de restricciones (Wang et. al., 2006). Si se necesita otro tipo de inferencia o razonamiento deberá ser construido aparte (Protégé incluye plugins para diversos motores de inferencia basados en reglas de producción, lógica, etc.). La forma más habitual de construir el razonamiento extra necesario es el uso de reglas.

4.4.4 Metodología empleada

La construcción de una ontología es un trabajo colaborativo y multidisciplinar. Normalmente el experto en el dominio (el que posee el conocimiento a conceptualizar) y el experto en ingeniería de conocimiento son dos sujetos diferentes (incluso puede existir un tercero, el programador de la aplicación). En la época de los sistemas expertos el proceso de construcción de una representación del conocimiento de un dominio era un proceso de extracción del conocimiento

[70] Es importante destacar que mediante estos axiomas se realiza (dentro de la herramienta) comprobación de restricciones, es decir, se comprueba si los datos introducidos cumplen las restricciones y si no es así, se advierte al usuario. Este funcionamiento es menos potente que la comprobación de consistencia que OWL puede realizar a partir de axiomas similares, donde el razonador trata de construir un modelo que satisfaga esos axiomas como forma de comprobar la consistencia de la representación (Wang et. al., 2006).

existente en la cabeza del experto. El paradigma del modelado del conocimiento no persigue ese fin y, por tanto, no utiliza ese método. El esquema de representación a construir debe ser un modelo del conocimiento del dominio y no del conocimiento del experto elegido para crearlo.

Habitualmente las metodologías para la construcción de ontologías describen un proceso con las siguientes fases[71]:

- Definir el alcance de la ontología.
- Evaluar la posibilidad de reutilización de otras ontologías y/o partes de ontología. Si existen ontologías que puedan ser reutilizadas habría que incluir el proceso para conseguir reutilizarla/s, proceso que a su vez consta de diferentes fases y tareas a realizar.
- Enumerar los términos que aparecen en el dominio.
- Definir la taxonomía de conceptos. De arriba abajo, de abajo arriba o combinando las dos.
- Definir las propiedades.
- Definir las facetas.
- Definir las instancias.
- Comprobar las estructuras creadas.

En el caso de la conceptualización del dominio objeto de la ontología descrita en esta tesis las particularidades del conocimiento de dicho dominio hacen que, por ejemplo, no sea viable el plantear una enumeración de conceptos y una descripción de la taxonomía de los mismos, debido a que la taxonomía no es la estructura de conocimiento más importante, tal como se ha apuntado en la sección 3.6 y se verá en próximos apartados.

En vez de realizar una enumeración de los conceptos y plantear la taxonomía de los mismos, la aproximación elegida para realizar la ontología está basada en la importancia que tiene el lenguaje de control dentro de la disciplina y de las técnicas de la teoría clásica de control en particular. La conceptualización se ha realizado (y así se mostrará) a partir de ejemplos concretos de uso de este lenguaje en el dominio, de una manera similar en la que los casos de uso se emplean para obtener un modelo de un sistema software.

4.5 Estructuras de conocimiento en la ontología

Seguidamente se presentan los aspectos más relevantes de la conceptualización llevada a cabo. Para cada uno de ellos se presentará en primer lugar el aspecto en

[71] Algunas metodologías no consideran alguna de estas fases.

los términos del lenguaje del control automático para seguidamente abordar la discusión sobre sus posibles formas y problemas de conceptualización incluyendo, si los hay, los trabajos previos relacionados con el problema. Finalmente se presentará la formalización realizada.

Se ha seguido una nomenclatura aceptada más o menos como estándar en la descripción de los conceptos en las ontologías (Noy y McGuinness, 2001). Así, los nombres de clases, slots e instancias no contienen caracteres extendidos de ASCII ni espacios en blanco. En los conceptos que contienen más de una palabra éstas se identifican poniendo la letra inicial de cada una de ellas en mayúscula. La primera letra de cada concepto que sea una clase será mayúscula, mientras que para las instancias será minúscula. Todos los slot tienen un nombre que comienza en minúsculas por el prefijo "has", facilitando su distinción respecto a clases e instancias y remarcando el carácter de relación que indica la posesión de una determinada propiedad.

Los diagramas que muestran la estructura de la ontología no siguen ninguna notación estandarizada. Suele utilizarse, en ocasiones, notación basada en el lenguaje UML (usando sólo diagramas básicos) como representación gráfica para ontologías. Esta notación no parece sin embargo lo suficientemente expresiva para reflejar la estructura de la ontología de forma conveniente. La notación utilizada se ha creado ex profeso para el fin perseguido. A continuación se exponen brevemente las convenciones utilizadas en esta notación para la representación de los diferentes elementos constructivos de la ontología.

Se utilizará un esquema de color para distinguir a los tres componentes principales de una ontología: las clases, los slot y las instancias. Este esquema es similar al empleado en la herramienta Protégé. También se ha intentado que puedan distinguirse los diferentes elementos en imágenes en escala de grises.

Las **clases** se representarán mediante un rectángulo con color de relleno ocre/marrón, indicando el nombre de la clase dentro de dicho rectángulo. Para facilitar la visualización en escala de grises el rectángulo tendrá un borde grueso negro y punteado. Además, una clase se distingue fácilmente porque su nombre comienza por mayúscula, mientras que en instancias y slots comienza por minúscula:

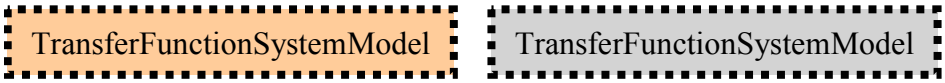


Figura 4.1. Representación gráfica de una clase

Los **slot** se representan como rectángulos de color azul con el nombre del mismo en su interior y sin ningún tipo de borde. Además, los nombres de los slot siempre comienzan por el prefijo "has":

Figura 4.2. Representación gráfica de un slot

Finalmente, las **instancias** se representan como rectángulos de color violeta con un borde grueso y de línea contínua en un tono más oscuro. Además, las instancias se distinguen porque su nombre comienza en minúscula y nunca con el prefijo "has":

Figura 4.3. Representación gráfica de una instancia

Dada la importancia que las instancias tienen en la ontología y base de conocimiento desarrolladas y dado que los esquemas basados en instancias permiten obtener una visión más clara de la estructura de la ontología, existe una versión más completa para la representación de una instancia que incluye información acerca de los slot y los valores que en ellos aparecen. En la figura 4.4 se observa un ejemplo.

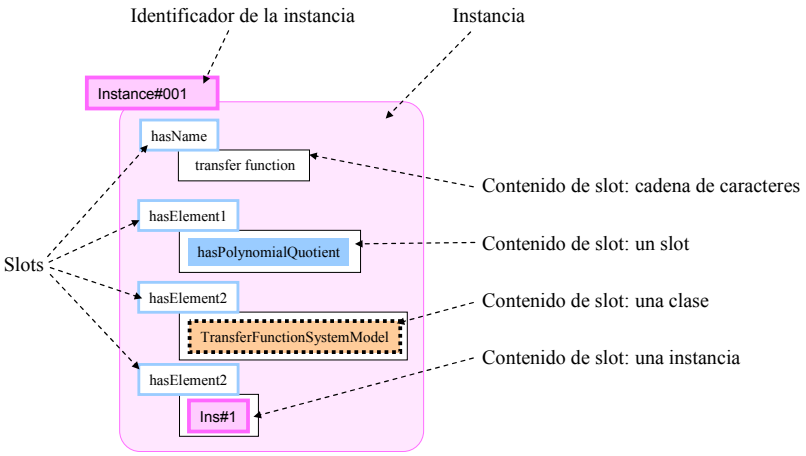


Figura 4.4. Representación gráfica de una instancia con información extendida

En este diagrama de la instancia instance#01 se puede observar que la clase que la describe tiene cuatro slots[72]:

- El slot hasName tiene como posibles valores (fillers) cadenas de caracteres. El slot perteneciente a una instancia se representa en este diagrama como un rectángulo blanco con borde azul, existiendo otro rectángulo blanco que recogerá el contenido del slot. En este caso el contenido es un string que se representa como un texto sin ningún tipo de adorno.
- El slot hasElement1 tiene en este ejemplo a otro slot como valor.
- El slot hasElement2 contiene como valor una clase.
- El slot hasElement3 contiene una instancia como valor del mismo.

Las instancias se representan por su nombre (que será una cadena de caracteres correspondiente al valor de uno de sus slots o a una combinación de varios slots y/o nombres de otras instancias incluidas en slots ésta[73]) si éste es relevante o por medio de un identificador que será el nombre de su clase empezando por minúsculas y habitualmente abreviado seguido de una almohadilla y un número entero para diferenciar diferentes instancias de una misma clase dentro de la misma figura.

En ocasiones se representarán instancias sin incluir ningún valor de slot, como en el caso de la figura 4.5.

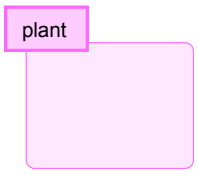


Figura 4.5. Representación extendida de una instancia sin representar sus slots

En este caso no es que la instancia carezca de slots, sino que en el contexto en el que aparece éstos no tienen relevancia.

Las **facetas** más importantes para un slot serán el tipo, la cardinalidad, valores por defecto, dominio, rango, carácter de requerido,... El tipo de un slot es la naturaleza de los elementos que pueden encontrarse como valores para ese slot (ejemplos de tipo son "String", "Instance", "Float" o "Class"). El dominio de un slot es el

[72] Para mantener la claridad sólo se incluirán los slot más representativos de cada instancia dependiendo de la estructura de conocimiento que se pretenda reflejar. La mayoría de las clases, y por lo tanto sus instancias, tienen slots dedicados a introducir el nombre de la instancia, comentarios, etc., que no son relevantes a la hora de describir la estructura del conocimiento.

[73] Se utiliza un nombre similar al construido mediante las posibilidades de la herramienta "set display slot" de Protégé. Ver: http://protege.stanford.edu/doc/tutorial/get_started/set_display_slot.html

conjunto de clases en cuya descripción aparece ese slot. El rango de un slot es el conjunto de clases cuyas instancias pueden hacer las veces de valor para ese slot (el rango sólo tiene sentido cuando el tipo del slot es "Instance" o "Class").

Los **axiomas** descritos en el lenguaje PAL no son relevantes a la hora de exponer la estructura de conocimiento modelada en la ontología al nivel en el que se describirá[74]. En el formalismo de marcos, al contrario que en las lógicas descriptivas, los axiomas se utilizan para describir conceptos, restringiendo el valor de los slot mediante expresiones que no se pueden recoger por medio de las otras facetas disponibles. Sin embargo estas restricciones no forman parte de la definición del concepto, como es el caso de las lógicas descriptivas, sino de su descripción y por lo tanto sólo sirven para comprobar, en tiempo de diseño, si existen errores en la introducción de datos[75].

En general se ha preferido presentar diagramas como los explicados en párrafos anteriores frente a capturas de pantalla del editor Protégé, ya que se considera que esta segunda opción es menos conveniente a la hora de comprender las estructuras de conocimiento.

Los términos utilizados a la hora de describir una ontología suelen ser motivo de confusión habitualmente. Se llamará en general "concepto" (concepto) o "marco" (frame) a cualquiera de los tres elementos básicos: clase, slot e instancia. Se llamarán "atributos" o "propiedades" a los conceptos que sirven para establecer relaciones entre otros conceptos en la ontología. El caso más sencillo de atributo será recogido por conceptos representados como slots pero también habrá atributos que sean expresados mediante instancias, tal como se expondrá.

Los conceptos en la ontología se han introducido utilizando el idioma inglés ya que de esta forma ésta puede tener mayor difusión. Por lo tanto, en la exposición de la conceptualización también se emplearán los términos en inglés. El permitir que en una misma ontología exista la posibilidad de utilizar diferentes idiomas para designar al mismo concepto es un campo activo de investigación (aunque este problema se podría abordar de forma más informal utilizando aproximaciones ad-hoc relativamente sencillas).

Los nombres de conceptos que aparecen en la ontología, ya sean clases, slots o instancias, aparecerán en el texto con un tipo letra Courier, como en este ejemplo: `ClaseDePrueba`.

---

[74] De hecho, el lenguaje PAL sirve sólo para establecer restricciones sobre el conocimiento existente, no para establecer axiomas tal como se entienden en una teoría lógica.

[75] Aquí se puede ver una diferencia fundamental entre los marcos y las lógicas descriptivas y más en concreto respecto a OWL. La herramienta es, en el caso de los marcos, la que recoge de manera expresa la semántica existente en las restricciones sobre los slot. Las facetas son restricciones construidas en la propia herramienta. Los axiomas PAL también son gestionados por la misma. En OWL las restricciones en las relaciones se expresan de forma independiente de la herramienta, de forma que la ontología construida es totalmente independiente.

Se presentarán los fundamentos de diseño de la ontología según fue el proceso de construcción. La lectura de la descripción de las diferentes estructuras presentadas a continuación no puede ser totalmente secuencial, ya que en algunos puntos la conceptualización de un determinado concepto utilizará conceptos que se describen posteriormente. Por ejemplo, para presentar la conceptualización de las características cualitativas hará falta presentar antes la conceptualización de las precondiciones. Pero para presentar estas últimas hará falta haber presentado previamente las características cuantitativas. Esta misma situación aparece en varias ocasiones y supone un importante problema, además de para la descripción de la conceptualización, para el propio diseño de la misma y de la ontología en su totalidad.

A continuación se presenta la estructura de los siguientes apartados. En primer lugar se trata la conceptualización de los conceptos matemáticos utilizados para construir el modelo en función de transferencia y después se presenta la conceptualización de este modelo como tal. La sección 4.5.2 se dedica a la conceptualización de la topología del sistema representada en un diagrama de bloques. Estos diagramas de bloques representan a los sistemas físicos, contienen su modelo en función de transferencia y son a los que se hará referencia en los procesos de análisis y diseño. Teniendo en cuenta esta correspondencia entre el sistema real y su modelo en función de transferencia representado en un bloque, se podrá hablar en las expresiones sobre el sistema y realizar los cálculos sobre el modelo utilizado. La sección 4.5.3 describe la conceptualización de los términos (entidades) del lenguaje de la teoría clásica de control basado en ofrecer nombres nuevos a conceptos ya representados. Posteriormente se presenta la conceptualización de las características aplicables a los sistemas dentro de la teoría clásica de control. Estas características son el resultado de la tarea de análisis, las que caracterizan el comportamiento dinámico del sistema y sirven de guía en el proceso de diseño de los controladores.

Todas las conceptualizaciones se mostrarán a partir de ejemplos concretos (expresiones utilizadas por ese lenguaje del control) que guiaron el proceso de desarrollo de la ontología, de manera similar al empleo de casos de uso al obtener un modelo de un sistema software.

4.5.1 Conceptualización de las estructuras matemáticas algebraicas

4.5.1.1 Estructuras de álgebra elemental utilizadas en teoría de control

La teoría clásica de control se basa en el uso de estructuras algebraicas que conforman los modelos utilizados para el análisis y diseño de sistemas. Como se

ha introducido, la función de transferencia es la estructura matemática algebraica que recoge y representa el comportamiento dinámico del sistema en la teoría clásica de control[76].

La función de transferencia es un cociente de polinomios en la variable compleja s (s representa una frecuencia compleja, fruto de la aplicación de la transformada de Laplace sobre un conjunto de ecuaciones diferenciales lineales). Todos los coeficientes de estos polinomios serán números reales, ya que son el resultado de aplicar las diferentes leyes de la física sobre el sistema de estudio (para obtener las ecuaciones diferenciales) y, por lo tanto, hacen referencia a los valores de los componentes físicos que forman el sistema. Las raíces de los polinomios serán números complejos, números reales puros o números imaginarios puros. La conceptualización, por tanto, deberá recoger elementos como el cociente de polinomios, el polinomio, coeficientes, raíces, diferentes tipos de números, etc.

La ontología EngMath (Gruber y Olsen, 1994) podría ser una candidata para reutilización a la hora de conceptualizar los aspectos matemáticos citados. Sin embargo, esta ontología está orientada a la representación de magnitudes físicas más que a los conceptos matemáticos utilizados en la teoría clásica de control. La conceptualización necesaria no es compleja ni extensa por lo que se ha decidido realizarla completamente. Se empezará por el concepto de más alto nivel en cuanto a estructura, el cociente de polinomios, y se irá conceptualizando cada uno de los conceptos que lo forman.

El concepto cociente de polinomios (clase `PolynomialQuotient`) tendrá dos slots que serán cada uno de los polinomios que actúan como numerador y denominador respectivamente. Se han representado solamente los dos componentes que forman la esencia del cociente de polinomios. No se ha conceptualizado el concepto "cociente" o "división", del que éste sería un caso particular.

Los individuos que irán en los slot del cociente de polinomios pertenecerán al concepto o clase que describe a un polinomio (clase `Polynomial`). La conceptualización del polinomio es algo más controvertida que la anterior. La descripción de un polinomio puede realizarse de diferentes formas. En concreto en los métodos de la respuesta en el tiempo dentro de la teoría clásica de control son relevantes dos de estas definiciones[77]:

[76] Esta función es válida en un entorno determinado de un punto de funcionamiento. La validez de la misma en entornos alejados del punto de funcionamiento depende del mayor o menor carácter lineal de la curva de respuesta en el entorno de ese punto. A efectos del presente estudio estos aspectos se obviarán y no se recogerán en la presente ontología.

[77] En el estudio de la respuesta en frecuencia también se utiliza una tercera forma: la denominada de Bode o formato constante de tiempo donde los polinomios están factorizados con términos de la forma (1+a·s)

- Una serie de raíces complejas y un coeficiente principal. La función de transferencia con los polinomios así expresados se dice que está en formato polo-cero.
- Una serie de coeficientes reales ordenados en orden ascendente o descendente de potencias en s. La función de transferencia con los polinomios así expresados se dice que está en formato polinomial.

La conceptualización de ambas estructuras para la definición del polinomio supone una redundancia en la ontología (ya que una se puede calcular a partir de la otra). Sin embargo el no incluir las dos posibilidades supondría darle más importancia a una que a otra y, lo que es más importante, tener la imposibilidad de referirse directamente a los conceptos no representados. El recurso de incluir información redundante en una base de conocimiento no es nuevo, ya en sistema KRL (Knowledge Representation Language) por ejemplo se utilizó como una forma de mejorar la eficiencia computacional (Bobrow y Winograd, 1976).

Los conceptos que se necesitan para definir cada una de las formas de representar un polinomio son: para la primera, una lista de números complejos (las raíces se han considerado como números complejos que, eventualmente, pueden tener parte imaginaria, o real, igual a cero) más un número real (el coeficiente principal); para la segunda: una lista ordenada de números reales. En este segundo caso será, por tanto, necesario tener algún sistema para expresar el concepto de orden dentro de una lista de elementos.

Además de las estructuras de representación de cada una de las formas de representar un polinomio deberá existir un mecanismo que permita obtener una a partir de otra. Por otro lado, no parece conveniente mezclar los slot correspondientes a las dos formas de representar el polinomio asociándolos directamente al polinomio. Por estas razones se han creado conceptos que representan las diferentes descripciones de un polinomio. Así, por un lado, se tendrá la descripción en raíces y coeficiente principal (tendrá como slots uno para las raíces cuyos elementos (fillers) serán instancias de números complejos y otro para el coeficiente principal, que será una instancia del concepto que representa al número real) y por otro la descripción como coeficientes ordenados según la potencia decreciente de la variable (que contará con un slot de cardinalidad múltiple donde se almacenarán los coeficientes).

En el segundo caso la lista de coeficientes recogida en el slot deberá estar ordenada en potencias decrecientes de la variable. El orden en la lista de instancias no se ha implementado como concepto sino que se usa el

almacenamiento interno del formato utilizado por la herramienta Protégé, que conserva la ordenación de elementos en slot múltiples[78].

Estos dos conceptos de tipos de descripción de polinomio (clases `RootsAndCoeffPolynomialDescription` y `DescendingPowersOfVariablePolynomialDescription`) se agrupan como subtipos de una clase padre de nombre `PolynomialDescription`. Ambos conceptos son un "tipo-de" descripción de polinomio y por eso están organizados como subclases de esta forma. Sin embargo la clase padre sólo tendrá como fin agrupar a los dos (o más, si se especificasen) tipos de descripción de polinomios, no tendrá ningún slot que hereden los hijos y por lo tanto desde el punto de vista de la herencia de propiedades no aportará nada, sólo podrá servir para ofrecer una explicación de que ambos conceptos hijos son formas de describir un polinomio u ofrecer al usuario la posibilidad de representar un polinomio en pantalla por medio de otro tipo de descripción.

A continuación se trata la conceptualización de los números, necesaria para expresar los coeficientes, las raíces y el coeficiente principal de las descripciones de polinomios vistas anteriormente.

4.5.1.2 Representación de los números

La distinción entre los números reales y naturales, enteros, racionales, irracionales, etc., no es demasiado importante en la conceptualización presente, ya que estos tipos de números no son de utilidad en el campo de estudio, donde lo que representan estos números son magnitudes físicas y/o de señales que tendrán un valor real (los números enteros se utilizan en ciertas características concretas pero pueden ser representados como números reales sin problema). El caso de los números complejos es diferente, ya que éstos tienen una gran importancia en el campo del control automático y deberán recogerse por tanto explícitamente en la conceptualización. En resumen, puede decirse que sólo se utilizarán números reales para recoger valores y medidas cuantitativas, y números complejos dedicados a la representación de las raíces de los polinomios.

La ontología EngMath (Gruber y Olsen, 1994) contiene una conceptualización de los números tomada de kif-numbers que, a su vez, está preconstruida dentro de

[78] En realidad esta conservación del orden es un aspecto casi fortuito y no construido en el formalismo de rerpresentación del conocimiento ya que Protégé no implementa (Wang et. al., 2006) la faceta :COLLECTION_TYPE de un slot que sí está definida en el protocolo OKBC (Chaudhri et. al., 1998b). Ver también http://article.gmane.org/gmane.comp.misc.ontology.protege.owl/4886/match=multiple+cardinality+slot+ordered.
Mencionar que OWL tampoco admite la representación de listas ordenadas aunque algunos trabajos consiguen presentar la mayoría de la semántica de las mismas ((Drummond et. al., 2006))

Ontolingua, no conteniendo una definición de composición entre los números complejos y reales (siendo ambos subclases de la clase Number):

```
Number
    Real-Number
       Rational-Number
          Integer
             Even-Integer
             Odd-Integer
             Natural
             Positive-Integer
             Nonnegative-Integer
    Positive
    Negative
    Complex-Number
 Zero
```

(Extraído de http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/kif-numbers.html)

```
.....
.....

(in-theory 'kif-numbers)


(define-class NUMBER (?x)
  "Number")

(define-class REAL-NUMBER (?x)
  "Real number"
  :def (number ?x))

.....
.....

(define-class COMPLEX-NUMBER (?x)
  "Complex number"
  :def (number ?x))

.....
.....
```

(Extraído de http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/kif-ontology.lisp.html)

La conceptualización de los números reales y complejos tiene varias posibles soluciones. Una de ellas consideraría al número real como entidad principal mientras que el número complejo estaría formado por dos números reales: uno haciendo las veces de la denominada "parte real" y el otro haciendo las veces de la denominada "parte imaginaria". La segunda aproximación se apoya en el hecho de

que cualquier número real puede ser representado como un complejo en el que su parte imaginaria es cero, por lo tanto los números complejos incluirían a los reales, no siendo necesario (llevando la consecuencia al límite) representar los números reales explícitamente en este caso (figura 4.6):

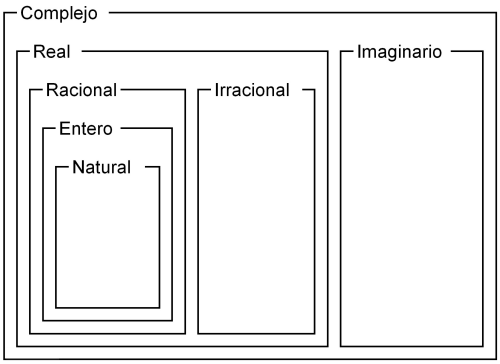


Figura 4.6. Clasificación de los números

Sin embargo, esta segunda aproximación no es útil en el caso de la presente ontología. El número real recoge la existencia de una magnitud medible mientras que los números complejos pueden verse como un artificio que se usa para representar dos medidas en un solo elemento[79]. Por esta razón, y dado que será necesario también definir parte real y parte imaginaria como magnitudes con valor (y por tanto como números reales), se ha optado por dejar como concepto primitivo al número real y hacer que el número complejo esté formado por un par de números reales[80]. Esta representación recoge la estructura del número complejo útil en ingeniería, obviándose aspectos más relacionados con la matemática como la representación de $\sqrt{-1}$ asociada con la parte imaginaria. Cabe mencionar que el uso de complejos y reales se hace discrecionalmente a lo largo de la conceptualización: por ejemplo, en el caso de las raíces de un polinomio, se supone que éstas serán, en todos los casos, instancias de la clase `ComplexNumber` (aunque tengan la parte imaginaria igual a cero). En el caso de los coeficientes de los polinomios éstos se describirán como instancias de números reales[81].

79 A los números complejos se les denominaba originariamente "imaginarios" por su falta de correspondencia con magnitudes y números "reales".

80 Este aspecto de la conceptualización está de acuerdo con algunas otras conceptualizaciones, como la realizada sobre el concepto ComplexNumber en la ontología SUMO (Suggested Upper Merged Ontology (Niles y Pease, 2001)). Allí se define ComplexNumber como "A Number that has the form: x + yi, where x and y are RealNumbers and i is the square root of -1"
<http://virtual.cvut.cz/ksmsaWeb/browser/print/3%23ComplexNumber>

81 Puede comprobarse cómo se sigue una aproximación pragmática a la hora de realizar la conceptualización. Una conceptualización más completa y de "grano más fino" tendría que considerar polinomios en los que los coeficientes pudieran ser complejos e incluir otros tipos de polinomios como los de varias variables, etc.

El concepto correspondiente al número real está compuesto (tiene un slot que lo recoge) por un valor numérico, expresado en el tipo de datos adecuado dentro del agente que almacene ese valor. En la presente ontología este valor es un tipo float del lenguaje de programación Java, o lo que es lo mismo, un número en coma flotante de simple precisión según el estándar IEEE 754 con tamaño de almacenamiento de 32 bits (rango de 1.40239846e–45 a 3.40282347e+38)[82].

La clase que representa a los números reales se denomina `RealNumber` y el slot donde se recoge el valor numérico se denomina `hasNumericalValue`. En cuanto a los números complejos éstos están representados por la clase `ComplexNumber` que, al igual que en caso de los polinomios, tiene slots en los que se almacenan las formas de describir un número complejo. En esta ontología se han representado la forma "parte real y parte imaginaria" y la forma "módulo y argumento". En la figura 4.7 se puede ver un ejemplo de instancia de número complejo. La instancia a la izquierda pertenece a la clase `ComplexNumber`, las dos siguientes son instancias de la clase `RealAndImaginaryPartsDescription` y `ModulusAndArgumentDescription` (ambas subclases de `ComplexNumberDescription`). Los slots de estas clases almacenan instancias de la clase `RealNumber`, que aparecen en la parte derecha. En estas instancias de `RealNumber` aparecen, como valores de los slot `hasNumericalValue`, los tipos de datos float con los que se almacenan los números reales en java/Protégé.

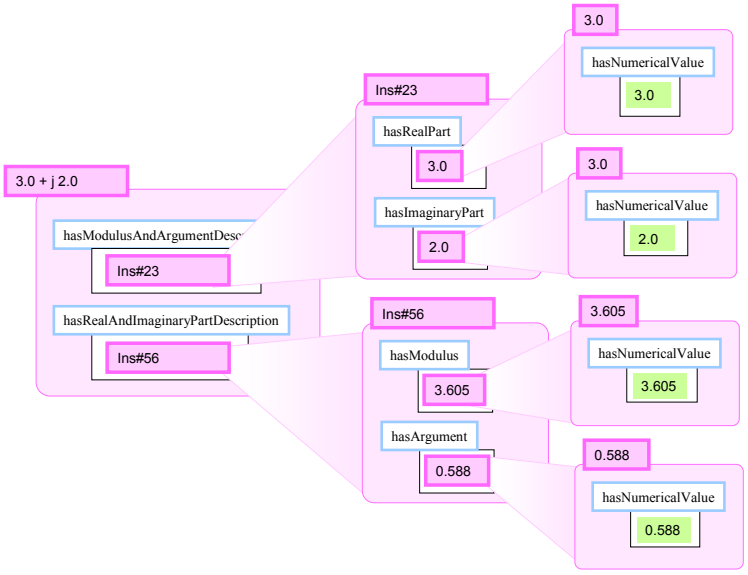


Figura 4.7. Instancia de número complejo 3+j2

[82] http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.2.3

La figura 4.8 es una captura del editor Protégé en el que se puede ver un esquema similar al presentado en la figura 4.7 pero con instancias reales de la ontología.

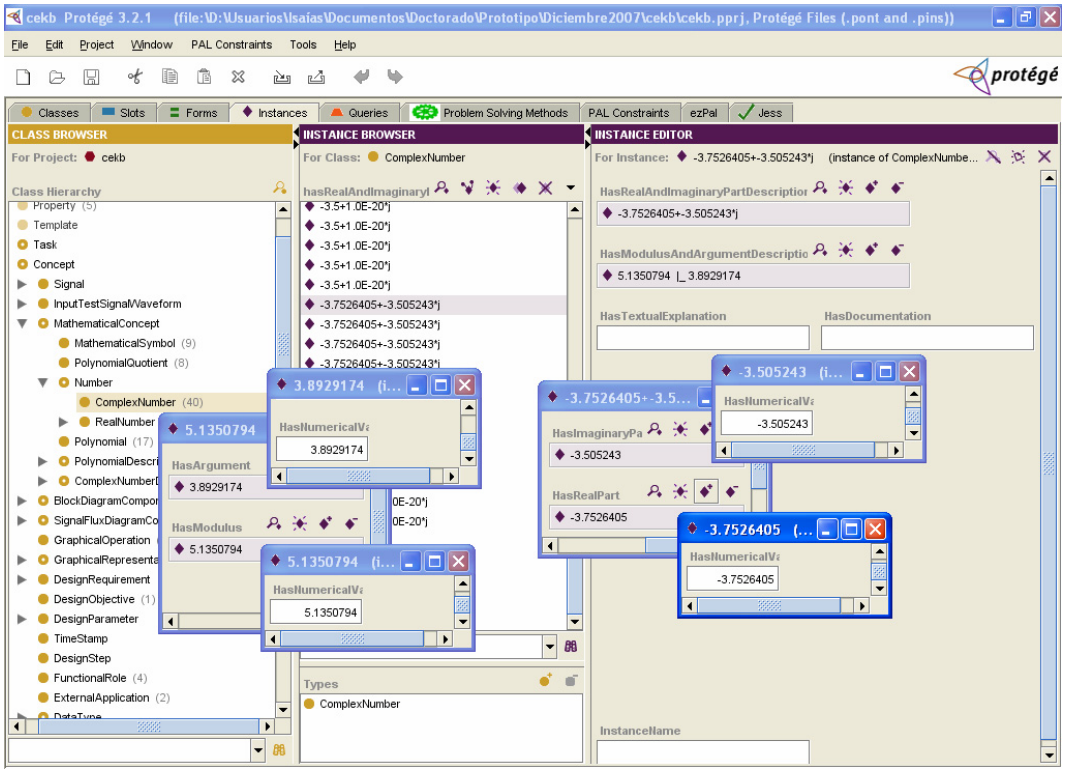


Figura 4.8. Instancia de número complejo en la ontología

Tanto `RealNumber` como `ComplexNumber` se han creado como subclases de la clase `Number`.

En este punto es conveniente comentar algo sobre el uso de las instancias que representan números en la ontología. La misma instancia de un número puede utilizarse en varios sitios siempre que haga referencia a un mismo individuo ó cuando aparezca solamente con el fin de procesar su valor numérico, por ejemplo en las comparaciones, donde se comparan valores numéricos y no instancias de números estrictamente hablando. Pero por otro lado las raíces de un polinomio, por ejemplo, siempre serán instancias de números complejos diferentes aunque tengan los mismos valores como parte real e imaginaria (incluyendo el 0+0j)[83], es

[83] El formalismo de marcos postula la presunción de nombre único (Unique Name Assumption - UNA -). Esto significa que no puede existir una instancia con dos nombres diferentes ni dos instancias diferentes con el mismo nombre. Esto no

decir, cada raíz es un individuo independiente y por lo tanto tendrá una instancia que lo represente.

4.5.1.3 Las expresiones compuestas

Dentro del dominio se encontrarán expresiones matemáticas que involucran a cantidades representadas mediante números. En la presente ontología se ha decidido hacer una conceptualización muy básica para estas expresiones con la finalidad de concentrar los esfuerzos de conceptualización en los aspectos más relevantes. Una expresión matemática será tratada como una cadena de caracteres en la que se supone que la notación empleada es la infija. La expresión resultante será evaluada por una aplicación externa de cálculo numérico de acuerdo a estas suposiciones. A estas expresiones se las denominará "expresiones compuestas" y serán instancias de la clase `CompoundExpression`. La nomenclatura empleada para los operadores matemáticos es básica y consiste en el siguiente conjunto de símbolos: {+, -, *, /, ^, sqrt, log, ln, e, sin, cos, tan, asin, acos, atan, (,)}

Una alternativa mejor sería conceptualizar los distintos operadores matemáticos así como las posibles expresiones. Para ello se pueden crear conceptualizaciones basadas en representaciones del tipo MathML[84] o similares. De esta forma se podría hacer que las expresiones pudiesen ser evaluadas por cualquier herramienta de cálculo numérico que "conozca" esa estructura de representación. Además, se podría comprobar que las expresiones están bien formadas y sería posible también ofrecer explicaciones sobre la expresión así como efectuar razonamientos cualitativos sobre la misma. La mayor complejidad de este tipo de representación hizo que se descartara su implementación en la presente ontología, con el fin de centrar el estudio en los conceptos de control.

Los únicos elementos que, dentro de la presente conceptualización, sí presentan cierto contenido semántico dentro de una expresión son las variables, que aparecerán en la expresión como cadenas de caracteres (sin poder utilizar los caracteres reservados a símbolos de operadores o funciones matemáticas mencionados anteriormente). Cada expresión tendrá asociada la traducción de estas variables a los valores correspondientes. Estos valores, a su vez, harán referencia a las características de los diferentes conceptos (sistemas, funciones de transferencia, polinomios, etc.) que se hayan definido en la ontología.

---

supone dificultad en la conceptualización de los números tal como se ha realizado porque lo que denomina "nombre" de la instancia es en realidad un identificador interno generado automáticamente por Protégé.

[84] http://www.w3.org/Math/

Un ejemplo de expresión compuesta puede ser la siguiente (la expresión no corresponde a ninguna que se use en la práctica del control, es sólo a título de ejemplo):

"(1+2*type_of_plant-order_of_plant)"
donde: " type_of_plant" representa al concepto "tipo de la planta" (type of plant).
" order_of_plant " representa al concepto "orden de la planta" (order of plant).

La expresión compuesta constará, por tanto, de la cadena de caracteres expresando la operación matemática en notación infija más una serie de traducciones de los nombres de variables a las características que representan. La correspondencia entre variable y la característica se representa mediante otra estructura conceptual denominada "enlace a variable" o "variable binding" (clase `VariableBinding`) que, a su vez, contará con la descripción del nombre de la variable y la característica a la que está asociada o enlazada. En la figura 4.9[85] se observa gráficamente el ejemplo propuesto.

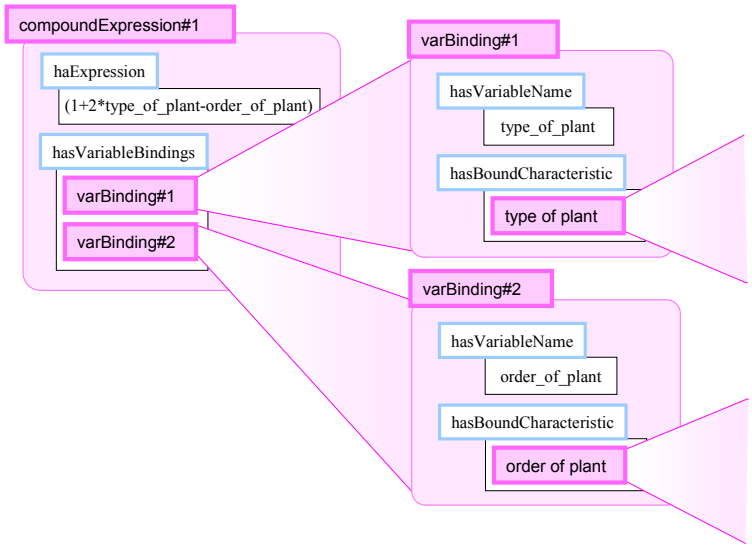


Figura 4.9. Instancia de la clase `CompoundExpression`.

Las características no sólo se dirán directamente de los diferentes sistemas que entran en juego en el problema, como es el caso del ejemplo presentado, sino que pueden referirse a cualquier elemento susceptible de tener una característica.

85 Todas las instancias que aparecen en los slots de otras instancias podrían ir representándose hasta llegar al nivel en el que las instancias no tienen a otras instancias en sus slots. Esto se hará en algunas ocasiones pero, en este ejemplo y en otros, no se hace toda esta "expansión" de las instancias para mejorar la legibilidad

Pueden aparecer enlaces a características de entidades complejas como la siguiente:

> "a" -> parte real de (tercer elemento de (polos reales ordenados de forma ascendente por su parte real) de planta)

La forma de expresar estas características y entidades (polos reales, el tercer elemento de una lista ordenada, etc.) se verá en las siguientes secciones. Además, se volverá a hablar de la clase `CompoundExpression` y `VariableBinding` en la sección 4.5.4.

El procesamiento de la estructura representada en una instancia de `CompoundExpression` consistirá en resolver las características a las que están enlazadas las variables y, posteriormente, evaluar la expresión. Este proceso puede ser llevado a cabo por un procesador programado a tal efecto (así es como se ha hecho en el presente trabajo, describiéndose el procesamiento en el siguiente capítulo) o bien la estructura de la expresión puede traducirse a un sistema de reglas de producción con capacidad para la representación de expresiones matemáticas y ser el motor de inferencia de reglas el que resuelva las variables enlazadas a características y evalúe la expresión. En concreto el sistema JESS (Java Expert System Shell) (Friedman-Hill, 2007) puede ser utilizado para este cometido.

4.5.2 Conceptualización de los modelos de sistemas y topologías de control

4.5.2.1 Los modelos de sistemas

En ingeniería de control se utilizan diversos modelos matemáticos para estudiar los sistemas, de forma que se puede crear una jerarquía de modelos y caracterizarlos respecto a sus particularidades. Los diferentes modelos creados se construyen a base de simplificaciones y/o transformaciones sobre modelos anteriores, con el fin de reflejar los aspectos relevantes del sistema o de facilitar las tareas de análisis y diseño.

La teoría clásica de control utiliza el denominado modelo en función de transferencia y, por lo tanto, este concepto será la base de la conceptualización en el dominio de estudio elegido[86], reflejándose en la clase

[86] La conceptualización, idealmente, podría recoger los diferentes niveles y modelos desde el modelo en función de transferencia hasta el propio sistema físico, con lo que en el proceso de diseño podría razonar usando los resultados no sólo de la simulación con el modelo en función de

`TransferFunctionSystemModel`. Este modelo se obtiene aplicando la transformada de Laplace sobre el sistema de ecuaciones diferenciales (lineales o linealizadas en torno a un punto de equilibrio) y ofrece, en forma de un cociente de polinomios de coeficientes reales, la relación entre la salida y la entrada del sistema, es decir, el comportamiento dinámico del mismo. Gráficamente, el sistema suele representarse como un bloque que tiene una entrada y una salida y en el que se muestra la función de transferencia o una letra o expresión que la representa (figura 4.10).

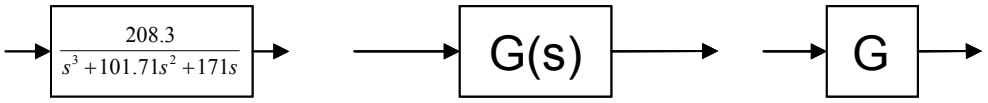


Figura 4.10. Instancia de la clase `CompoundExpression`.

En la conceptualización se incluye el concepto genérico "modelo de sistema" (clase `SystemModel`) como el de más alto nivel para mantener abierta la representación dejando espacio a la inclusión de los diferentes modelos utilizados en ingeniería de control. Todos estos modelos, incluido el modelo en función de transferencia, serán, por tanto, un subtipo[87] del concepto modelo de sistema. No existe una serie de propiedades que los modelos compartan, ya que cada uno puede estar basado en estructuras diferentes.

El comportamiento dinámico del sistema (de acuerdo a este modelo) viene totalmente definido por medio de la función de transferencia y por lo tanto el concepto tendrá como elemento constitutivo esa función de transferencia (que, a su vez, es un cociente de polinomios tal como se ha dicho). Por lo tanto la clase `TransferFunctionSystemModel` tiene como slot principal a `hasPolynomialQuotient`.

4.5.2.2 El esqueleto conceptual básico

Uniendo las anteriores conceptualizaciones se puede representar el esqueleto básico formado por las clases más relevantes del dominio de las matemáticas que

transferencia sino la simulación con el modelo de parámetros agrupados o incluso con las pruebas reales en el sistema.

[87] La relación tipo-de (type-of) o subtipo-de (subtype-of) (también llamada "is-a", de generalización, de subsunción, de categorización o de herencia) es una de las principales relaciones que se pueden encontrar en cualquier dominio. Es la relación que forma la columna vertebral del paradigma de orientación a objetos Es también la relación que mejor manejan los formalismos de representación basados en lógicas descriptivas, automatizando el proceso de clasificación de un concepto como subconcepto de otro siempre que este concepto pueda definirse de manera intencional (mediante el establecimiento de condiciones necesarias y suficientes).

se utilizan para describir el modelo en función de transferencia. En la figura 4.11 se representa esta estructura básica. Las flechas indican el tipo de conceptos que pueden ir en los slots, es decir, unen una clase con un slot, de forma que el rango de ese slot son las instancias de esa clase.

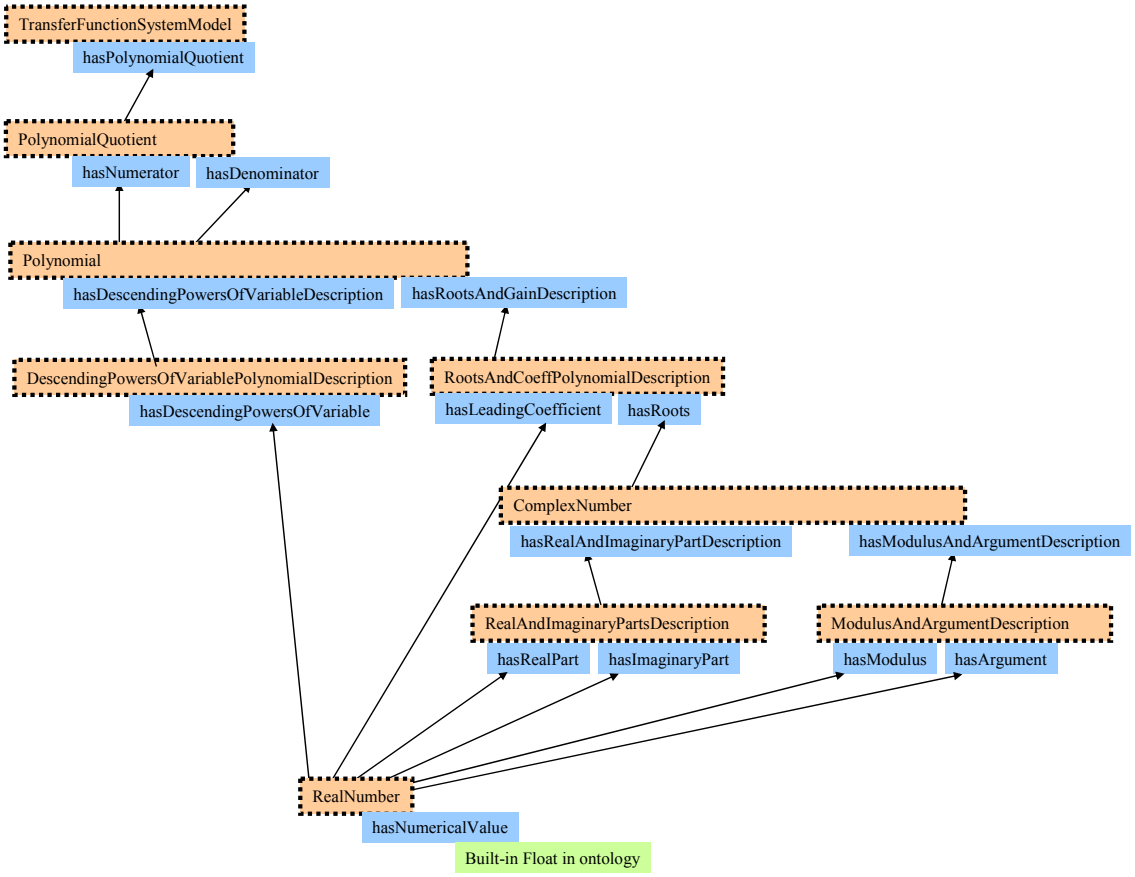


Figura 4.11. Estructura básica de conceptos.

A partir de esta estructura básica se crearán los demás conceptos pertenecientes al campo del control y su lenguaje. Los slots se han utilizado para representar la información esencial de cada concepto, es decir, los atributos (propiedades) esenciales. Los demás atributos que se pueden asociar a los conceptos, como por ejemplo las características cuantitativas y cualitativas, no serán representados como slots de esos conceptos sino como una predicación[88] de los mismos almacenada en una estructura asertiva separada. Esta decisión, además de ser

[88] Se usa el término "predicación" como traducción del término inglés "predication" utilizado en representación del conocimiento. Podría usarse "afirmación" como término con sentido similar. Lo que se pretende transmitir es la idea de "lo que se dice de algo" y como tal se toma el significado original que Aristóteles dio al término "predicatio" y que a menudo se utiliza también en lógica matemática como "predicación lógica".

coherente con ciertos estudios sobre la naturaleza de los atributos de los conceptos (ver párrafos siguientes), también permite mantener la conceptualización de los conceptos lo más sencilla, comprensible y reutilizable posible, así como facilitar la propia descripción de la estructura de las características, aspecto de gran importancia en este dominio.

Los slots que están asociados a cada concepto son, por tanto, aquellos que sirven para describir lo que hace que ese concepto sea lo que es, aquellos que lo describen totalmente. En el dominio aquí tratado esto significa describir cómo está formado o compuesto ese concepto. Los atributos o propiedades de los conceptos que son específicos de un dominio son los que se han separado de la descripción "esencial" y se han colocado en una estructura separada. Esta aproximación a la descripción o definición de conceptos distinguiendo la esencia (o atributos esenciales) de otros atributos accesorios parte de los estudios filosóficos de Aristóteles (Smith, 2004), aunque aquí se toma con una visión más pragmática y no se persigue encontrar la "verdadera definición de las cosas".

En el caso de un dominio artificial correspondiente al ámbito científico, como el que se trata, la esencia se asimilará a la descripción de composición de conceptos que definen a uno dado. Como se resalta en (Burek, 2004) "en muchos dominios, incluyendo los técnicos y científicos, las definiciones no intentan proporcionar la esencia de los conceptos. Parece que los autores de esas definiciones no intentan señalar a ninguna propiedad fundamental de los mismos. En estos casos, si se sigue estrictamente la sugerencia [de separar características esenciales de las que no lo son] muchos conceptos no tendrían definiciones en el TBox y todo el conocimiento afirmado sobre ellos estaría contenido en el ABox, donde las características no esenciales, pero que sí son definitorias estarían mezcladas con el conocimiento puramente afirmativo"[89]. En (Burek, 2005) se postula distinguir entre atributos (propiedades) esenciales y no esenciales, pero teniendo en cuenta en el caso de estas últimas aquellas que son definitorias para el concepto y las que no. La aproximación seguida en la presente conceptualización sigue una línea similar aunque, como se ha mencionado aquí, se asimilan las propiedades esenciales con las que definen la composición de los conceptos que parece el aspecto más importante en el caso de esta disciplina.

Así, un polinomio tiene como descripción esencial sus coeficientes o bien sus raíces y su coeficiente principal, mientras que características que se pueden aplicar al polinomio, como el grado, no pertenecen a su esencia ya que, por ejemplo, pueden ser calculadas a partir de la información esencial. Por tanto, el

[89] La separación entre conocimiento esencial y no esencial se sigue en mayor o menor medida en varios sistemas basados en lógicas descriptivas donde la TBOX o red terminológica contendría la información esencial (las propiedades necesarias y suficientes que definen a los conceptos) y la ABOX o red asertiva contendría la información no esencial sobre los conceptos. En CLASSIC, por ejemplo, se utilizan reglas para especificar predicados con propiedades no esenciales (Brachman et. al., 1991).

coeficiente principal y las raíces aparecerán como slots del concepto, mientras que el grado no. El grado será una característica que tendrá una conceptualización propia consistente en la descripción de la forma de ser calculado. Además, existirá una estructura asertiva en la que aparecerá el grado como una relación entre un polinomio concreto y el valor de esta característica para el mismo. La generación de este concepto en la estructura asertiva se hará automáticamente a partir de la conceptualización de la característica "grado".

4.5.2.3 Los diagramas de bloques y flujos de señal

El diseño y construcción de un esquema de control supone crear un sistema nuevo que, incluyendo al sistema inicial que se pretende controlar, tenga una dinámica que se ajuste a las especificaciones deseadas. Para conseguir este objetivo se utiliza una combinación de estas dos técnicas:

- Diseñar e introducir un nuevo sistema, denominado controlador.
- Modificar la topología de flujo de las señales diseñando e incluyendo un bucle (habitualmente de realimentación) de forma que se genere una señal de error que, junto con el controlador, permita conseguir el comportamiento deseado.

Para poder diseñar y representar la topología de flujo de señales se utiliza un modelo en el que plasma la interconexión de los modelos en función de transferencia que entran a formar parte de la configuración. Esta representación es el denominado diagrama de bloques, que cuenta con una representación equivalente en el denominado diagrama de flujo de señal. Estos diagramas son específicos de la disciplina del control automático y hacen énfasis en la representación de los flujos de información y el tratamiento de los sub-sistemas como bloques entrada-salida interconectados.

La configuración (colocación) del controlador y elección de la topología son dos aspectos que forman parte de las decisiones de diseño habituales al crear un sistema de control. Los procesos de diseño que llevan a la definición de la dinámica adecuada del controlador variarán en función de su configuración y de la topología en la que se encuentre por lo que será necesario incluir la topología dentro de la conceptualización (aunque en esta ontología sólo se contemple una topología). La figura 4.12 muestra un ejemplo de topología (realimentación negativa) y de configuración del controlador (en cascada o en serie con el sistema a controlar). Ésta es, además, la topología en la que se centrará la conceptualización realizada en la presente ontología (con la salvedad de que se considerará realimentación unitaria, esto es, H=1)

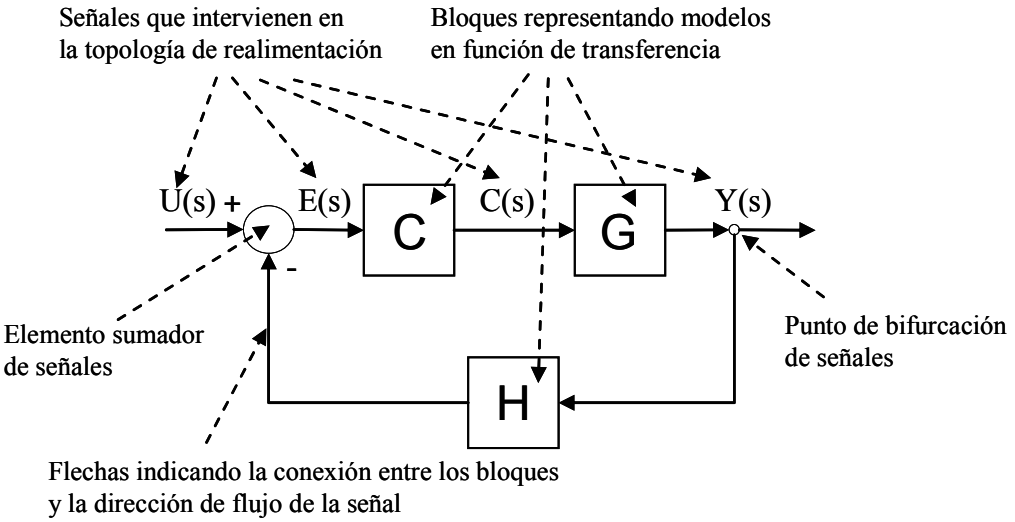
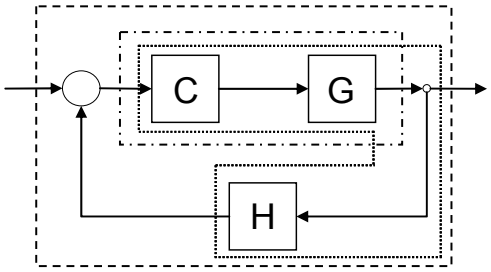


Figura 4.12. Bloques y señales en una topología con el compensador en serie con la planta y sin considerar perturbaciones.

En la topología también aparecen otros elementos importantes además de los bloques que recogen los modelos en función de transferencia:

- Enlaces o conexiones entre bloques, que están etiquetados con el nombre de la señal que fluye por ellos.
- Sumador de señales, elemento que compara la señal de referencia con la salida del sistema.
- Punto de bifurcación, elemento sin comportamiento que sirve para crear varios caminos de señal a partir de uno.

Además, el diagrama de bloques es también la base para definir nuevos bloques y por lo tanto nuevos subsistemas relevantes a la hora de realizar las tareas de análisis y diseño. En la figura 4.13 se pueden apreciar estos bloques.



	Nombre	Función de transferencia
Bloques	Sistema a controlar	G
	Controlador	C
	Sensor	H
Bloques calculados	Sistema en cadena abierta	$C \cdot G \cdot H$
	Sistema en cadena directa	$C \cdot G$
	Sistema total ó Sistema en cadena cerrada	$\frac{G \cdot H}{1 + C \cdot G \cdot H}$

Figura 4.13. Bloques que intervienen en la topología.

Los bloques denominados "calculados" tendrán una función de transferencia propia que, en vez de venir dada, se obtendrá mediante alguna de las siguientes operaciones que pueden realizarse:

- Cálculo del resultante de varios bloques en serie.
- Cálculo del resultante de varios bloques en paralelo.
- Cálculo del resultante de bloques incluidos en una topología que involucra bucles de realimentación positiva y/o negativa.

En todos los casos las operaciones requeridas son sumas y productos de funciones de transferencia (de polinomios en último término). En el caso de dos bloques en serie o paralelo la función de transferencia total puede calcularse de forma sencilla. Conceptualmente se representará la operación indicando los bloques que entran en juego y el tipo de operación (serie o paralelo) de que se trate. En el caso de involucrar lazos de realimentación el cálculo de la función de transferencia total equivalente suele hacerse mediante la fórmula de Mason. La conceptualización en este caso deberá recoger las señales entrada y salida del bloque equivalente total que se pretende calcular. Las señales a su vez están conceptualizadas en asociación con los enlaces entre bloques tal como se explica en los siguientes apartados.

A continuación se presenta la conceptualización realizada para representar los bloques y los diagramas en los que éstos aparecen.

Conceptualización de la topología y bloques

Es necesario tener la posibilidad de que exista una representación de las topologías y configuraciones de control, así como de los bloques que la conforman, sin que existan datos concretos sobre los sistemas que aparecen en ellas (las reglas de diseño se crearán hablando de los bloques en la topología de forma genérica).

La solución adoptada ha pasado por la conceptualización de bloques denominados "canónicos" que representan a los diferentes elementos que aparecen en un diagrama de bloques, incluyendo los que representan a los sistemas involucrados. Estos bloques canónicos que representan a los sistemas no tienen inicialmente ningún modelo en función de transferencia asociado sino solamente un nombre para identificarlos y hacer referencia a ellos en el resto de estructuras de la ontología. Cuando aparezca un problema concreto deberá establecerse una relación de mapeo entre estos bloques canónicos y las funciones de transferencia concretas (instancias de `TransferFunctionSystemModel`).

La clase que describe a los bloques canónicos se denomina `CanonicalBlockDiagramComponent`[90] y contiene slots para indicar el nombre o alias que se le da al bloque dentro de una determinada topología (controlador, sistema a controlar, etc.) y el tipo de bloque (función de transferencia, sumador, etc.) que representa. No existirán bloques si no es dentro de una topología. En la figura 4.13 se pueden ver ejemplos de bloques canónicos correspondientes a la topología de la figura 4.12.

Los bloques que aparecen en una topología concreta serán pues instancias de la clase `CanonicalBlockDiagramComponent`. Este uso de instancias en el formalismo de marcos se corresponde en cierta medida con los *objetos o instancias prototípicos* (también llamadas *roles*) de UML (Booch et. al., 2005). Allí estos elementos se utilizan para construir diagramas de interacción en los cuales se representa al conjunto de objetos de una determinada clase de forma genérica[91].

La conceptualización de una topología incluye la especificación de los bloques que forman parte de la misma, tal como se ha explicado, y la forma en la que están conectados estos bloques (incluyendo las señales que fluyen por las conexiones entre los mismos). Las clases que representan a las topologías como diagrama de bloques y como diagrama de flujo se han denominado `CanonicalBlockDiagram` y `CanonicalSignalFluxDiagram` respectivamente.

Para conseguir describir la disposición de los bloques en un diagrama es necesario, de forma general, utilizar tanto la mereología como la topología. La mereología estudia la descripción de los sistemas u objetos que están compuestos de otros, es decir, estudia la relación entre el todo y sus partes. Por su parte, la topología estudia la forma en la que se establecen las conexiones o la disposición de las partes dentro del todo. En medicina y muchas ramas de la ingeniería estas relaciones son muy importantes, más incluso que la relación subtipo[92]. Sin embargo, en el caso de la conceptualización de la teoría clásica de control, tal como se ha postulado para el presente estudio, la representación de las relaciones

[90] El bloque canónico representa a cualquier elemento que en un diagrama de bloques aparece conectado por las flechas. Por tanto representará tanto a bloques función de transferencia como a sumadores y puntos de bifurcación e incluso a sumideros y generadores de señal necesarios para mantener la coherencia de la representación de las topologías como se verá más adelante.

[91] El uso de una instancia para representar al conjunto de instancias de una determinada clase es también una solución que se ha dado a ciertos problemas de representación del conocimiento desde el ámbito de la Web Semántica (Uschold y Welty, 2005). El problema en este caso surge ante la necesidad de utilizar clases como valores de las propiedades. Este hecho haría que el lenguaje a utilizar pasase de ser decidible y tratable a ser indecidible (se pasaría de OWL-DL a OWL-Full). Para mantener el lenguaje en la variante de lógicas descriptivas, una de las estrategias consiste precisamente en utilizar una instancia como representante genérica de la clase (la aproximación denominada "approach 2" en (Uschold y Welty, 2005)).

[92] La creación del lenguaje SysML (ver apartado 2.2.2) ha tenido muy en cuenta este hecho. Uno de los aspectos más importantes en la construcción de este lenguaje de modelado a partir de la especificación 2.0 de UML fue la práctica desaparición de la relación subtipo (clase-subclase) y la construcción en el lenguaje de estructuras apropiadas para reflejar la relación parte-de cómo una relación básica en la descripción de modelos en ingeniería de sistemas.

mereológicas y topológicas no es demasiado problemática. El suponer sistemas de una sola entrada y una sola salida (SISO) en el diagrama de bloques hace que la forma en la que éstos se conectan para formar el todo esté definida por las uniones que existen entre ellos, de forma que para cada bloque de tipo función de transferencia sólo existirá una flecha saliente y otra entrante. La topología, al contrario que en dominios como la ingeniería mecánica (por poner un ejemplo), no es muy compleja en este caso. Lo que importa es qué bloque está conectado con qué otro, pero no sus posiciones relativas.

La suposición de sistemas SISO hace también que no sea necesario introducir el concepto de puerto. El enlace es tratado como una relación binaria dirigida, especifica el flujo de la señal entre una salida de un bloque y una entrada del siguiente. Cada enlace representa una señal de las que fluyen a través de los bloques en la topología.

Para mantener la coherencia de la representación (para que todos los enlaces tengan un bloque de salida y uno de entrada) se necesitan dos elementos (bloques) que habitualmente no aparecen en la representación gráfica de un diagrama de bloques:

- Una fuente de señal, que no tendrá entrada y tendrá una salida.
- Un sumidero de señal, que tendrá una entrada y no tendrá salida.

La topología canónica se describirá diciendo qué bloque está conectado con qué otro bloque. Cada uno de estos bloques será una instancia de la clase denominada `CanonicalBlockDiagramComponent`. Para representar la topología de la figura 4.12, que es la que se trata en esta conceptualización, se necesitarán las instancias canónicas de la figura 4.14

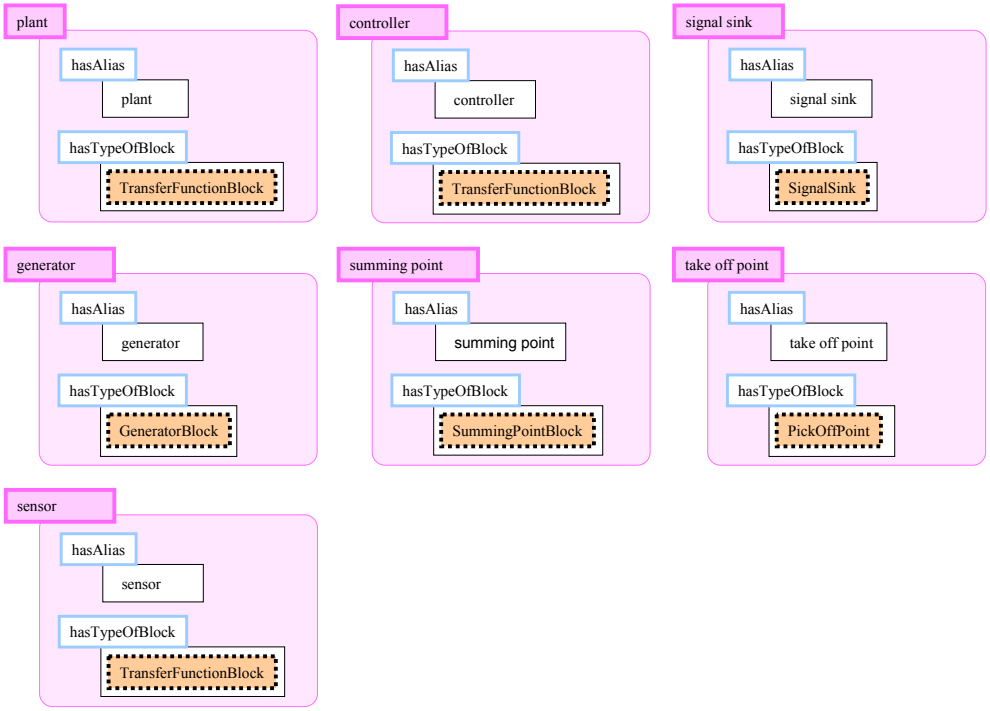


Figura 4.14. Instancias representando a los bloques en la topología de control.

Las instancias `plant` y `controller` son las instancias canónicas representantes de los sistemas físicos de las que se habló anteriormente. Es el esquema de representación de más bajo nivel que hay en la ontología. Representan, mediante el uso de un alias (la cadena de caracteres que aparece en el slot `hasAlias`) a un sistema real. Mediante el slot `hasTypeOfBlock` se especifica el rol que juega el bloque canónico, es decir, si actúa como un bloque que contiene una función de transferencia, si es un sumador, un punto de bifurcación, etc.

Además, para representar la topología, se necesitarán enlaces que digan qué bloque está conectado con qué otro tal como se ha explicado. Los enlaces se representan mediante relaciones binarias dirigidas conceptualizadas como clases, ya que llevarán información propia del enlace (la señal que fluye por él por ejemplo). En la figura 4.15 se representan las instancias de estas relaciones para el caso de la topología de la figura 4.12. En la instancia `link#5`, que corresponde al enlace entre el punto de bifurcación y el sumidero de señal, se han expandido las instancias que aparecen en los slots. De esta forma puede verse cómo es la estructura de conocimiento creada. Las instancias de bloques involucradas ya se han visto anteriormente. La señal se representa mediante una instancia cuya clase tiene dos slots que son cadenas de caracteres y sirven para identificarla. Si se

incluyese en la ontología el sistema físico completo estas señales tendrían una relación con una magnitud física determinada.

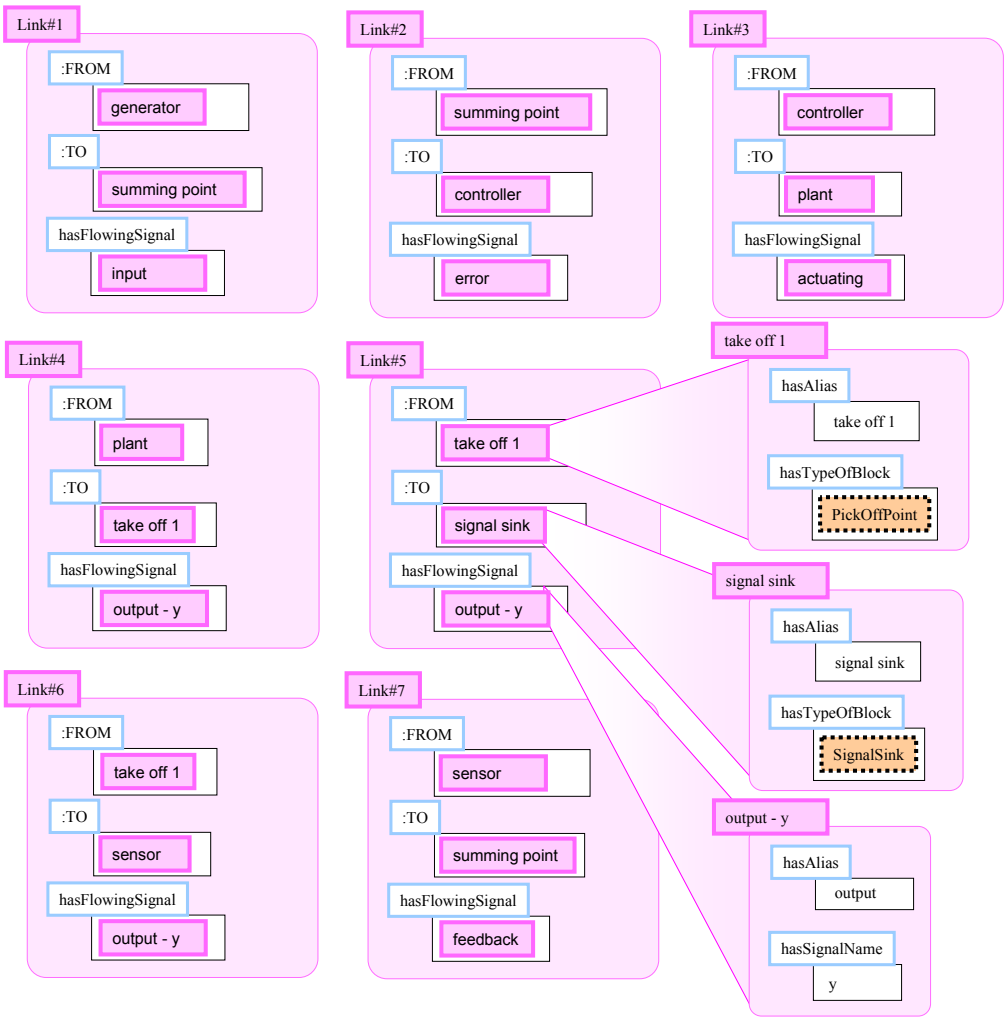


Figura 4.15. Instancias representando a los enlaces en la topología de control.

La clase que representa los enlaces se denomina `CanonicalBlockConnectionLink` y se ha implementado como subclase de la clase preconstruida :DIRECTED-BINARY-RELATION de Protégé. La ventaja de hacer esto reside en el tratamiento interno que Protégé ofrece para las subclases de esta clase, sobre todo a la hora de mostrar las relaciones gráficamente. Los slots :FROM y :TO están heredados de esta clase y contendrán respectivamente la instancia (de `CanonicalBlockDiagramComponent`) de la que parte la relación y la instancia (también de `CanonicalBlockDiagramComponent`) a la que llega la relación.

Cada enlace entre bloques lleva también asociada una señal. Mediante esta señal se podrán definir bloques calculados como los descritos en los párrafos anteriores. También servirán para crear el diagrama de flujo de señal correspondiente.

El diagrama de flujo de señal puede verse como una notación diferente para el diagrama de bloques y no como un modelo diferente para representar los sistemas y la topología. Este diagrama suele utilizarse para efectuar cálculos sobre el grafo que representa el diagrama de bloques del sistema y ese es también el uso que tendrá en la ontología. El diagrama de flujo de señal se ha conceptualizado mediante una clase denominada `CanonicalSignalFluxDiagram` y tiene slots para representar arcos y nodos. Los nodos serán instancias de una clase `CanonicalFluxDiagramNode` que a su vez contará con un slot para representar la señal asociada al nodo. Por su parte los arcos son relaciones binarias como las descritas anteriormente, pero uniendo ahora nodos y contando con un slot para representar la ganancia del arco.

La traducción de diagrama de bloques a diagrama de flujo se realiza mediante una llamada a una función externa.

4.5.2.4 Situación de la conceptualización hasta el momento

La conceptualización realizada hasta el momento describe, por un lado, la topología de control sobre la que se realizarán las tareas de análisis y diseño. Esta topología tiene como elementos constituyentes los bloques que representan a los sistemas que entran en juego. Estos sistemas se representan mediante instancias prototípicas que contienen un nombre que servirá para identificarlos. En la topología objeto de estudio los dos sistemas que aparecen son "controlador" (`controller` en la ontología) y "sistema a controlar" (`plant` en la ontología). El sistema "sensor" se supondrá de ganancia igual a 1 y sin dinámica y por lo tanto no entrará en juego en las tareas de análisis y diseño.

Por otro lado, se ha presentado también la conceptualización de los elementos matemáticos algebraicos que se utilizan para construir los modelos en función de transferencia, reflejados en la figura 4.11.

El modelo utilizado para el estudio de un sistema es a la vez una simplificación y un sucedáneo de dicho sistema físico. Cuando se realiza el análisis o el diseño se suelen hacer referencias a características de los sistemas involucrados sin mencionar el modelo que se está utilizando (el cual se supone). Todo lo que se diga debería estar referido al modelo utilizado, aunque la noción del sucedáneo se lleva hasta el extremo y se utilizará el nombre del sistema real. De esta forma no se dice:

*Los **polos** del **modelo en función de transferencia** del **sistema a controlar***

Sino:

*Los **polos** del **sistema a controlar***

Esta característica es común en las expresiones de ingeniería de control: se habla del sistema cuando en realidad debería hablarse del modelo correspondiente. Desde el punto de vista de la práctica de la disciplina este hecho es totalmente comprensible ya que de hablar con total propiedad las expresiones serían demasiado complejas. Desde el punto de vista de la ontología este hecho es relevante para las etapas de adquisición y mantenimiento del conocimiento, ya que si la conceptualización no recoge esta forma de simplificación al referirse a los conceptos habría que introducir expresiones muy largas.

En la ontología este aspecto en el uso del lenguaje de control se ha reflejado en la existencia mediante la utilización por un lado de los nombres (slot `hasAlias`) de los bloques canónicos, que representan a los sistemas de forma genérica y, por otro, con una relación de mapeo entre estos bloques y las funciones de transferencia. Además, la ontología debe permitir que las expresiones hagan referencia a esos bloques canónicos durante la fase de adquisición del conocimiento, pero debe existir un mecanismo que traduzca esas expresiones para que se refieran al modelo en función de transferencia del sistema en vez de al sistema directamente. Así, se permite que se pueda introducir la expresión "la característica c1 del sistema a controlar", debiendo ser traducida a "la característica c1 del modelo en función de transferencia del sistema a controlar"

El mapeo entre bloque canónico y función de transferencia es una instancia muy simple (de una clase denominada `Mapping`) con dos slots, uno para la instancia del bloque canónico (de la clase `CanonicalBlockDiagramComponent`) y otro para una instancia de `TransferFunctionSystemModel`.

A continuación se describe cómo se conceptualizan los conceptos del lenguaje de control, tomando como punto de partida las estructuras mencionadas hasta el momento.

4.5.3 Conceptualización de las estructuras del lenguaje de control automático

Como se ha mencionado en la sección 4.3, la ingeniería de control cuenta con un lenguaje propio que se construye sobre lenguajes de otras ciencias y tecnologías, sobre todo de las matemáticas. Los conceptos matemáticos pasan a tener nuevos nombres al ser utilizados en el ámbito del control. Este hecho no es gratuito ya que, como se ha explicado, es la base para la comunicación adecuada del conocimiento en esta rama de la ingeniería. Un ejemplo muy claro de uso del lenguaje de control es el caso del concepto "polo", nombre que se le da a cada una de las raíces del denominador de la función de transferencia del modelo matemático (en función de transferencia) de un determinado sistema.

Además de los conceptos propios del lenguaje de control, también será necesario representar otros que pertenecen al campo de las matemáticas, como es el caso del concepto "raíces" (`roots`) (que, en la ontología, es el nombre que recibe un conjunto de números complejos que aparecen formando parte de cierta forma de describir a un polinomio), "denominador" (`denominator`) (que, en la ontología, es el nombre que recibe un polinomio cuando aparece formando parte de la descripción de un cociente de polinomios), etc.

A continuación se abordará la conceptualización del mecanismo de representación en la ontología de esta forma de construir los términos matemáticos y los de control sobre estos primeros. A todos estos conceptos se les llamará "entidades" ("entities" ó, en singular, entity).

4.5.3.1 Entidades que consisten en instancias o colecciones de instancias almacenadas en slots

La mejor forma de mostrar la conceptualización de estas entidades es mediante ejemplos concretos. Uno de los conceptos más básicos del lenguaje de control que habrá que conceptulizar será el de "función de transferencia":

> *La función de transferencia es el nombre que recibe el cociente de polinomios que forma parte de la descripción del modelo en función de transferencia de un sistema.*

El concepto matemático que se encuentra debajo del de "función de transferencia" es el de "cociente de polinomios", es decir, la clase `PolynomialQuotient` descrita anteriormente. Respecto a la ontología, y teniendo en cuenta la conceptualización presentada hasta el momento, se puede decir que las instancias

de `PolynomialQuotient` que aparezcan en el slot `hasPolynomialQuotient` de la clase `TransferFunctionSystemModel` recibirán el nombre de función de transferencia (`transfer function` en la ontología).

El concepto `transfer function` no tiene existencia como tal, es sólo un nombre que se le da a una instancia de otro concepto (`PolynomialQuotient`) cuando aparece formando parte de la esencia (cuando aparece en un slot) de un tercero (`TransferFunctionSystemModel` en este caso). Es por tanto el nombre que recibe una instancia bajo determinadas condiciones, es decir, en un determinado contexto.

Una vez descrito el problema de conceptualización hay que tomar la decisión de incorporar el nuevo concepto "función de transferencia" en la ontología como una clase, como un slot o como una instancia. La elección entre una u otra opción no es trivial ni tiene una solución única y óptima.

Si se elige conceptualizar el concepto "función de transferencia" como una clase habría que tener una forma de especificar su definición en la forma en la que se ha descrito anteriormente a nivel de clase. Este hecho supone el uso de metaclases, es decir, clases que deberán llevar más información que la que habitualmente las describe con el fin de recoger esa definición. Esto significaría mayor complejidad del formalismo de representación de marcos (aunque es perfectamente posible hacerlo). El uso generalizado de metaclases es habitual en algunas ontologías basadas en marcos, como el modelo fundacional de anatomía FMA (Foundational Model of Anatomy) (Rosse y Mejino, 2003).

También hay que tener en cuenta que construir una nueva clase para "función de transferencia" sería poner este concepto al nivel de otros descritos mediante clases, es decir, al mismo nivel que el propio concepto `PolynomialQuotient`. Lo cierto es que ambos conceptos no son equivalentes sino que, como se ha dicho, "función de transferencia" es un nombre que reciben ciertas instancias de `PolynomialQuotient` cuando aparecen en la descripción de otro concepto. Si se hiciese una clase, habría instancias exactamente iguales que tendrían clases diferentes ya que cada instancia no puede serlo de dos clases diferentes[93]. Podría hacerse la mezcla de las clases `PolynomialQuotient` y una clase hipotética `TransferFunction` creando una clase hija de esas dos (la herencia múltiple sí es un mecanismo habitual en las ontologías basadas en marcos) pero esto se suele hacer cuando ambas clases ofrecen características o propiedades propias a la clase hija y en este caso ya se ha visto que es una

[93] En el formalismo de marcos una instancia sólo lo puede pertenecer a una sola clase. En OWL sí se permite que una instancia lo sea de más de una clase.

cuestión de contexto. Por estas razones no parece conveniente poner este concepto a nivel de clase.

La solución de conceptualizar "función de transferencia" como un slot sería en la práctica sustituir el slot `hasPolynomialQuotient` de la clase `TransferFuncitonSystemModel` por un slot `hasTransferFunction`. En este caso se perdería la separación entre el lenguaje de control y el de las matemáticas, no habría forma de utilizar el término "función de transferencia" como concepto independiente.

La tercera opción, que es la que se ha elegido, es utilizar una instancia. Aparte de la menor adecuación de clase y slot vista anteriormente, la justificación de la decisión se basa en que, de esta manera, se puede utilizar el nivel de clase para describir cómo se definen o calculan los conceptos propios de la ingeniería de control. Con esta aproximación, el nivel de las clases contendrá el esqueleto para especificar el contexto en el que se aplica el concepto de control, mientras que las instancias de estas clases serán los propios conceptos de control en los que se explicitan los diferentes conceptos.

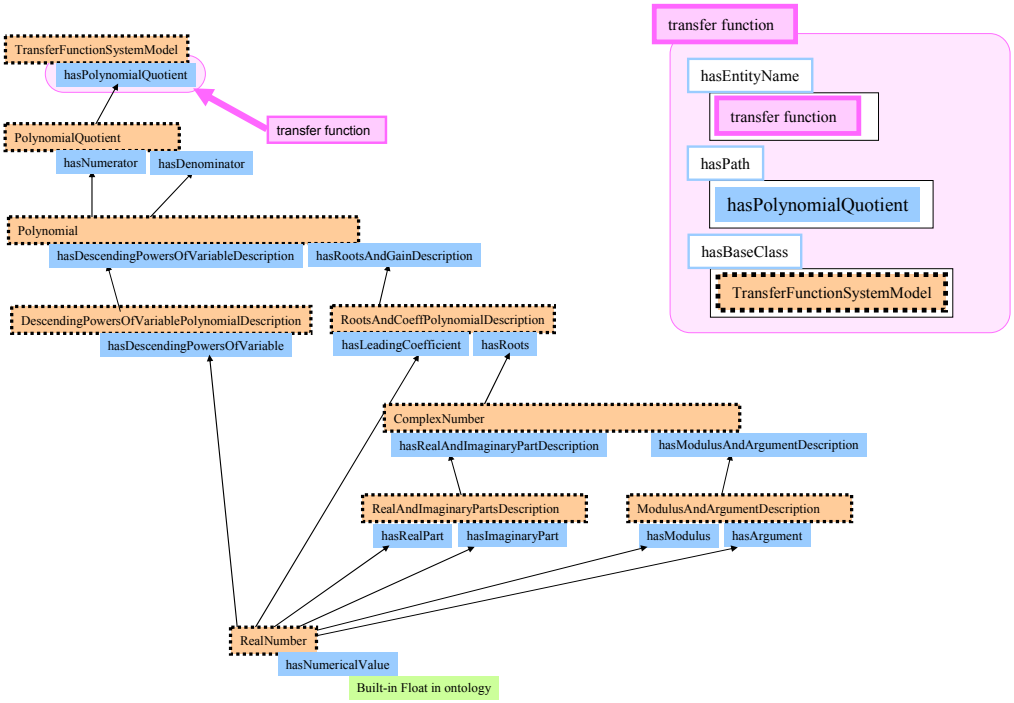


Figura 4.16. Posible conceptualización del concepto `transfer function` en base a la estructura básica de conceptos.

Tomando como ejemplo el concepto "función de transferencia" habría que poder expresar que esta instancia describe a las instancias de `PolynomialQuotient`

en un determinado contexto y que este contexto, en este caso, es cuando esa instancia de `PolynomialQuotient` aparece en el slot `hasPolynomialQuotient` de la clase `TransferFunctionSystemModel`. Bajo estas premisas se describe seguidamente una primera conceptualización posible para la clase que representa a los conceptos de control.

Esta clase tendría tres slots como información esencial: uno para el nombre del concepto de control. A este slot se le ha denominado `hasEntityName` y en el caso del ejemplo contendrá la cadena de caracteres ”transfer function”. Otro slot denominado `hasPath` (se verá el por qué de este nombre en los siguientes párrafos) se utiliza para describir el slot que contiene la instancia a la que se le va a dar ese nombre. El valor de este slot en el presente ejemplo será el slot `hasPolynomialQuotient`. En tercer lugar se tendrá un slot denominado `hasBaseClass` que será utilizado para describir la clase en la que aparece el slot especificado anteriormente (en el presente ejemplo la clase será `TransferFunctionSystemModel`). En la figura 4.16 se puede observar de forma gráfica la relación entre el concepto (la instancia) `transfer function` y su definición a partir del esqueleto conceptual básico presentado anteriormente en la figura 4.11.

En el caso de `transfer function` sólo hay un slot involucrado en la descripción del concepto pero existen conceptos en los que más de un slot puede entrar en juego. Para ilustrar este caso se utilizará un concepto proveniente del dominio del álgebra: el concepto "raices" (`roots`). La definición habitual para el concepto raíces aplicada a un polinomio es la siguiente:

> *Las raíces de un polinomio son aquellos números que, evaluados en la variable del polinomio, hacen que su valor sea cero.*

En el caso de la ontología no se conceptualizará esta descripción de raíces ya que no interesa su descripción desde el punto de vista matemático en el contexto del control. Lo que sí habrá que reflejar en la ontología es que el concepto raíces será una de las partes que describen al propio concepto polinomio. La descripción de raíces a conceptualizar en la ontología será pues:

> *Las raíces de un polinomio es el conjunto de números complejos que forma parte (junto con el coeficiente principal) de la descripción de un polinomio mediante raíces y coeficiente principal.*

Según el esquema de la figura 4.11 el concepto `roots` sería el nombre que recibe el conjunto de instancias que aparecen en el slot `hasRoots` de la instancia de `RootsAndCoeffPolynomialDescription` que está en el slot

`hasRootsAndCoeffPolynomialDescription` de cualquier instancia de `Polynomial`. Como puede observarse en este caso hay dos slots involucrados en la definición del concepto. En la figura 4.17 puede verse la representación gráfica de este hecho. Estos dos slot definen una especie de "camino" conceptual para la definición del concepto.

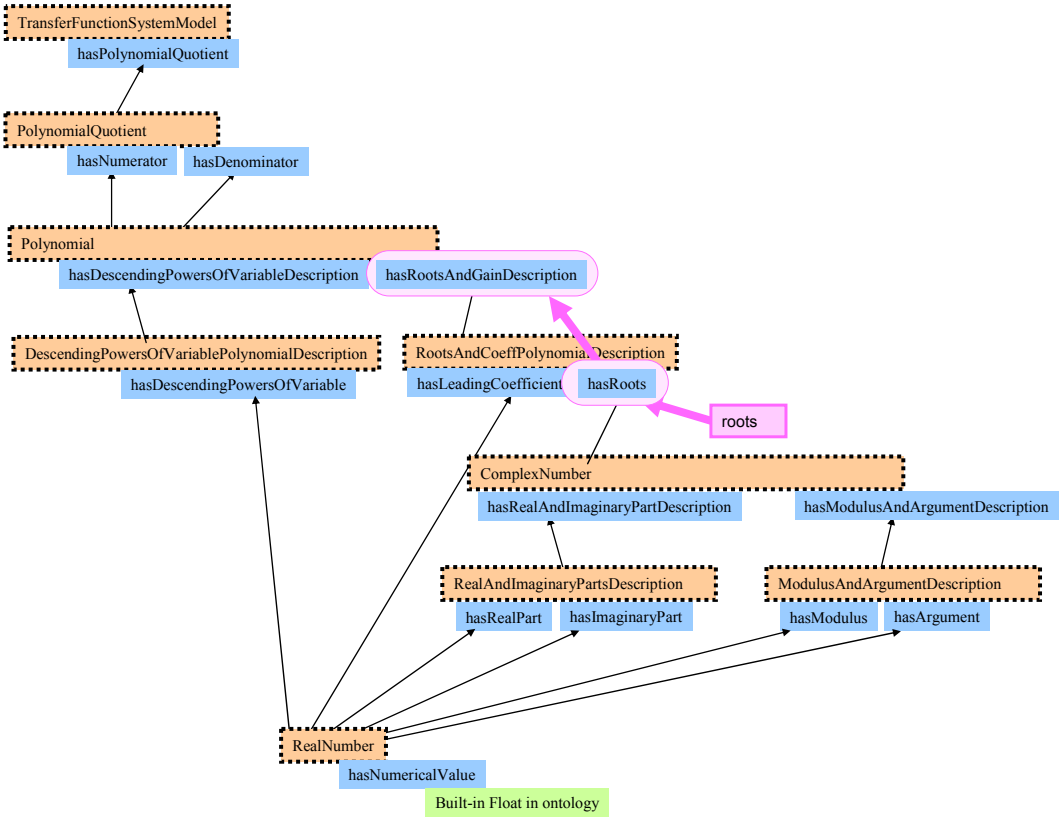


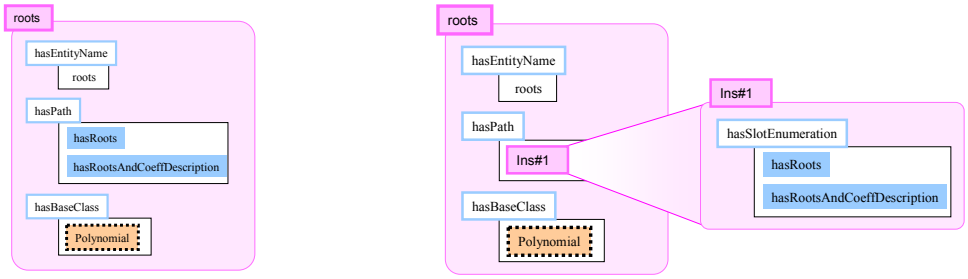
Figura 4.17. Definición del concepto `roots` en relación a conceptos ya representados.

La solución de conceptualización utilizando este camino a través de los slot es similar al denominado encadenamiento o composición de propiedades (property chaining o property composition) que aparece en algunos trabajos sobre esquemas de conocimiento en ontologías. El ejemplo típico es el de la definición de la relación `tiene_sobrino` o `tiene_tío`. Esta relación se puede definir en base a otras, en concreto en base a la de `tiene_hijo` y `tiene_hermano`. De esta forma, si la instancia1 está relacionada con la instancia2 mediante `tiene_hermano` y la instancia2 está relacionada con la instancia3 mediante `tiene_hijo`, entonces la instancia1 estará relacionada con la instancia3 mediante la relación `tiene_sobrino`. Este esquema conceptual fue eliminado

de los requerimientos de partida para la expresividad semántica de OWL[94] para evitar problemas de decidibilidad del lenguaje resultante si esa característica se implementase (Antoniou y van Harmelen, 2004b). Cuando este tipo de construcción es necesario suele implementarse utilizando reglas (en este sentido hay varios estudios sobre la combinación del uso de reglas para incrementar la expresividad de OWL (Kattenstroth, 2007), (Antoniou et. al., 2005)). También se han propuesto otras aproximaciones como el uso de macros (Vrandecic, 2005).

En el presente caso se ha decidido ofrecer una representación explícita de esta estructura (encadenamiento de propiedades o de slots hablando en términos de marcos) en la ontología. La representación de la estructura en la ontología es importante para la fase de adquisición del conocimiento y para la interacción con el usuario ya que permitirá utilizar conceptos más cercanos a los que usa el experto en el dominio al constituir la parte central de la naturaleza del lenguaje de control. La implementación práctica de la misma se podrá realizar posteriormente mediante un proceso de traducción a reglas o mediante procesamiento directo mediante un lenguaje de programación.

Como se ha visto, para definir el concepto `roots` debe utilizarse un camino de slots en el que están involucrados dos de ellos. En general, la descripción de un término mediante la conceptualización realizada necesitará poder especificar una lista ordenada de slots que definan el camino mencionado, en vez de un solo slot como se había planteado al hablar del ejemplo de `transfer function`.



(a) (b)

Figura 4.18. Definición del concepto `roots` en relación a conceptos ya representados.

Se ha creado una clase independiente para representar el camino de slots y se le ha dado el nombre `SlotPath`. La instancia `roots` utilizará pues una instancia de esta nueva clase que a su vez contendrá (en uno de sus slots) la lista de slots. Se prefiere hacer esto así a introducir directamente la lista de slots en la instancia de

[94] Esta fue una de las causas para no utilizar OWL como lenguaje tal como se mencionó en el apartado 4.4.1, aunque la versión 1.1 del lenguaje se ha planteado la inclusión de la composición de propiedades en el mismo ante la gran relevancia de esta estructura de conocimiento en muchos dominios.

roots para facilitar el procesamiento automático del camino de slots. En la figura 4.18a se representa la solución en la que la lista de slots se incluiría en la propia instancia roots; este esquema es similar al de la figura 4.16 haciendo al slot `hasPath` múltiple. En la parte 4.18b se observa la solución empleada finalmente. La instancia introducida en el slot `hasPath` de `roots` se representa expandida para comprender mejor la conceptualización.

La importancia del camino de slots y de la definición de los conceptos de control mediante esta estructura se aprecia mejor con uno de los conceptos más importantes en el ámbito de la teoría clásica de control, el concepto de "polos" (`poles`). La definición del concepto polos podría ser:

> *Los polos son las raíces del denominador de la función de transferencia del modelo en función de transferencia del sistema.*

Atendiendo a la conceptualización de los conceptos matemáticos, se tendría gráficamente que la definición del concepto "polos" se obtiene mediante la especificación de un "camino" a través de los slots que describen a esos conceptos matemáticos, tal como se ve en la figura 4.19.

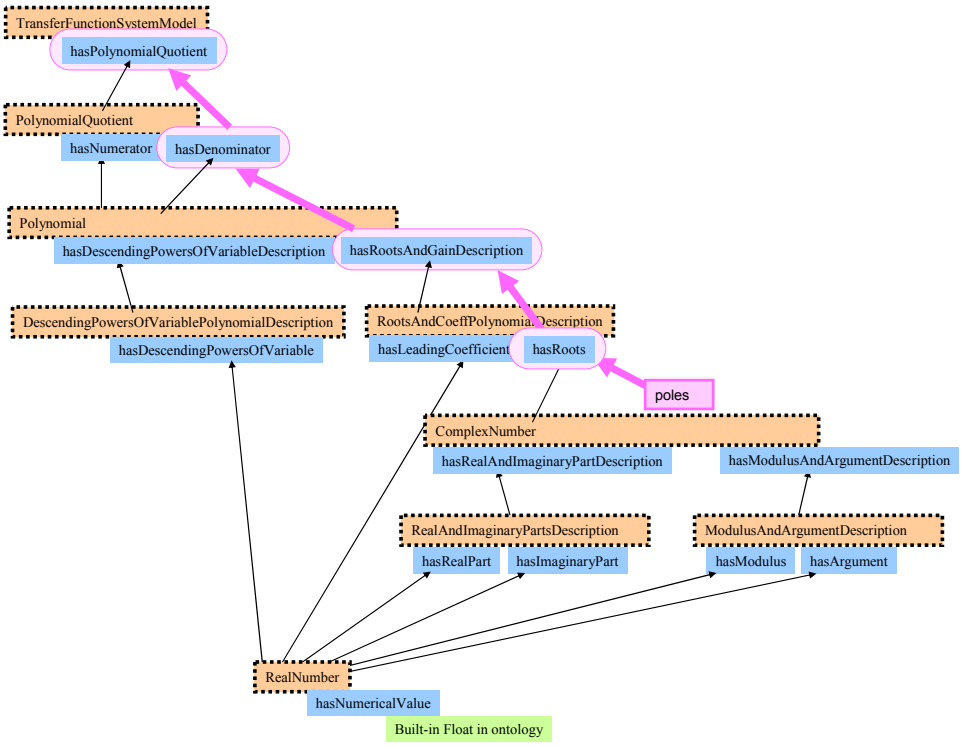


Figura 4.19. Posible definición del concepto `poles` en relación a conceptos ya representados.

Como puede observarse en la estructura de la ontología de la figura, `poles` sería el nombre de las instancias de `ComplexNumber` que se encuentran en el slot `hasRoots` de las instancias de `RootsAndCoeffPolynomialDescription` que se encuentran en el slot `hasRootsAndCoeffDescription` de las instancias de `Polynomial` que se encuentran en el slot `hasDenominator` de las instancias de `PolynomialQuotient` que se encuentran en el slot `hasPolynomialQuotient` de las instancias de `TransferFunctionSystemModel` de las que se esté diciendo algo.

Dentro del camino de slots que define a `poles` se encuentran slots a los que ya se le ha dado nombre en los ejemplos anteriores, en concreto `transfer function` y `roots`. Utilizando estos conceptos se puede acortar la definición de `poles`. Primeramente utilizando el concepto `transfer function` se pasaría de esta definición:

> *`poles` es el nombre de las instancias de `ComplexNumber` que se encuentran en el slot `hasRoots` de las instancias de `RootsAndCoeffPolynomialDescription` que se encuentran en el slot `hasRootsAndCoeffDescription` de las instancias de `Polynomial` que se encuentran en el slot `hasDenominator` de las instancias de `PolynomialQuotient` que se encuentran en el slot `hasPolynomialQuotient` de las instancias de `TransferFunctionSystemModel` de las que se esté diciendo algo*

a esta otra:

> *`poles` es el nombre de las instancias de `ComplexNumber` que se encuentran en el slot `hasRoots` de las instancias de `RootsAndCoeffPolynomialDescription` que se encuentran en el slot `hasRootsAndCoeffDescription` de las instancias de `Polynomial` que se encuentran en el slot `hasDenominator` de las `transfer function` de las instancias de `TransferFunctionSystemModel` de las que se esté diciendo algo*

Se ha resaltado el texto equivalente en ambos párrafos. Haciendo lo mismo con la expresión que define a `roots` y creando una nueva para el concepto `denominador`, podría llegarse a una definición mucho más cercana a la utilizada por un ingeniero de control:

> *`poles` es el nombre de las `raíces(roots)` del `denominador(denominador)` de la `función` de*

> *transferencia (transfer function) de las instancias de TransferFunctionSystemModel de las que se esté diciendo algo*

Los términos raíces, denominador y función de transferencia aparecen resaltados para ver su correspondencia en la versión original:

> *poles es el nombre de las instancias de ComplexNumber que se encuentran en el slot hasRoots de las instancias de RootsAndCoeffPolynomialDescription que se encuentran en el slot hasRootsAndCoeffDescription de las instancias de Polynomial que se encuentran en el slot hasDenominator de las instancias de PolynomialQuotient que se encuentran en el slot hasPolynomialQuotient de las instancias de TransferFunctionSystemModel de las que se esté diciendo algo*

Los conceptos denominador (denominator) y raíces (roots) no pertenecen al dominio del control sino al de las matemáticas, aunque el método para definirlos es conceptualmente similar. Así denominador sería:

> *denominator es el nombre de uno de los componentes de un cociente (de polinomios en este caso)*[95].

denominator es pues el nombre que recibe la instancia que se encuentra en el slot hasDenominator de una instancia cualquiera de PolynomialQuotient. Por su parte, roots sería el conjunto de instancias que aparecen en el slot hasRoots de la instancia de RootsAndCoeffDescription que está en el slot hasRootsAndCoeffPolynomialDescription de cualquier instancia de Polynomial.

El caso de denominator es muy similar conceptualmente al de transfer function ya que sólo un slot está involucrado en su definición. Una vez definidos los conceptos roots, denominator y transfer function el concepto polos (poles), como se ha visto, puede definirse utilizando estos conceptos (entidades) en vez de utilizar el camino de slots completo. En la figura 4.20 puede verse gráficamente la forma de definir el concepto poles de forma alternativa a la expresada en la figura 4.19.

[95] En la ontología no existe una conceptualización explícita de los papeles de numerador y denominador en un cociente.

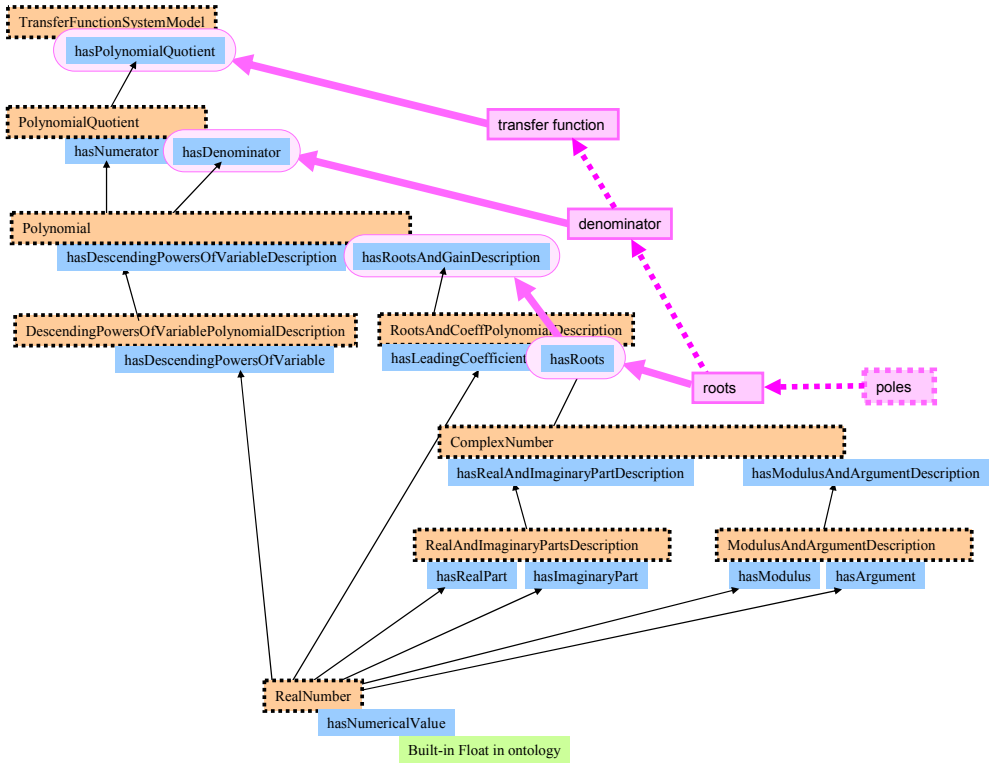


Figura 4.20. Definición del concepto `poles` en relación a los conceptos `roots`, `denominator` y `transfer function` que, a su vez, se definen en relación al esquema conceptual básico.

Como puede verse, los conceptos se pueden definir por composición de slots (en el caso de `denominator`, `roots` y `transfer function`) o de otros conceptos previamente definidos (en el caso de `poles` y `zeros` por ejemplo). Por esta razón será necesaria una nueva clase para expresar los conceptos que se describen por medio de un camino de conceptos en vez de por medio de un camino de slots. La clase será similar a la del camino de slots pero en este caso con una lista de conceptos (entidades) en vez de una lista de slots.

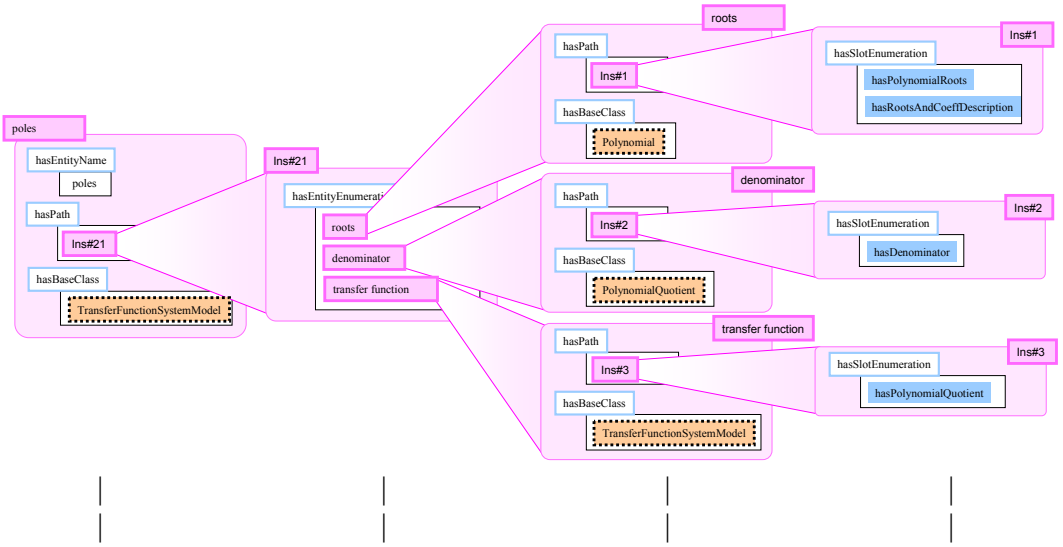


Figura 4.21. Instancia que representa al concepto `poles`.

En la figura 4.21 puede verse cómo se describe el concepto `poles` a partir de los conceptos `roots`, `denominator` y `transfer function`. Esta figura es una representación de la estructura del concepto en la ontología y puede compararse con la figura 4.20 para comprobar cómo se organizan los diferentes conceptos.

La figura 4.22 es una captura del editor Protégé en el que se ve cómo se ha definido el concepto `poles`. Es la captura real correspondiente al esquema de la figura 4.21.

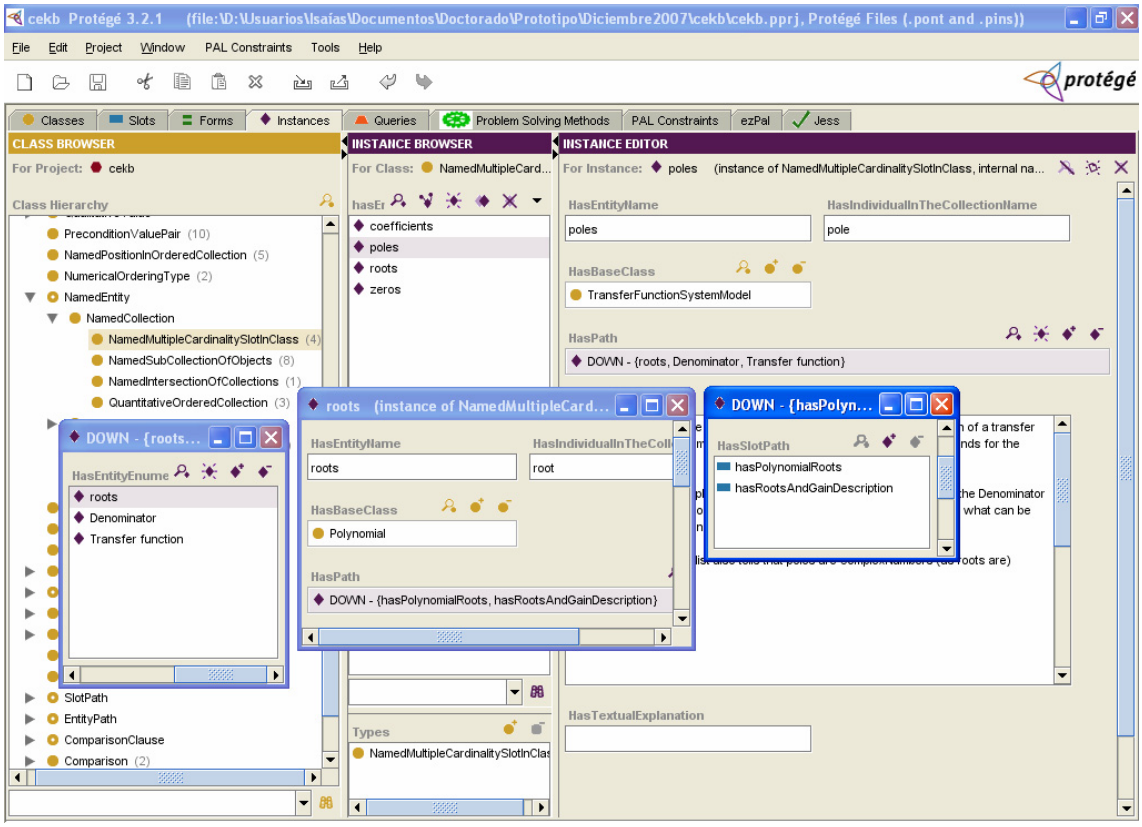


Figura 4.22. Instancia que representa al concepto `poles` en la ontología (se ha expandido el concepto `roots`)

Estructuras para representar la composición o camino de slots (y de entidades)

Se ha creado una clase `SlotPath` que es abstracta (no puede tener instancias) y dos subclases, una de ellas denominada `DownwardsSlotPath` que representa el camino de slots tal como se ha definido anteriormente, es decir, especifica una serie de slots de forma que, a partir de una determinada instancia y, accediendo a los slots en el orden en el que aparecen en la lista, se llega a obtener otra instancia. El primer slot de la lista representa un slot de la instancia a la que se aplica el camino de slots, el segundo slot de la lista representa a un slot de la instancia que está en el primer slot y así sucesivamente. Los slots que se encuentran en esta lista deben estar ordenados de forma coherente, es decir, la clase del dominio de uno debe aparecer en el rango del siguiente. Este requerimiento se puede expresar mediante un axioma en PAL. También debe asegurarse que todos los slots de la lista son no múltiples (excepto el último que sí lo puede ser).

En cualquier caso, la clase `DownwardsSlotPath` tendrá un slot, denominado `hasPath`, que será un campo múltiple de slots ordenados adecuadamente según la definición pretendida.

Existe otra subclase de `SlotPath` que se denomina `UpwardsSlotPath` y representa un camino de slots pero con un significado diferente al visto anteriormente. En este caso el recorrido no se hace "hacia abajo" sino "hacia arriba" en la estructura de clases y slots. Teniendo como referencia la figura 4.17 podría utilizarse esta forma de recorrer los slots para obtener, a partir de una instancia, otra instancia que esté relacionada con ella por medio de un camino de slots. Por ejemplo, a partir de una instancia de `ComplexNumber` se puede obtener la instancia de `PolynomialQuotient` cuyo denominador tiene entre sus raíces a esa instancia de `ComplexNumber`. El camino de slots se recorrería buscando en este caso la referencia a la instancia de partida en el slot indicado de alguna clase que contenga a ese slot. En el ejemplo, el camino de slots sería el definido por: `hasRoots` – `hasRootsAndCoeffPolynomialDescription` - `hasDenominator`. De esta forma, a partir de la instancia de `ComplexNumber` se obtendría la instancia de `RootsAndCoeffPolynomialDescription` en cuyo slot `hasRoots` esté ese `ComplexNumber`. A partir de esa instancia de `RootsAndCoeffPolynomialDescription` se obtendría la instancia de `Polynomial` en cuyo slot `hasRootsAndCoeffPolynomialDescription` esté la instancia anterior y a partir de esta última se obtendría la instancia de `PolynomialQuotient` en cuyo slot `hasDenominator` esté esta instancia de `Polynomial` encontrada en el paso previo. En este caso cada instancia sólo puede tener una sola referencia (una instancia que la contenga en el slot indicado) ya que, en caso contrario, se encontraría más de una instancia al finalizar el proceso.

El caso de los caminos de entidades se reduce al de camino de slots ya que, expandiendo los caminos de slots que cada entidad representa se obtendría un camino de slots total. La clase dedicada a representar los caminos de entidades se ha denominado `EntityPath` (con subclases `DownwardsEntityPath` y `UpwardsEntityPath`, con el mismo significado que el explicado para los slots).

Estructuras para representar los conceptos descritos mediante caminos de slots o entidades

Los conceptos (entidades) que son definidos mediante un camino de slots o de entidades tienen la siguiente conceptualización: En primer lugar, existen dos clases que recogen las entidades que son colecciones de instancias y las que son una única instancia, denominadas `NamedMultipleCardinalitySlotInClass` y `NamedSingleCardinalitySlotInClass` respectivamente. La estructura conceptual para las mismas es la siguiente:

- Slot `hasEntityName`, que llevará el nombre que se le da al concepto (a la entidad).
- Slot `hasPath`, que llevará una instancia de la clase `SlotPath` o `EntityPath` donde se definirá la lista ordenada de slots.
- Slot `hasBaseClass`, que llevará una clase que indica la clase base a la que se aplicará la definición dada por el slot anterior. La clase que aparecerá en este slot debe estar entre las que aparezcan en el dominio del último slot de la lista en la instancia que está en `hasPath`. Habitualmente ese último slot tendrá una sola clase como dominio, en cuyo caso el slot `hasBaseClass` no tendría razón de ser ya que se podría sacar la misma información del dominio, pero puede haber ocasiones en que esto no sea así

En el caso de `NamedMultipleCardinalitySlotInClass` se ha añadido un slot para hacer referencia al nombre que recibirá cada uno de los elementos en la colección. Así, si la entidad nombrada es `roots`, el elemento dentro de esa entidad múltiple se denominaría "root".

El procesamiento de estas estructuras conceptuales puede realizarse, una vez más, traduciendo la estructura a una serie de reglas o realizando un procesamiento directo en un lenguaje de programación.

Una vez vistas las entidades básicas, se describirá a continuación la conceptualización de entidades que pueden ser obtenidas a partir de las descritas anteriormente y que no se corresponden ya con instancias o colecciones de instancias que se encuentran en slots.

4.5.3.2 Entidades que representan a sub-colecciones de instancias obtenidas a partir de otras entidades

El primer tipo de entidad es aquella que representa a sub-colecciones de individuos provenientes de una colección representada previamente por otra entidad. La formación de la nueva sub-colección se hará estableciendo una serie de restricciones que los individuos de la colección inicial deberán cumplir para pertenecer a la nueva sub-colección. Estas restricciones serán expresiones matemáticas en las que estarán involucradas características que pueden ser aplicadas al tipo de elementos que aparece en la colección original. Esta forma de definir nuevas colecciones es muy similar desde el punto de vista semántico con la definición de clases en OWL, ya que las condiciones que los individuos de la nueva sub-colección deben cumplir pueden verse como una serie de condiciones necesarias y suficientes.[96]

[96] Existen también importantes diferencias respecto a la aproximación OWL, por ejemplo, en este caso, las entidades no son clases.

Un ejemplo de este tipo de entidades sería "polos reales", que puede definirse de la siguiente forma[97]:

> *Los polos reales de un sistema son aquellos polos del sistema cuya parte imaginaria es igual a cero.*

El conjunto o colección de individuos original será `poles` y de éste se extraerá un subconjunto mediante la aplicación de una condición o condiciones que se establecerán como comparaciones realizadas de acuerdo al valor de una expresión matemática (instancia de `CompoundExpression`) en la que aparecerán involucradas variables que representan a características aplicables al tipo de instancia que forma la colección. En este caso, en la expresión matemática, podrán aparecer características que se puedan aplicar a instancias de números complejos (de la clase `ComplexNumber`), ya que los polos son colecciones de números complejos en último término. En el ejemplo de polos reales la característica que se utiliza es la parte imaginaria (`imaginary part`). La condición o condiciones se establecerán mediante la inclusión en la descripción de este tipo de entidades de una precondición (instancia de la clase `Precondition`).

La clase que describe a este tipo de entidades se denomina `NamedSubcollectionOfObjects` y tiene los siguientes slots:

- `hasEntityName`
- `hasIndividualInTheCollectionName`
- `hasBaseCollection`, que representa a la colección original a partir de la cual se calculará la nueva.
- `hasPrecondition`, que contendrá la precondición que deben cumplir las instancias de la colección a la que hace referencia el slot `hasBaseCollection` para formar parte de la nueva colección.

Ejemplos de este tipo de conceptos son "polos reales" (`real poles`), "polos complejos" (`complex poles`), "polos en semiplano negativo" (`left half plane poles`), "polos en el origen" (`poles in origin`), etc.

4.5.3.3 Entidades que representan a colecciones creadas al combinar los elementos de dos o más colecciones existentes.

Este tipo de entidades se conceptualiza para evitar el tener que repetir las mismas estructuras ya creadas para definir otras entidades. Un ejemplo de este tipo de

[97] Para la descripción de la conceptualización de estas expresiones será necesario utilizar la conceptualización de las características, descritas en la sección 4.5.4 así como la de las precondiciones, descritas en la sección 4.5.5.

entidades son los "polos reales en el semiplano negativo" (`real poles in left half plane`).

La conceptualización en este caso consta de un nombre para la colección (slot `hasEntityName`) y una lista de colecciones recogida en el slot `hasCollections`. La colección resultante estará formada por la intersección de los elementos de ambas colecciones.

4.5.3.4 Entidades que representan a colecciones ordenadas de acuerdo al valor de alguna característica cuantitativa

En ciertas ocasiones es necesario hacer referencia a algún elemento de acuerdo a su posición dentro de un orden. Un ejemplo puede encontrarse en la regla para colocar el cero en un compensador de adelanto de fase[98]:

> *El cero del compensador de adelanto se colocará a la izquierda del tercer polo real de la planta.*

Más que la regla de diseño en sí lo que importa aquí es ver cómo se utiliza una colección ordenada (la de polos reales del sistema a controlar) y se nombra a un elemento dentro de esa colección. Este hecho da lugar a dos conceptualizaciones: aquella que recogerá el concepto de colección ordenada y aquella en la que un individuo se nombrará según su posición dentro de una colección ordenada. En primer lugar se presenta la primera de estas conceptualizaciones.

Para representar una colección ordenada es necesario tener como referencia una colección de partida y expresar la característica (aplicable a los elementos de esa colección) que se utilizará para establecer el orden. También será necesario especificar si el orden es respecto a cantidades crecientes o decrecientes de esa característica. La ordenación en orden creciente o decreciente se establecerá respecto al primer elemento de la colección, elemento que debería estar marcado como la cabeza de la lista, pero que no se ha conceptualizado como tal sino que se utiliza, una vez, más el almacenamiento interno de Protégé (ver apartado 4.5.1.1).

Los slot que tendrá por tanto este concepto (clase `QuantiativeOrderedCollection`) son:

- Slot `hasEntityName`, identificará a cada instancia de colección ordenada. El nombre del concepto se formará concatenando el nombre de

[98] La colocación del cero del compensador de adelanto de fase es un proceso en el que interviene mucho la heurística como experiencia del diseñador. Existen sin embargo ciertas reglas, como la que se menciona aquí, que pueden ser aplicadas, comprobando posteriormente su adecuación.

la colección sin ordenar más el tipo de ordenación más la característica utilizada para ordenar.

- Slot `hasCollection`, especifica la colección sin ordenar que se toma como dato de partida para la ordenación.
- Slot `hasQuantitativeCharacteristicForOrdering`, especifica la característica cuantitativa para realizar la ordenación.
- Slot `hasNumericalOrderingType`, especifica si el orden es ascendente o descendente. El tipo de orden se ha conceptualizado en una clase aparte, denominada `NumericalOrderingType`, que tendrá solamente dos instancias posibles que serán `ascending` y `descending`.

4.5.3.5 Entidades que representan a un elemento dentro de una colección ordenada

La motivación para la conceptualización de esta entidad proviene del ejemplo presentado en el apartado anterior. En este caso lo que se necesita es representar la posición de un elemento dentro de una colección ordenada. La conceptualización contará con un slot para hacer referencia a la colección ordenada más otro para hacer referencia a la posición en esa colección.

La posición se ha conceptualizado de tal forma que no consiste solamente en número entero especificando el índice dentro de la colección sino que se le ha dado rango de concepto con el fin de introducir la posibilidad de hacer referencia a la posición mediante una expresión textual en vez de un número. Así, se ha creado la clase `NamedPositionInOrderedCollection` que contiene un slot (`hasElementOrderName`) para referirse al texto que identifica la posición y otro (`hasIndexInCollection`) para indicar el número entero que representa el índice en la colección.

4.5.4 Las características

Como se ha mencionado en apartados anteriores, la teoría clásica de control tiene como base un modelo matemático descrito mediante la denominada función de transferencia. En este modelo está reflejada toda la información sobre el comportamiento dinámico del sistema, es decir, toda la información sobre cómo evoluciona la salida del sistema ante una entrada dada. Para describir cómo es este comportamiento dinámico se utilizan una serie de medidas que se obtienen a partir de la expresión matemática de la función de transferencia (y suponiendo una determinada forma de la señal de entrada).

Dentro de las medidas que caracterizan la respuesta de un sistema en el tiempo sólo se conceptualizarán las relacionadas con la respuesta ante una entrada escalón unitario al ser la más utilizada y la que más información proporciona sobre la dinámica del sistema.

Estas medidas sirven para caracterizar cualitativa y cuantitativamente al sistema y para guiar los procesos de análisis y diseño de compensadores. Algunas características se obtienen analíticamente a partir de las características de los polinomios que forman la función de transferencia, mientras que otras sólo pueden ser obtenidas a partir de un proceso de simulación mediante la función de transferencia correspondiente.

Además de por su forma de ser calculadas, también se pueden distinguir dos tipos de características según la naturaleza de su valor. Así, se tendrán características cuantitativas, cuyo valor será un número real, y características cualitativas, cuyo valor es un símbolo que se corresponderá con un cierto intervalo cuantitativo o con el cumplimiento de cierta precondición establecida mediante expresiones que involucran a otra u otras características cuantitativas.

La ontología deberá, por tanto, reflejar todos estos tipos de características y su forma de obtenerlas, así como la forma en la que se asociará su valor al sistema al que corresponden. También existirán características aplicables a otros conceptos que no sean sistemas, como es el caso de los polinomios, números complejos, etc.

En principio todas las características podrían ser calculadas por medio de rutinas y llamadas a funciones externas. Con esta aproximación, que es la que se haría en el caso de una aplicación de CACSD típica, se estaría perdiendo parte de la semántica. Por ejemplo, como se verá a continuación, el orden de un sistema se define en base el grado del polinomio denominador de la función de transferencia. Esta información puede obviarse y ser el resultado de una evaluación externa o puede conceptualizarse en la ontología y servir para ofrecer explicaciones y definiciones sobre los términos. Evidentemente lo ideal es almacenar tanto conocimiento explícito como sea posible en la ontología.

La mayor parte de las características que pueden aplicarse a los conceptos pertenecerán a la denominada "información no esencial" y por lo tanto su aplicación a cada concepto aparecerá separada de su estructura de slots. Existirán, sin embargo, excepciones en el caso de que alguno de los slots que describen la información esencial de un concepto sea también utilizado como una característica del mismo.

Para representar el valor de las características no se han utilizado los tipos de datos predefinidos en Protégé. Para las características cuantitativas se ha utilizado la clase `RealNumber` y para las características cualitativas se han creado clases

cuyas instancias representan los símbolos que hacen las veces de valor para las mismas.

A continuación se ofrecen las decisiones de conceptualización con respecto a las características. El apartado se ha dividido en dos partes, tratando de forma separada a las características cuantitativas de las cualitativas.

4.5.4.1 Características cuantitativas

Las características cuantitativas son aquellas cuyo valor de medida es un número real. Dentro de este tipo de características se distinguen tres subtipos que necesitarán conceptualizaciones diferentes:

- Las características que, a su vez, son conceptos estructurales.
- Las características cuya expresión (para obtener su valor) puede representarse explícitamente en la ontología.
- Las características cuyo valor se calcula mediante llamadas a funciones externas.

A continuación se trata cada uno de estos tres tipos.

Conceptos estructurales como características cuantitativas

Determinados elementos estructurales de los conceptos pueden ser utilizados como características cuantitativas de los mismos. O, si se quiere, determinadas características cuantitativas son también elementos estructurales de los conceptos. En todos los casos deberá ocurrir (dada la definición de característica cuantitativa que se ha hecho) que ese elemento estructural en cuestión sea un número real. Un ejemplo de este tipo de características es el módulo (`modulus`) de un número complejo.

La conceptualización de este tipo de características se realiza del mismo modo que se hizo en el caso de una entidad y, de hecho, en la ontología se representarán como entidades y también como características. Siendo entidades y debido a que su naturaleza es un valor numérico, pueden asimilarse a características y, por otro lado, también podría decirse que siendo características y, debido a que se corresponden con un elemento estructural de un concepto (con un slot de ese concepto), pueden considerarse entidades.

En definitiva, en la conceptualización se indicará la clase sobre cuyas instancias se aplicará la característica/entidad y un camino de slots que determina la posición del slot que hace las veces de característica/entidad. En la figura 4.23 se puede ver

la instancia correspondiente a la definición de la característica módulo (`modulus`) de un número complejo:

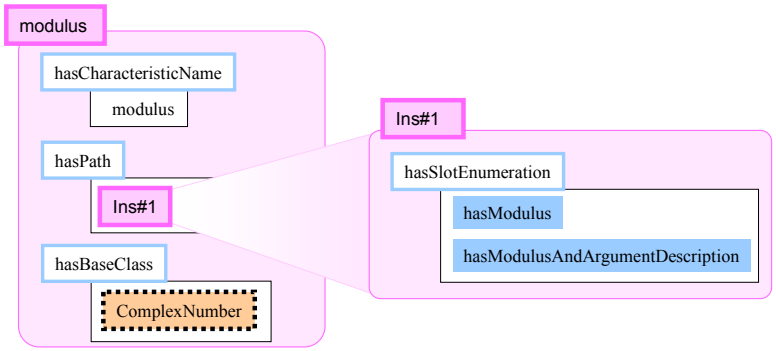


Figura 4.23. Instancia que representa a la característica `modulus` en la ontología.

La clase que representa a este tipo de características se denomina `QuantitativeCharacteristicAsNamedSlot` en la ontología y es hija tanto de `QuantitativeCharacteristic` (reflejando su naturaleza de característica) cómo de `NamedIndividual` (reflejando su naturaleza de entidad).

Cabe mencionar que el módulo (y las demás características de este tipo) podría incluirse en la categoría de característica definida, tal como se explica en el siguiente apartado, creando una expresión que reflejase el cálculo de esta característica como el resultado de la raíz cuadrada de la suma de los cuadrados de la parte real y la parte imaginaria del número complejo. Sin embargo, la decisión de ofrecer las dos descripciones de número complejo (como parte real y parte imaginaria por un lado y como módulo y argumento por el otro) de forma explícita y al mismo nivel hace que, una vez que existe un slot para representar la relación de información estructural entre un número complejo y un número real que quiere representar al módulo, sea necesario que esta característica se refiera a este slot, ya que la característica es a la vez información estructural y por lo tanto no se calcula sino que forma parte de su descripción a priori.

Algunos ejemplos de características como elemento estructural que aparecen como instancias en la ontología son las siguientes:

Parte real (`real part`) de un número complejo.
Parte imaginaria (`imaginary part`) de un número complejo.
Módulo (`modulus`) de un número complejo.
Argumento (`argument`) de un número complejo.
Coeficiente principal (`leading coefficient`) de un polinomio.

Características definidas

Las características definidas (representadas por la clase `QuantitativeCharacteristicAsExpression`) son aquellas cuya forma de ser calculadas consiste en una expresión que aparece explicitada en la ontología. De forma general consistirá en una expresión matemática (una instancia de la clase `CompoundExpression`) en la que podrán aparecer variables que representarán características cuantitativas de otras entidades asociadas a aquella de la que se quiere calcular la presente.

Como ejemplo se mostrará la conceptualización de una característica perteneciente al ámbito matemático: el grado de un polinomio. Existen diferentes formas de definir (o calcular) el grado de un polinomio:

> *El grado de un polinomio es el valor máximo del exponente de la variable (o el mayor grado de los monomios que lo componen).*
>
> *El grado de un polinomio coincide con el número de raíces que éste tiene.*
>
> *El grado de un polinomio puede calcularse restando uno al número de coeficientes del polinomio (teniendo en cuenta que se han de especificar todos los coeficientes desde el primero diferente de cero, estando ordenados de mayor a menor potencia en la variable e incluyendo el término independiente).*

Dado que la representación en la ontología del polinomio se realiza mediante los coeficientes en orden de potencia decreciente en la variable por un lado y las raíces y el término principal por otro, pueden tomarse cualquiera de las dos últimas definiciones para ser expresadas en la ontología. La variable del polinomio no se ha representado (sería la variable s representando frecuencia compleja pero, al no usarse en los procesos de análisis y diseño, no se ha incluido en la conceptualización), y tampoco se ha representado el concepto de exponente o potencia sobre la misma[99], por lo que la primera definición no puede aparecer reflejada en la ontología.

En definitiva, la expresión a modelar para la definición de grado de un polinomio puede ser "el número de raíces del polinomio" o "el número de coeficientes del polinomio menos uno". En cualquiera de los dos casos entran en juego entidades

[99] De hecho sólo se consideran polinomios de una sola variable (si se considerase más de una variable la definición de grado como se ha introducido no sería válida).

ya reflejadas en la ontología como raíces (`roots`) y coeficientes (`coefficients`), pero también entra en juego una expresión nueva: el "número de elementos". "Número de elementos" es, a su vez, una característica que se aplica a una entidad que consista en una colección de elementos, es decir, a una entidad que se traduzca a un camino de slots cuyo último slot sea múltiple. De esta forma, podría aplicarse a entidades como `roots`, `poles`, `zeros`, `real poles`, etc. Sin embargo, el número de elementos no es una característica de la instancia de esa entidad, sino de la colección que recoge los elementos que forman la estructura a la que da nombre dicha entidad. Esta característica no podrá ser descrita de ninguna forma bajo las conceptualizaciones que se han hecho en la ontología. Parte del problema radica en que el propio concepto "colección" no está definido en la ontología sino que es construido internamente como una estructura de datos a partir de la faceta multiplicidad de un slot.

La conceptualización de la característica "número de elementos" en la presente ontología consistirá solamente en una instancia que la represente. Habitualmente, a nivel del lenguaje de representación, esta característica se resuelve mediante una llamada a una función como ocurre en DAML o en KIF. De forma similar se hará en el presente caso, la característica será evaluada mediante la correspondiente instrucción del lenguaje en el que se procese la estructura ó será traducida a la correspondiente expresión en el lenguaje de reglas que se utilice.

Resumiendo la situación para el caso del cálculo del grado de un polinomio, el número de elementos será una característica que se aplicará a la colección de instancias denominada coeficientes (`coefficients`) que, a su vez, viene definida como una entidad que hace referencia a las instancias del slot `hasDescendingPowersOfVariable` de la instancia de `DescendingPowersOfVariablePolynomialDescription` que está en el slot `hasDescendingPowersOfVariablePolynomialDescription` de la instancia de `Polynomial` de la que se está hablando (de la que se quiere obtener el grado).

En la expresión compuesta (instancia de `CompoundExpression`) que describirá cómo se calcula la característica, existirá por tanto una instancia de `VariableBinding` que ligue el nombre de una variable ("num_coeff" por ejemplo) con esta característica aplicada al polinomio. La expresión a formar para calcular el grado del polinomio sería por tanto:

(num_coeff-1)

En este caso el enlace entre la variable " num_coeff " y la característica es ligeramente diferente al descrito en el apartado 4.5.1.3. En aquel caso, sin mencionarlo, parecía que todas las características que pueden aparecer en una instancia de `VariableBinding` eran características aplicadas a alguna

instancia concreta (una instancia de bloque canónico `plant` en aquel ejemplo). En esta ocasión el enlace entre variable y característica debe expresarse sin indicar la instancia concreta a la que se aplicará. Dicho de otra forma, es necesario expresar que la variable " num_coeff" está enlazada a la característica "número de elementos de los coeficientes" sin completar la expresión indicando una instancia de `Polynomial` concreta. De esta forma se puede crear una expresión genérica aplicable a cualquier polinomio[100].

A modo de resumen, y en referencia a la clase `VariableBinding`, se puede decir que una variable en una expresión matemática (`CompoundExpression`) puede estar enlazada a alguno de los siguientes conceptos:

- Característica aplicada a algo:
  - Característica de un sistema (instancia de `QuantitativeCharacteristicOfAGivenSystem` en la ontología). El sistema estará expresado como una instancia de bloque canónico o bloque canónico calculado. La instancia a la que realmente se aplica la característica será la función de transferencia a la que estará mapeada la instancia del bloque canónico correspondiente. Ejemplo: `type of plant.`
  - Característica de una entidad obtenida a partir de un determinado sistema (instancia de `QuantitativeCharacteristicOfANamedEntityOfAGivenSystem` en la ontología). No hay instancia concreta a la que esté asociada la característica, la instancia para calcular la característica se sacará del contexto donde esta característica aparezca. A este caso pertenece la situación descrita en los párrafos anteriores. Ejemplo: `degree of denominator of transfer function of plant`
- Característica sin ser aplicada, sólo el nombre de la misma
  - Nombre de una característica (instancia de `QuantitativeCharacteristic` en la ontología). Ejemplo: `real part`
  - Nombre de una característica asociada a una entidad (instancia de `QuantitativeCharacteristicOfANamedEntity` en la ontología): `degree of denominator of transfer function`

Las características aplicadas a alguna instancia concreta aparecerán habitualmente cuando se expresen hechos sobre los diferentes sistemas del diagrama de bloques, lo cual ocurrirá sobre todo en expresiones que pertenecen a la especificación de las reglas del proceso de diseño del controlador.

[100] Este aspecto de la conceptualización se expresaría en un lenguaje basado en lógica utilizando una variable cuantificada universalmente: Para todo x, si x un polinomio, el grado de x se calcula…..

Las características no aplicadas a ninguna instancia concreta aparecerán habitualmente formando parte de la definición de otras características o entidades. Por ejemplo, para expresar que los polos reales son aquellos polos que tengan la parte imaginaria igual a cero no se puede hablar de ninguna instancia específica y por lo tanto sólo aparecerá la característica `imaginary part` igualada a cero en una condición. Lo que sí se sabe es que `imaginary part` puede ser aplicada a instancias de `ComplexNumber` porque así lo especifica el slot `hasBaseClass` que describe a esta característica. Las características no aplicadas (las que aparecen solas o referidas a una entidad) podrán aplicarse a una instancia de la clase indicada en el slot `hasBaseClass` de la característica o de la última entidad que aparezca en el segundo caso (o de la clase que aparezca en el dominio, idealmente).

En la figura 4.24 se presenta la conceptualización completa para la característica grado (`degree`, de una instancia de `Polynomial`).

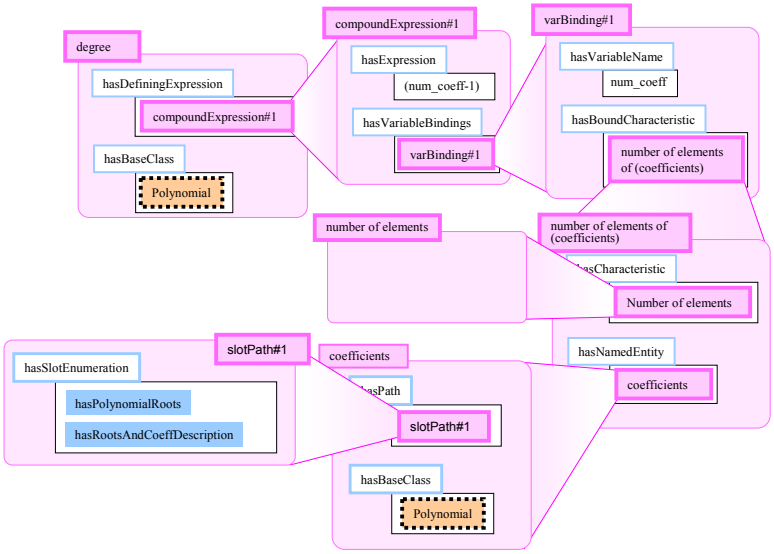


Figura 4.24. Instancia que representa a la característica `degree`.

La figura 4.25 recoge una captura de pantalla de Protégé correspondiente al esquema de la figura 4.24.

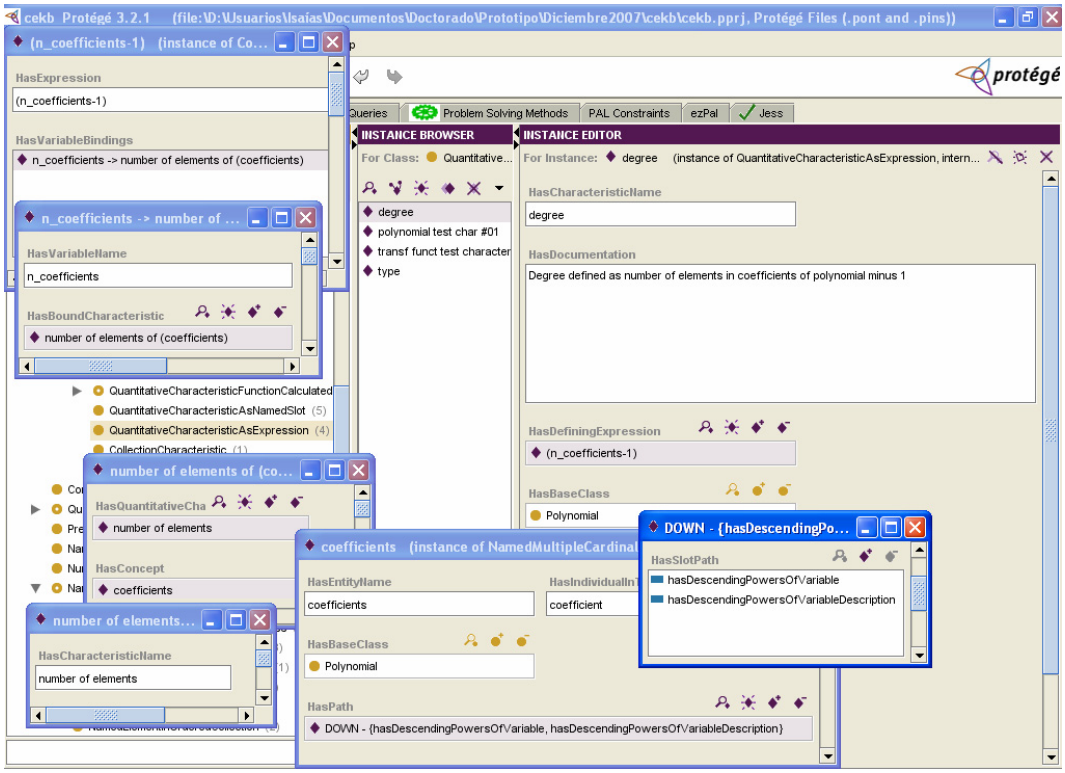


Figura 4.25. Instancia que representa a la característica `degree` en la ontología.

Ejemplos de características definidas que aparecen como instancias en la ontología son: grado de un polinomio (`degree`) o tipo (`type`) de un modelo en función de transferencia (y por extensión de un sistema).

Características calculadas mediante funciones externas

Existen ciertas características cuya expresión para ser calculadas no puede ser representada en la ontología por exceder la capacidad de representación de la conceptualización realizada o porque esas características no cuentan con una expresión analítica, sino que deben ser calculadas mediante procesos de simulación.

La forma de resolver la representación de estas características es ofrecer una conceptualización respecto a la forma en la que pueden ser calculadas, es decir, se conceptualizará la interacción entre la ontología y la aplicación externa en orden a facilitar la automatización de la llamada a la función y la conversión automática entre tipos de datos de los parámetros que se le pasen. Este tipo de características y su forma de calcularlas mediante llamadas a funciones externas es una estrategia similar a la técnica de asignación de procedimientos (procedural attachment)

introducido en el ámbito del formalismo de los marcos (Minsky, 1975) para calcular el valor de ciertos slots.

La conceptualización de este esquema conceptual, en el caso de la presente ontología, se describe a continuación.

Las características serán instancias de la clase `QuantitativeCharacteristicFunctionCalculated`, contando con los siguientes slots:

- Slot `hasCharacteristicName`. El nombre de la característica. Es una cadena de caracteres.
- Slot `hasBaseClass`. La clase a cuyas instancias se aplica esa característica.
- Slot `hasFunctionToCalculateValue`. La función externa a llamar. Es una instancia de la clase `ExternalFunction` que sirve para conceptualizar las funciones externas.
- Slot `hasParametersForFunction`. Los parámetros a pasar a la función. Es una colección de caminos de entidades o slots.
- Slot `hasInternalOntologyClass`. El tipo de datos devuelto (en términos de la ontología). Es la clase que representa al tipo de datos devuelto por la función.

La función a llamar deberá contar con una conceptualización propia, no sirve con especificar el nombre de la misma sino que deberá existir información sobre cual es la aplicación a la que pertenece esa función. La conceptualización de las funciones a llamar se hará mediante instancias con la siguiente información (correspondiente a los slots de la clase que las representa que se ha denominado `ExternalFunction`):

- Slot `hasFunctionName`. El nombre de la función
- Slot `hasApplication`. La aplicación a la que pertenece esa función.
- Slot `hasParametersDataTypes`. La lista (ordenada) de tipos de datos de los parámetros esperados. Estos tipos de datos lo son respecto a la aplicación externa y también están formalizados (en la clase `ExternalApplicationDataType`).
- Slot `hasReturnDataType`. El tipo de datos retornado, también respecto a la aplicación externa.

El orden de los parámetros debe corresponder en la ontología con el orden en el que la función los espera.

Los parámetros a pasar a la función deben extraerse a partir de la instancia sobre la que se calculará la característica. Se indicará cada parámetro por medio de un camino de entidades o slots donde el último slot del camino (de la última entidad en su caso) tendrá como dominio a la misma clase de cuyas instancias se calcula esta característica.

En la figura 4.26 se muestra la conceptualización correspondiente a la característica "sobreoscilación" (`overshoot`). Como puede verse, esta característica se calcula llamando a una función de Matlab denominada `findOvershoot` la cual espera como parámetros el numerador y denominador de la función de transferencia. Los parámetros a pasar son por tanto estos dos, el numerador y el denominador de la función de transferencia del modelo en función de transferencia.

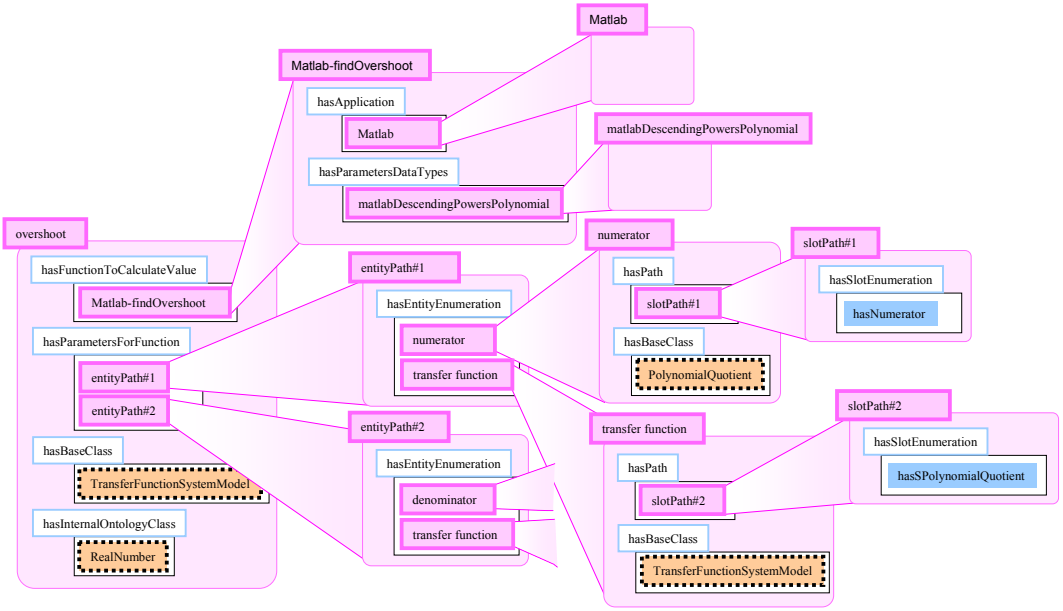


Figura 4.26. Instancia que representa a la característica `overshoot`.

La característica `overshoot` se aplica a instancias de la clase que aparece en el slot `hasBaseClass`, es decir, a instancias de `TransferFunctionSystemModel`. Los parámetros a pasar a la función se definen mediante entidades que se obtendrán a partir de esa instancia aplicando los caminos de entidades que hay en el slot `hasParametersForFunction`. Así, en este caso, el primer parámetro es "el numerador de la función de transferencia" aplicado a una instancia de `TransferFunctionSystemModel`.

Esta conceptualización permite explicar cómo se calcula la característica `overshoot` en términos de a qué función hay que llamar y qué datos se le pasan, pero no existe una representación del significado en control del término sobreoscilación más allá de una posible descripción textual.

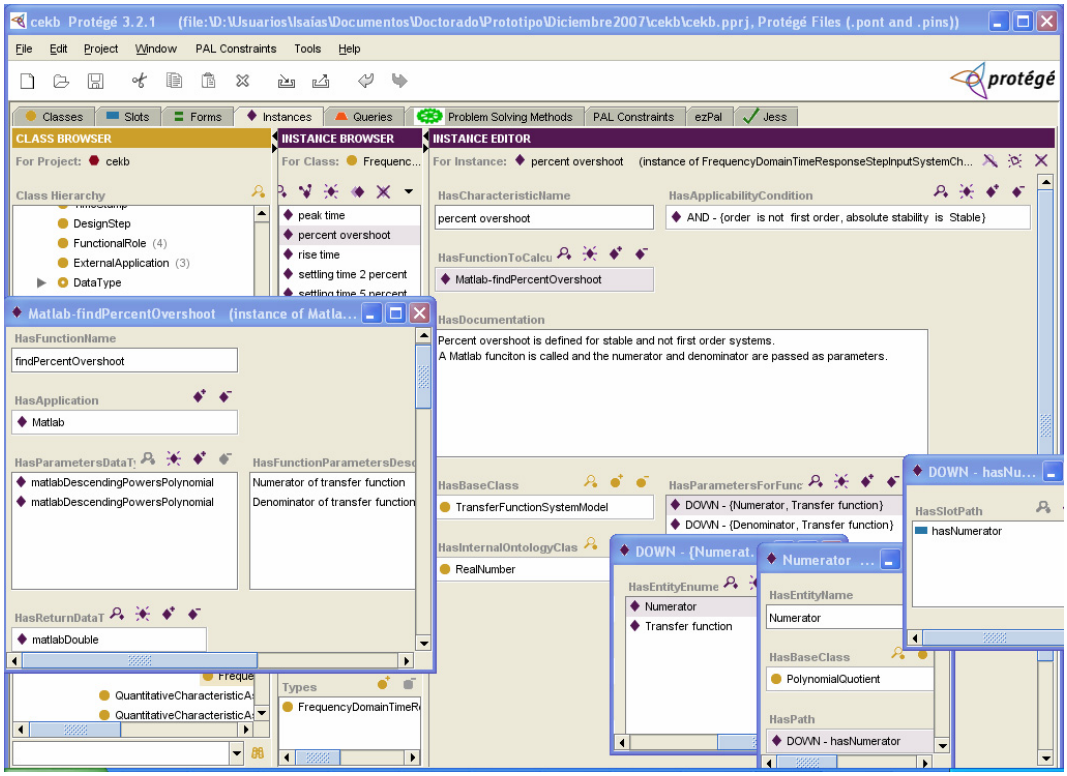


Figura 4.27. Instancia que representa a la característica `percent vershoot`.

Algunos ejemplos de características calculadas que aparecen como instancias en la ontología son las siguientes `overshoot`, `percent overshoot`, `peak time`, `rise time`, `settling time 2%`, `settling time 5%`, etc.

4.5.4.2 Características cualitativas

Las características cualitativas son aquellas cuyo rango de posibles valores es una serie de símbolos no numéricos. En esta ontología se hace una representación de las características cualitativas que suelen encontrarse en la caracterización de los sistemas. Un ejemplo puede ser la estabilidad de un sistema:

> *La definición de estabilidad del sistema puede consistir en el siguiente enunciado: si para una entrada acotada la respuesta del sistema es*

> *acotada, entonces el sistema es estable. Si la respuesta continúa creciendo cuando el tiempo tiende hacia infinito no es posible acotar la señal de salida y, por tanto, el sistema es inestable. Por último, existe un caso en el que la respuesta ante la entrada presenta un aspecto oscilatorio, sin aumentar ni decaer sino recorriendo un ciclo entre un valor máximo y uno mínimo de forma indefinida, en este caso se dice que es sistema es marginalmente estable.*

Esta definición de estabilidad se aplica a la respuesta observada de un sistema pero en control hay formas de saber si el sistema es estable o no observando las características de la función de transferencia. En concreto se tiene que:

> *Si el sistema tiene todos sus polos en el semiplano izquierdo (parte real negativa) entonces el sistema es estable, mientras que si algún polo está en el semiplano derecho (parte real positiva) entonces el sistema es inestable. La estabilidad marginal se produce cuando los polos están colocados a lo largo del eje imaginario (si alguno de los polos del eje imaginario tiene multiplicidad mayor de 1 el sistema será inestable).*

Esta relación entre la posición de los polos de un sistema y su estabilidad o inestabilidad es uno de los hechos que permite observar la cantidad de información almacenada en la función de transferencia y también es un ejemplo de la carga semántica que contiene el lenguaje de control.

En cuanto a la característica de estabilidad de un sistema, puede comprobarse cómo esta es de naturaleza cualitativa, tomando los valores simbólicos de "estable", "inestable" y "marginalmente estable" de acuerdo a una medida cuantitativa establecida como "el número de polos que hay en el semiplano complejo positivo". La condición de estabilidad marginal es más teórica que práctica ya que en la mayoría de las ocasiones la posición exacta de los polos no puede ser determinada con una precisión suficiente (pensando por ejemplo en la posición de los polos ante un ajuste de la ganancia del controlador y en las posibles variaciones ante perturbaciones externas). En este sentido, sería interesante el establecimiento de diferentes grados de estabilidad e inestabilidad dependiendo de la cercanía de los polos al eje imaginario, aunque en la presente representación no se ha abordado este tema. En la ontología se ha conceptualizado la denominada estabilidad absoluta que exige, para que el sistema sea estable, que tenga todos sus polos en el semiplano real negativo.

La mayoría de las características cualitativas en la teoría clásica de control pueden definirse estableciendo una partición en un intervalo respecto al valor de una característica cuantitativa o una expresión en la que aparezcan una o varias características cuantitativas que puedan aplicarse al sistema objeto de estudio o a

alguno de los componentes o entidades del mismo. Dentro de este tipo de características pueden encontrarse ejemplos como el orden de un sistema, el tipo de amortiguamiento, la naturaleza de fase mínima o fase no mínima, etc.

Las características cualitativas podrían asimilarse al nivel de clase ya que se hace una partición de los sistemas de acuerdo a ellas. Ejemplo: sistemas de fase no mínima frente sistemas de fase mínima.

Al igual que en el caso de las características cuantitativas, las características cualitativas se dirán del modelo en función de transferencia con el que se está trabajando, aunque en las expresiones habituales de control se apliquen directamente al sistema.

Por último, es necesario mencionar también que existen características cualitativas que se dicen de elementos que no son modelos de sistema, por ejemplo de los conceptos matemáticos, como es el caso del signo de un número real (positivo, negativo).

La conceptualización que se ha hecho, en vez de asociar intervalos de valor numérico a un valor simbólico, se ha basado en relacionar el cumplimiento de una precondición con la aplicabilidad de ese valor simbólico cualitativo (comparaciones y precondiciones se tratan en la sección 4.5.5). El concepto que representa a una característica cualitativa tendrá por tanto una serie de pares precondición - valor simbólico.

Los valores simbólicos se han conceptualizado aparte como instancias (de alguna de las subclases de `QualitativeValue`) para poder utilizarlos en las expresiones (si se introducen como cadenas de caracteres en la propia estructura de la definición de característica luego no podrían utilizarse en expresiones, por ejemplo dentro de las reglas que describen el proceso de diseño de compensadores). En el caso de las características cuantitativas, todas ellas se traducen a instancias de la clase `RealNumber`. En este caso, cada característica cualitativa tendrá asociada una clase cuyas instancias serán los posibles valores que la característica podrá tomar.

En las figuras 4.28 y 4.29 puede verse el ejemplo de la característica cualitativa estabilidad absoluta de un sistema (`absolute stability`).

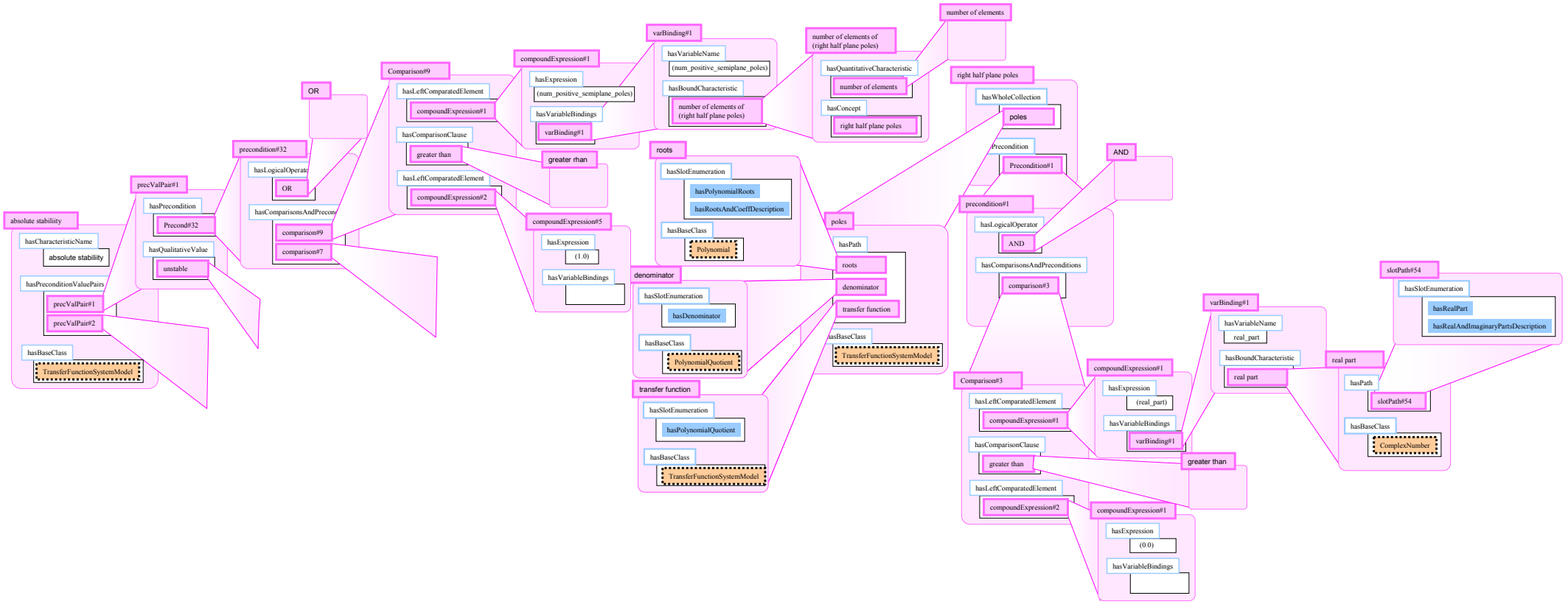


Figura 4.28. Esquema de la instancia que representa a la característica `absolute stability` en la ontología.

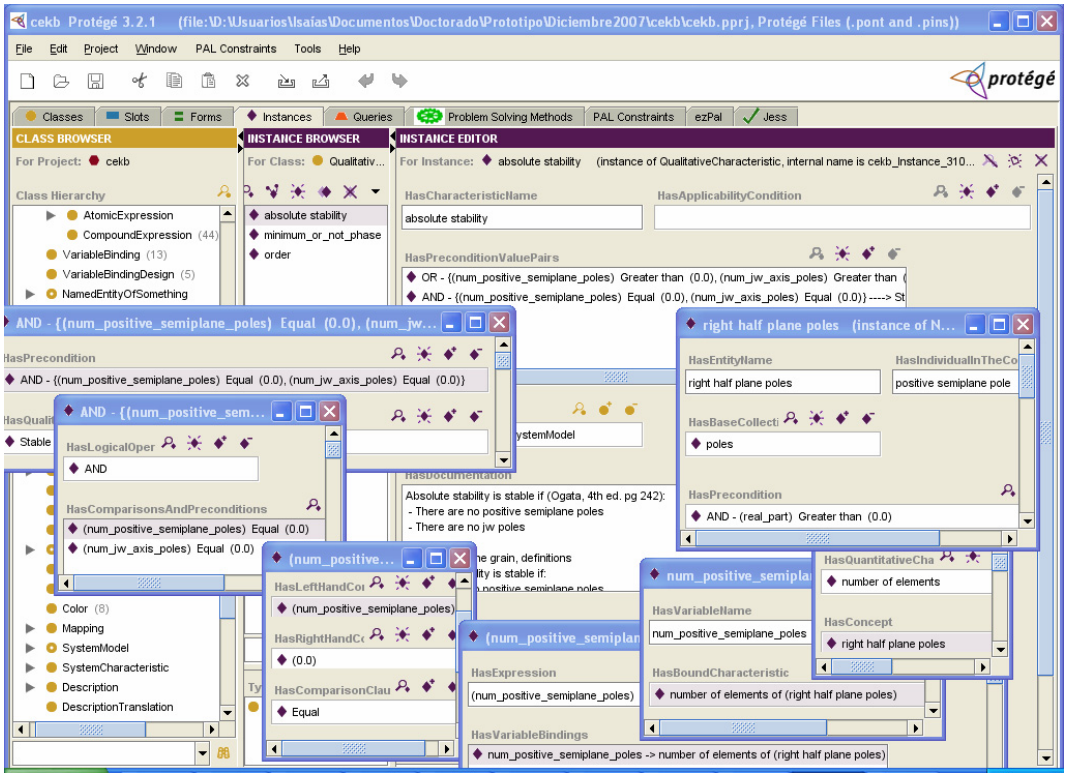


Figura 4.29. Instancia que representa a la característica `absolute stability` en la ontología.

Definir la correspondencia entre valor cuantitativo y cualitativo mediante precondiciones tiene la ventaja de permitir condiciones sobre más de una característica y poder construir expresiones más complejas, aunque la conceptualización realizada tiene también algunos problemas, como el hecho de que no existe forma de asegurar que la partición del intervalo de valores es disjunta y completa, es decir, que no hay solapamientos y que se cubren todos los posibles valores del intervalo.

4.5.4.3 Condiciones de aplicabilidad de las características

Para terminar el apartado dedicado a la descripción de la conceptualización de las características es necesario mencionar que, tanto en el caso de la conceptualización para características cuantitativas como cualitativas, existe un slot dedicado a representar la condición de aplicabilidad de la característica.

La condición de aplicabilidad de una característica consiste en una serie de requisitos que debe cumplir el individuo sobre el que se calcula la característica para que dicha característica pueda serle aplicada. Por ejemplo, la característica tiempo de pico (`peak time`) sólo tendrá sentido y aplicabilidad en sistemas estables y que no sean de primer orden.

El slot que recoge la condición de aplicabilidad (`hasApplicabilityCondition`) tiene como valor una precondición, estructura cuya conceptualización se describe a continuación.

4.5.5 Comparaciones y precondiciones

Se ha dicho ya que los procesos de análisis y diseño en la teoría clásica de control se basan en el estudio y toma de decisiones en base a los valores de ciertos parámetros que sirven para caracterizar a los sistemas involucrados en estos procesos.

La estructura básica para elaborar y describir los procesos de razonamiento sobre las características es la comparación. La comparación es un proceso en el que una característica o, más genéricamente, una expresión matemática en la que pueden aparecer diferentes características se compara con otra expresión o con un valor numérico (en el caso de una característica cuantitativa) o simbólico (en el caso de una característica cualitativa). El resultado de la comparación será un valor de verdad (verdadero o falso).

Una determinada agrupación de comparaciones puede formar un bloque de decisión más complejo, que se denominará precondición. Las precondiciones pueden agruparse también, formando precondiciones de más alto nivel. Tanto la agrupación de comparaciones como la de precondiciones se hará mediante los operadores lógicos AND y OR, ya que la naturaleza del resultado de la evaluación, tanto de comparaciones como de precondiciones, es un valor de verdad.

4.5.5.1 Estructura para la conceptualización de una comparación

La estructura básica de una comparación consiste en dos expresiones a comparar y una cláusula de comparación. Los términos a comparar se denominarán "término a la izquierda" y "término a la derecha", comparándose el resultado de la evaluación del primero contra el del segundo.

La expresión a comparar será una instancia de la clase `CompoundExpression` definida en el apartado 4.5.1.3 (y en el apartado 4.5.4.1 para ciertos aspectos de la

clase `VariableBinding`). La cláusula de comparación será también conceptualizada con el fin de traducir la expresión en la ontología al lenguaje en el que se implementará la evaluación de la misma.

La cláusula de comparación es una instancia de una subclase de la clase `ComparisonClause`. Las subclases que existen son dos: una dedicada a las cláusulas de comparación de medidas cuantitativas (`QuantitativeComparisonClause`) y otra para las medidas cualitativas (`QualitativeComparisonClause`).
Las cláusulas de comparación cuantitativas son:

- Igual (`equal`).
- Mayor que (`greater than`).
- Menor que (`less than`).
- Diferente (`different`).

En el caso de las cláusulas de comparación cualitativas sólo existirá la posibilidad de la igualdad o no igualdad, que serán representadas mediante las expresiones "es" (`is`) y "no es" (`is not`) respectivamente. Como se puede ver en la conceptualización de las características cualitativas, éstas se limitan a representar con un símbolo una precondición, esto es, una serie de comparaciones, y por lo tanto una comparación cualitativa expresa, en realidad, si se cumplen (o no) esa serie de comparaciones cuantitativas.

En la figura 4.30 se representa la estructura conceptual correspondiente a una comparación en la que se expresa la comparación entre el resultado de sumar 1 al tipo de la planta (`type of plant`) y el número cero (0.0). La cláusula de comparación es "mayor que".

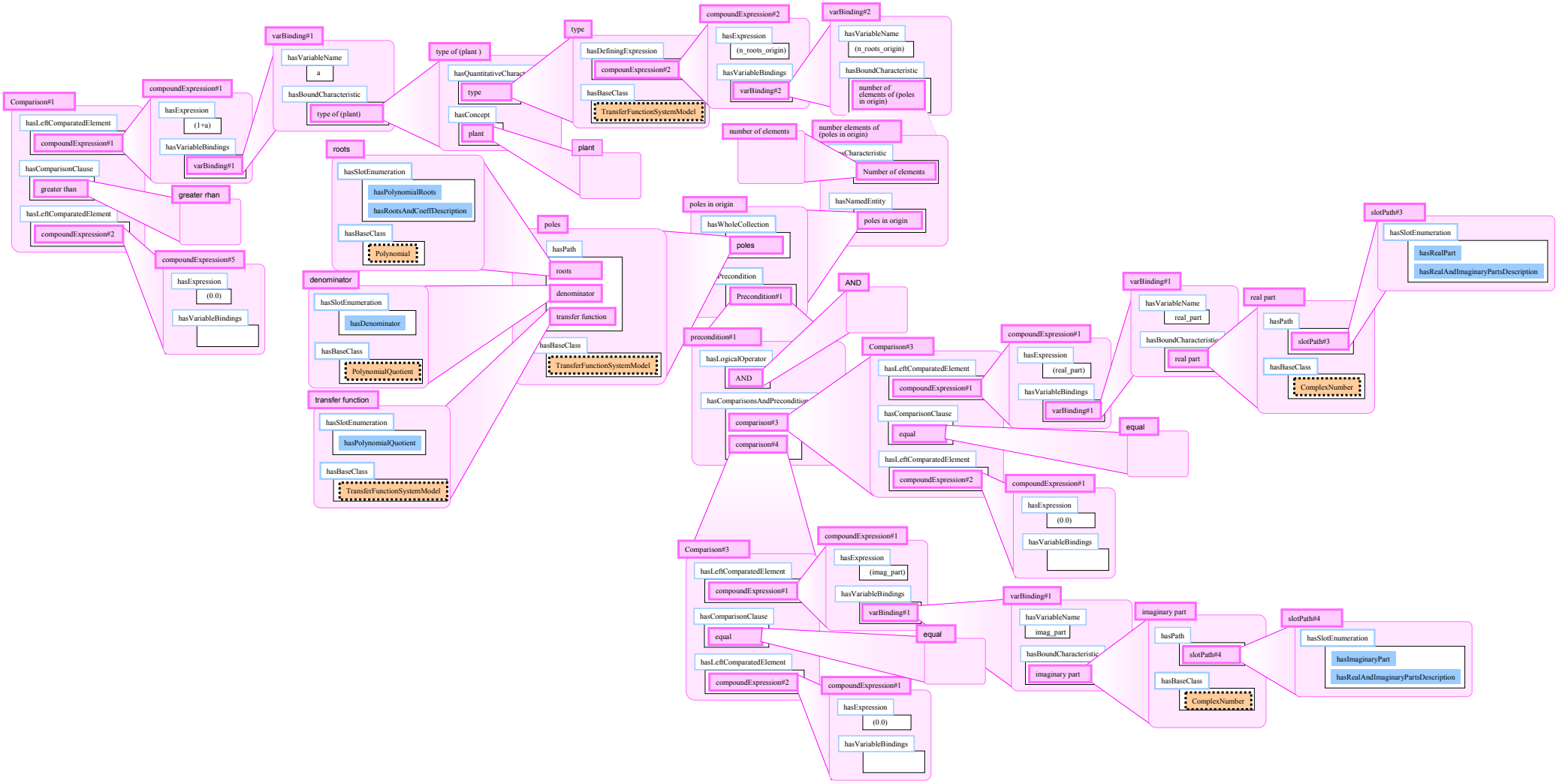


Figura 4.30. Instancia que representa a una comparación en la ontología

4.5.5.2 Estructura para la conceptualización de precondiciones

Las precondiciones son, como se ha dicho, bloques de comparaciones unidas por los operadores lógicos AND y OR. Agrupan a una serie de comparaciones (también se pueden agrupar otras precondiciones, estableciendo así diferentes niveles en las mismas) creando una expresión compleja que, en conjunto, será evaluada a un valor de verdad.

Las precondiciones se utilizarán en diferentes lugares, formando parte de diferentes estructuras conceptuales en la ontología. A continuación se describen algunos usos:

- Para establecer rangos o condiciones sobre las que se definen las características cualitativas (descrito en la sección 4.5.4.2).
- Para definir entidades que son subcolecciones a partir de entidades que son colecciones, estableciendo algún tipo de restricción establecida sobre las características aplicables a los elementos que forman la colección inicial y que todos los elementos de la sub-colección deben cumplir. Es el caso del concepto "polos reales" (`real poles`) por ejemplo (descrito en la sección 4.5.3.2).
- Formando parte de la parte izquierda (de las condiciones) de las reglas de producción que describirán el aspecto dinámico de las aplicaciones que se implementen en base a esta ontología (no abordado en esta tesis).
- Para expresar condiciones de aplicabilidad de características (descrito en la sección 4.5.4.3)

A modo de ejemplo, en la figura 4.31 se presenta una precondición sencilla formada por dos comparaciones. Esta comparación puede utilizarse, por ejemplo, para obtener la subcolección de “polos imaginarios puros”, ya que establece la comparación entre la parte real (de un número complejo) y el número 0.0 utilizando el operador de igualdad.

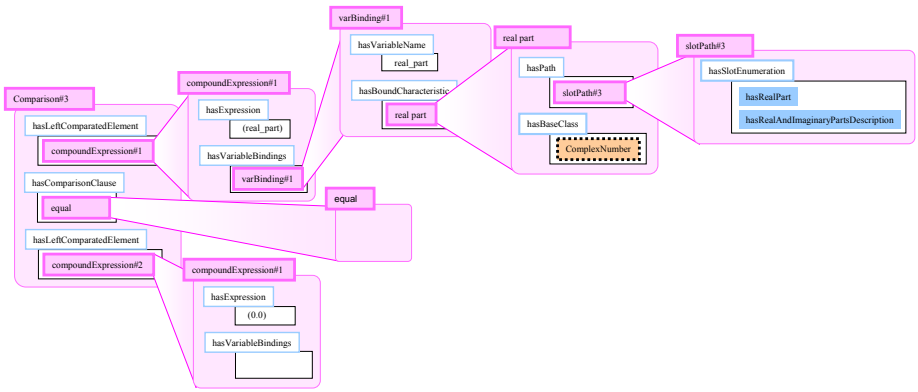


Figura 4.31. Instancia que representa a una precondición sencilla

La figura 4.32 recoge una precondición más compleja, en la que entran en juego dos comparaciones que, para que la precondición se cumpla, deben evaluarse al valor de verdad TRUE.

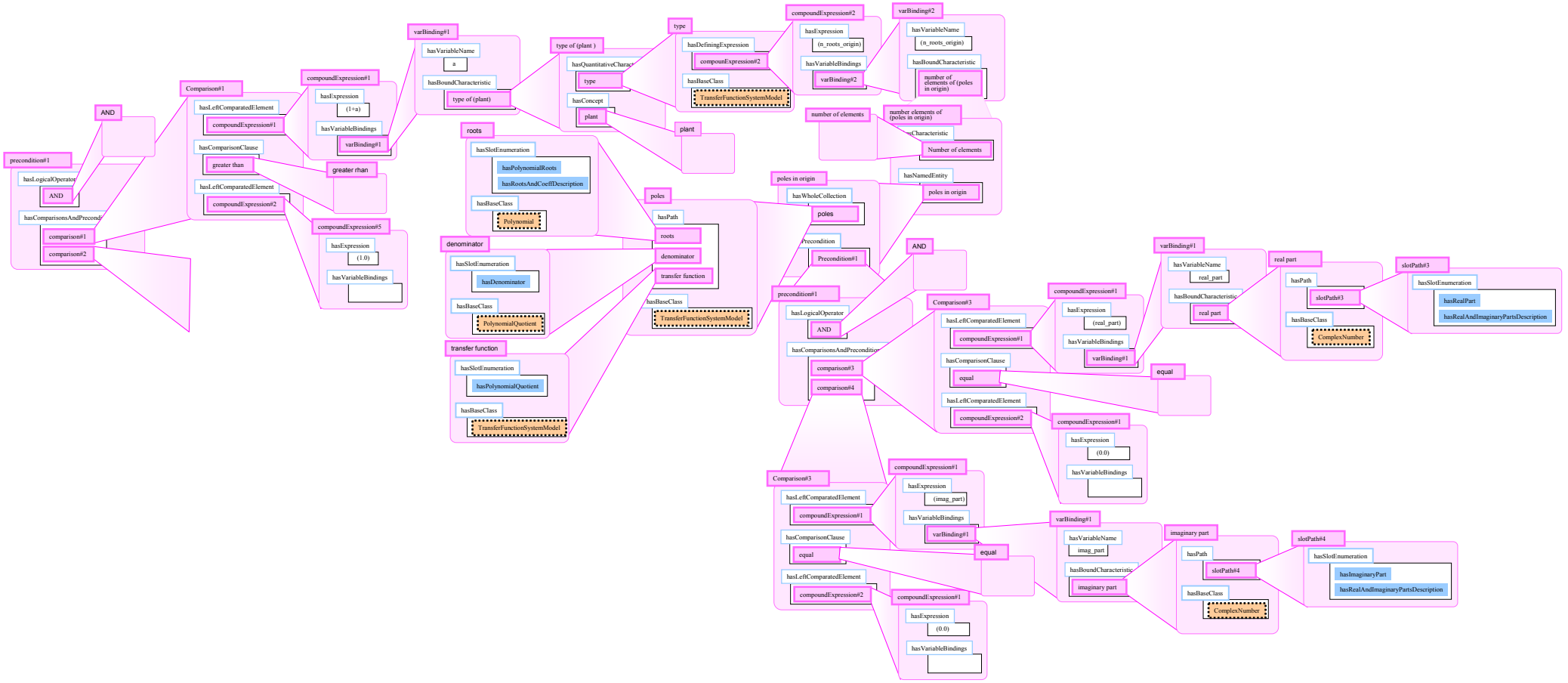


Figura 4.32. Instancia que representa una precondición formada por dos comparaciones

4.5.6 Predicación sobre los conceptos: atributos no esenciales

Como se mencionó en el apartados 4.5.2.2, la conceptualización de la presente ontología distingue entre propiedades (atributos) esenciales y no esenciales. Las esenciales, identificadas con el mecanismo de descripción estructural, se representan mediante slots en las clases, mientras que las no esenciales se representan en una estructura asertiva aparte. En el presente apartado se describe esta estructura asertiva.

Las propiedades no esenciales de los conceptos se han modelado como triplas objeto-atributo-valor, que también pueden verse como relaciones binarias entre un concepto y su propiedad (atributo). Mediante este mecanismo se representarán todas las características de los conceptos vistas en la sección 4.5.4 y todas las entidades tratadas en la sección 4.5.3.

Las triplas serán instancias de una clase que refleje su estructura objeto-atributo-valor. Un aspecto importante respecto a esta estructura de triplas es que ninguna instancia existirá en la misma hasta que existan datos concretos de un problema a resolver. Una vez que existan estos datos, las triplas se generarán de forma automática a partir de la definición que de los atributos (características y entidades) existe en la ontología.

La estructura de la tripla es sencilla, estando representada por la clase `Triple`, formada por tres slots[101]: uno de ellos para albergar la instancia que representa al concepto del que se dice algo (el objeto), el último para albergar qué es lo que se dice de esa instancia (el atributo) y uno más para recoger el valor que para esa instancia tiene lo que de ella se dice. Los tres slots contendrán instancias de la ontología. Ejemplos de triplas serían:

(`plant`, `poles`, (`-2+j3`, `-2-j3`, `-1`)) (1)

(`denominator of plant`, `degree`, `3`) (2)

(`plant`, `order`, `higher order`) (3)

(`plant`, `type`, `0`) (4)

La tripla (1) se generará a partir de la definición de la entidad "polos" (`poles`), por lo tanto el valor en esta tripla será una colección de números complejos

[101] Se distinguen dos tipos de triplas: las que representan a entidades y las que representan a características. Están representadas por las clases `NamedEntityValueTriple` y `CharacteristicValueTriple` respectivamente, ambas subclases de la clase `Triple`.

(instancias de `ComplexNumber`) que representan las raíces del denominador del cociente de polinomios de la función de transferencia del sistema a controlar (instancia `plant`).

La tripla (2) se obtiene a partir de la definición de la característica cuantitativa "grado" (`degree`), por lo tanto el valor de la tripla, en este caso una instancia de `RealNumber`, se obtiene al calcular el número de coeficientes del polinomio denominador del cociente de polinomios de la función de transferencia del sistema a controlar y restarle uno.

La tripla (3) se obtiene a partir de la definición de la característica cualitativa "orden" (`order`) y la (4) a partir de la definición de "tipo" (`type`).

La generación de las triplas se hará mediante un procesamiento de la definición de las entidades y características. Este procesamiento puede ser llevado a cabo mediante un lenguaje de programación procedural o bien transformando estas definiciones en reglas de producción[102], haciendo que las mismas se disparen ante la presencia de datos. En el apartado 5.2 del siguiente capítulo se presenta la implementación según el primer método mencionado.

4.5.7 Visión general de la base de conocimiento

En este punto es interesante presentar una visión general de la base de conocimiento (que consiste en la ontología más el conocimiento asociado a un problema concreto), así como de la forma en la que ésta se construye.

En la figura 4.33 puede verse, a modo de resumen general, la estructura de la ontología y base de conocimiento completa (incluyendo el conocimiento dinámico de diseño). Esta estructura se genera en diferentes etapas y de diferentes formas. Partiendo desde cero se tendrían los siguientes pasos:

1. La fase 1 es en la que se define la estructura de clases y slots, pero sin instancias. Las conceptualizaciones más importantes son las relacionadas con las estructuras matemáticas y las que describen la forma de definir entidades y de calcular características: `RealNumber`, `ComplexNumber`, `TransferFunctionSystemModel`, `QuantitativeCharacteristic`, ……

2. En la fase 2 se introducen los términos utilizados en el lenguaje de ingeniería de control (entidades, características, …) que serán empleados

[102] A este tipo de reglas se las suele llamar de derivación (Wagner, 2002)

en la fase 3: `poles`, `real poles`, `denominator`, ..., `rise time`, `absolute stability`, `degree`,.....

3. La fase 3 es una fase de adquisición del conocimiento. El conocimiento introducido en esta fase es conocimiento dinámico, que formará la estructura dinámica de la ontología.

4. La fase 4 la realiza el usuario de la aplicación construida en base a la ontología y la base de conocimiento. Los datos introducidos en esta fase pertenecen a un problema concreto. En la topología conceptualizada (realimentación unitaria y negativa, con el controlador en serie con la planta) esta fase se limita a introducir los polinomios que forman la función de transferencia del sistema a controlar así como las especificaciones de diseño requeridas.

5. La fase 5 se lleva a cabo automáticamente al procesar las definiciones sobre entidades y características definidas en la ontología dentro de la fase 2. Las instancias creadas en esta fase tampoco formarán parte de la ontología porque son específicas del problema que se está tratando.

6. La fase 6 se ejecuta automáticamente al aplicar el conocimiento dinámico introducido en la fase 3 sobre la base de conocimiento. Durante esta fase se crearán también instancias que no pertenecerán a la ontología, ya que son también específicos al problema concreto.

7. El proceso de generación de conceptos seguiría, ante el planteamiento de un problema nuevo, volviendo a la fase 4.

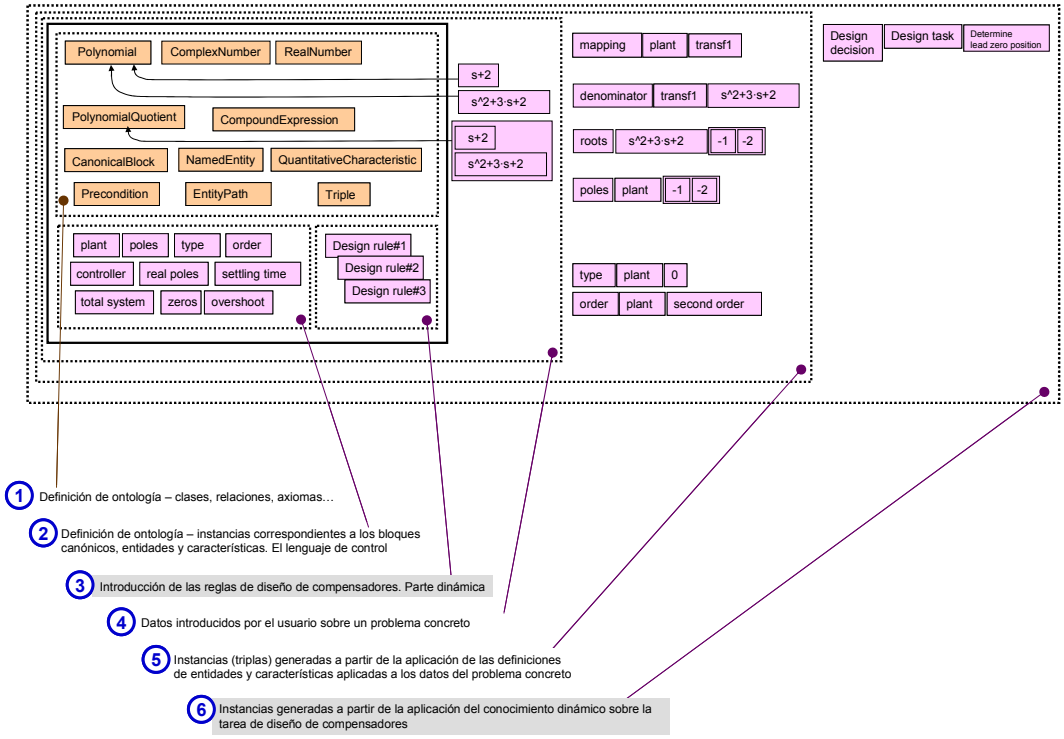


Figura 4.33. Esquema general de la ontología y base de conocimiento

La creación de conceptos puede producirse en fase de diseño o en fase de ejecución de la misma (en este segundo caso los conceptos creados son instancias siempre). Los conceptos que se mantienen invariables serán los obtenidos en las fases 1, 2 y 3, mientras que los demás (correspondientes a las fases 4, 5 y 6) son conceptos relacionados con un problema concreto.

Las fases 1 y 2 (y la estructura de triplas de 5) son las que se han descrito hasta el momento. La fase 3 queda fuera del alcance de la presente tesis y se está desarrollando en otro trabajo de tesis. Las fase 4 y la implementación de la 5 se explicarán en el capítulo 5. La fase 6 corresponde a la ejecución de la estructura dinámica especificada en 3 y por lo tanto también queda fuera del alcance del presente trabajo.

4.5.8 Conceptos gráficos

La importancia de las representaciones gráficas en la ingeniería de control es bien conocida. Inicialmente, estos métodos se crearon para comprender mejor los procesos de análisis y diseño y para servir de aproximaciones alternativas a los complejos cálculos numéricos que debían realizarse. Estas gráficas permiten,

además, obtener una visión general y profunda del comportamiento dinámico de los sistemas

La mejora de prestaciones de los ordenadores ha hecho que los métodos gráficos tengan hoy en día una importancia renovada (Bissell, 2004), (Johanson et. al., 1998), ya no tanto como métodos de simplificar los cálculos numéricos, que ahora pueden ser realizados por los ordenadores de forma muy rápida, pero sí en su vertiente de ofrecer una visión práctica y directa del comportamiento del sistema.

La representación en la ontología de los conceptos involucrados en este tipo de gráficas puede contribuir a aumentar, todavía más, las posibilidades que las mismas tienen como elementos centrales en la interacción con el usuario.

Los conceptos de la ontología que tienen una representación gráfica en el plano complejo son los siguientes:

- Las especificaciones de diseño, representadas como zonas en el plano complejo. Pueden estar delimitadas por líneas verticales, horizontales, círculos o líneas que pasan por el origen.
- El conjunto de especificaciones de diseño, que son representadas como un área de diseño que es la resultante de combinar las zonas especificadas anteriormente.
- Los polos y los ceros de los sistemas, representados mediante aspas y círculos, respectivamente.
- El lugar de las raíces, representado mediante curvas tipo spline cúbico.

4.5.8.1 Representación del lugar de las raíces.

En primer lugar, se ha creado una representación para especificar un "camino" de puntos en el plano. La clase correspondiente se denomina `ParametricPointPath`. La clase tiene un slot de cardinalidad múltiple en el que se almacena la lista de puntos. Estos puntos, a su vez, están representados por una clase con slots para la posición del punto (que viene dada por un número complejo), el valor del parámetro variable en ese punto (un número real) y la pendiente de la curva que los puntos definen en ese punto concreto (representada como un número complejo). Cada rama del lugar de las raíces será uno de estos caminos de puntos.

Se ha definido también una estructura dedicada a agrupar varios caminos de puntos (clase `SetOfParametricPointPaths`). El lugar de las raíces será una instancia de esta clase. De hecho, cualquier segmento del lugar de las raíces (es decir, el conjunto de ramas del mismo entre dos valores del parámetro) será una instancia de esta clase.

El lugar de las raíces se ha tratado como una entidad especial, como una entidad calculada externamente y no definida (caso del resto de entidades). La clase correspondiente a este concepto es `NamedCalculatedEntity` (es una subclase de `NamedEntity`) y tiene una estructura similar a la descrita para las características calculadas, es decir, se explicita la función externa a llamar, los parámetros a pasarle y el tipo de datos de retorno.

4.5.9 Otras conceptualizaciones

Además de las estructuras de conocimiento presentadas hasta el momento existen otras que complementan a éstas y que están dedicadas a diferentes cometidos. Las más relevantes se presentan a continuación.

4.5.9.1 Expresiones de entidades y características

En el método del lugar de las raíces toda entidad que exista se obtendrá, en último término, de un modelo en función de transferencia. Sin embargo, puede existir gran complejidad en la composición de estas entidades cuando aparecen en las expresiones del lenguaje de control. También, cualquier característica se dirá de alguna entidad que, en último término, esté referida a un modelo en función de transferencia. A modo de ejemplo, se puede hacer referencia a la siguiente expresión:

"el grado del denominador de la función de transferencia de la planta"

En esta expresión entran en juego una característica y una expresión de entidades en las que una entidad se dice de otra entidad que, en último término, se dice de un modelo en función de transferencia. Así, se tiene:

- La característica "grado" que se dice de la entidad "denominador de la función de transferencia de la planta".
- La entidad "denominador" que se dice de la entidad "función de transferencia de la planta".
- La entidad "función de transferencia" que se dice de "la planta" (y que en último término, haciendo uso del mapeo, hará referencia a un modelo en función de transferencia (instancia de `TransferFunctionSysteModel`).

Para reflejar estas expresiones se utilizan diferentes conceptos en la ontología. En concreto, para crear expresiones de entidades que se dicen sobre otras entidades se tienen la clase `NamedEntityOfSomething` que tiene, como subclases:

- Clase `NamedEntityOfANamedEntity`, que se utiliza para aplicar una entidad a otras previamente existentes. En esta clase hay un slot para la entidad que se dice de otra y para la entidad de la que se dice algo que, a su vez, puede ser una instancia de esta misma clase, permitiendo así anidamientos de expresiones de este tipo.
- Clase `NamedEntityOfAGivenSystem`, que se utiliza para aplicar una entidad o una expresión compleja de entidades definida como instancia de la clase mencionada anteriormente, a una instancia de bloque canónico (clase `CanonicalBlockDiagramComponent` ó clase `CalculatedCanonicalBlock`).

Existe otra clase, denominada `QuantitativeCharacteristicOfSomething`, que recoge las clases dedicadas a expresar características de sistemas, de entidades, y de entidades aplicadas a sistemas. Estas clases son, respectivamente:

- Clase `QuantitativeCharacteristicOfAGivenSystem`. Por ejemplo: `type of (plant)`
- Clase `QuantitativeCharacteristicOfANamedEntity`. Por ejemplo: `degree of (Denominator)`
- Clase `QuantitativeCharacteristicOfANamedEntityOfAGivenSystem` Por ejemplo, la expresión total mencionada al comienzo del apartado: `degree of (Denominator of (transfer function of (plant)))`

4.5.9.2 Requerimientos de diseño

La conceptualización de los requerimientos de diseño en la ontología se utilizará en la parte dinámica de la misma. El uso actual de la misma consiste en su representación gráfica como área de diseño (en el caso de las especificaciones de régimen transitorio).

Para representar el conjunto de requerimientos de diseño se ha utilizado la clase `DesignObjective`, que tiene un slot múltiple en el que se representan los diferentes requerimientos de diseño. Estos requerimientos serán instancias de la clase `FrequencyDomainDesignRequirement` que, a su vez, estará formada por tres slots: uno para recoger la característica sobre la que se establece el requerimiento, otro para establecer la cláusula de comparación y otro más para recoger el valor deseado para ese requerimiento de diseño.

4.5.9.3 Documentación textual

Todos los conceptos en la ontología tienen un par de slots dedicados a ofrecer información textual sobre los mismos. Uno de ellos estará dedicado a ofrecer una explicación sobre la conceptualización realizada (orientada hacia un ingeniero del conocimiento) mientras que el segundo contiene información sobre la definición o descripción de los conceptos de la ontología (orientado hacia el usuario de la aplicación CACE).

La documentación textual es de gran importancia en una ontología (Uschold, 2006), tanto para obtener información que no esté explícitamente almacenada en la misma como para servir de ayuda a desarrolladores de ontologías ó aplicaciones que utilicen, o reutilicen, la ontología.

4.5.10 Métricas de la ontología

A modo orientativo, las métricas para los diferentes tipos de elementos que aparecen en la ontología construida son las siguientes:

- Clases: 235.
- Slots: 297.
- Instancias: 468.

Estas cifras se corresponden a la ontología sin datos concretos de problemas, es decir, a los puntos 1 y 2 de la figura 4.33.

4.6 Discusión y conclusiones parciales

El proceso de conceptualización y creación de la ontología descrita en este capítulo permite realizar reflexiones sobre diferentes aspectos del mismo y desde diversos puntos de vista: sobre el proceso de construcción de la ontología, sobre el conocimiento de control, en cuanto a las estructuras de conocimiento encontradas y su formalización, etc.

4.6.1 Sobre el proceso de construcción de ontologías en general

Existen numerosas formas de abordar y realizar una conceptualización, lo que da lugar a diferentes ontologías posibles para un mismo dominio. Este hecho hace que, pese a utilizar formalismos o lenguajes de representación equivalentes, e incluso utilizando los mismos, el modelo conceptual final pueda ser muy

diferente. En muchas ocasiones, las estructuras creadas y el conocimiento capturado en ellas depende en parte del uso práctico que está previsto dar a las mismas. A este problema, ya planteado en los años 1980s, se le dio el nombre del "problema de la interacción" y fue descrito del siguiente modo:

> *"La representación del conocimiento con el fin de solucionar algún problema está fuertemente afectada por la naturaleza del problema y la estrategia de inferencia que será aplicada al problema"* (Bylander y Chandrasekaran, 1988)

Estos estudios llevaron a postular la separación del conocimiento sobre resolución de problemas (conocimiento dinámico) del conocimiento estático dentro de una ontología, de forma que este conocimiento estático no presentase el problema de la interacción y por tanto las ontologías construidas para el mismo fuesen realmente reutilizables en diferentes aplicaciones y tareas (Musen, 1998).

Otra solución para este problema parte de distinguir entre diferentes tipos de ontologías, postulando que las más genéricas serán más reutilizables, al estar afectadas en menor medida por el problema de la interacción, mientras que las ontologías de dominio, con un conocimiento a representar mucho más restringido, serán más susceptibles de reflejar un punto de vista particular.

En cualquier caso, actualmente se admite la posibilidad de que existan diferentes puntos de vista al conceptualizar un dominio, pero también se postula que el desarrollo de las ontologías debe perseguir el fin de que las mismas sean reutilizables en la mayor parte de dominios (Guarino, 1997).

En la experiencia obtenida en el desarrollo de la presente ontología el problema de la interacción se ha hecho patente en las dos vertientes o causas identificadas en (Bylander y Chandrasekaran, 1988). Por un lado, la tarea a implementar sobre el conocimiento representado tiende a determinar qué tipos de conocimiento deben ser representados, de forma que no es deseable modelar todo el conocimiento posible, sino solamente el que sea necesario. En segundo lugar, se tiende a codificar el conocimiento de tal manera que la estrategia de razonamiento a implementar sobre el mismo pueda ser eficiente.

El primero de los factores que producen el problema de la interacción se manifiesta de forma muy clara en la ontología desarrollada dentro de la conceptualización de los elementos matemáticos realizada. Ésta ha sido parcial e incompleta en algunos aspectos, sirviendo bien para el fin perseguido, pero no siendo útil para reutilizarse directamente en cualquier otro dominio que pueda plantearse. El suponer polinomios de una sola variable, no conceptualizar el exponente, suponer sólo coeficientes reales, etc., son algunos ejemplos. Otro ejemplo puede ser la forma en la que se ha definido el grado de un polinomio

(como el número de coeficientes menos uno), que sirve para poder calcularlo directamente pero no si lo que se persigue es transmitir la noción y la definición de ese concepto. El haber desarrollado la conceptualización de esta forma se justifica por el hecho de que, de no haber sido así, el esfuerzo realizado para conceptualizar completamente este dominio de las matemáticas habría sido, al menos, comparable al del resto de la ontología.

El desarrollo de un mayor número de ontologías, una de más alto nivel, y otras centradas en dominios concretos y acotados, junto con la reutilización de las mismas, puede solventar este factor que hace manifestarse al problema de la interacción. Para que esto ocurra deberán desarrollarse una gran cantidad de ontologías, lo que conlleva contar con equipos multidisciplinares formados por expertos en los dominios correspondientes e ingenieros del conocimiento, así como utilizar un formalismo que sea fácilmente integrable y automatizable para conseguir la reutilización. En el campo de las matemáticas, por ejemplo, pueden y deben realizarse ontologías que no estén orientadas a ningún tipo de aplicación. Se dice que las matemáticas son el lenguaje de la ciencia y como tal sería representado el dominio: como una serie de términos del lenguaje que, posteriormente, fuesen utilizados en otros dominios como es el caso del control automático.

Por último, mencionar que, si bien OWL ofrece un lenguaje basado en teoría lógica de modelos y por tanto la integración sería posible, su expresividad hoy en día no es suficiente para representar la mayoría de las estructuras de conocimiento relevantes necesarias. El uso de mecanismos de representación del conocimiento como las reglas de producción dificultan la reutilización de las ontologías.

El segundo aspecto que influye en el problema de la interacción aparece también en la ontología desarrollada. Los términos de control (entidades, características) se modelaron como instancias, entre otras razones, para poder ser utilizados en posteriores procesos de adquisición del conocimiento. En concreto, estas instancias son utilizadas para definir otros conceptos y son también la base para la conceptualización del conocimiento dinámico. En ambos casos se estaba pensando en la implementación de la estructura de tareas involucrada en los procesos de diseño de controladores. Pese a que existen estudios dedicados a evitar esta vertiente del problema de la interacción, las investigaciones acerca de la representación y procesamiento de las estructuras dinámicas de las ontologías están mucho menos desarrolladas que las correspondientes a las estructuras estáticas[103] (Chandrasekaran et. al., 1999).

[103] Esto es debido, en gran parte, a la aplicación del conocimiento factual de dominio en las investigaciones acerca de la comprensión del lenguaje y también a las aplicaciones directas en la Web Semántica.

Por otro lado, aun cuando la ontología pueda considerarse completa, pueden existir diferentes conceptualizaciones que sean igual de correctas desde el punto de vista de ser un modelo válido del conocimiento pero que tengan estructuras de conocimiento muy diferentes[104]. Una posibilidad para solventar este problema es ofrecer mecanismos de mapeo de conceptos que permitan comparar y traducir conceptos y estructuras equivalentes entre ontologías (Noy, 2004), otra sería que surjan conceptualizaciones que puedan ser consideradas estándar y adoptadas por un amplio grupo de usuarios.

4.6.2 Sobre aspectos de conceptualización del dominio de la ingeniería de control

4.6.2.1 La naturaleza del conocimiento de control

En primer lugar, sobre la propia naturaleza del conocimiento en ingeniería de control, puede decirse que la particularidad más relevante radica en el uso de un lenguaje propio con el que se transmite dicho conocimiento. Lo que se expresa en este lenguaje son los términos de la denominada "aproximación sistémica", hecho que caracteriza a la ingeniería de control frente a otras ingenierías y que también condiciona la forma de conceptualizar los conceptos frente a otras conceptualizaciones existentes acerca de dominios de la ingeniería que tienen en cuenta los dispositivos, su conexión, su comportamiento y/ó su función.

En el caso de la ingeniería de control no se hace referencia directa al sistema físico, sino a un modelo matemático del mismo. Y es más, lo que se usa en el lenguaje de control son nombres que se le dan en este dominio a determinados componentes estructurales de esos modelos matemáticos y, por tanto, a nivel de la ontología, nombres que se le dan a elementos que ya existirán en la conceptualización del dominio matemático. Esta estrategia sirve, en último término, para simplificar las expresiones con las que se transmite el conocimiento en esta disciplina tecnológica. Sin embargo, esto no es una cuestión sólo de ahorro de expresiones, sino que la disciplina necesita este lenguaje para poder existir, desarrollarse y ser comunicada.

Uno de los aspectos clave de la conceptualización ha sido cómo reflejar este hecho en la ontología. La solución final, como se ha visto, fue la utilización de instancias para representar a las denominadas “entidades” de control. Estas entidades son los nombres que reciben los elementos estructurales de los modelos matemáticos de los sistemas (por ejemplo, los polos son las raíces del denominador de modelo en función de transferencia).

[104] De hecho la discrepancia entre conceptualizaciones puede aparecer en varios aspectos aún cuando se utilicen lenguajes muy formales como OWL. Ver (Uschold, 2003).

Aparte de las entidades, el otro elemento conceptual importante en el dominio son las medidas utilizadas para caracterizar a los sistemas y guiar los procesos de análisis y diseño. Estas características tienen una naturaleza similar a la de las entidades ya que "se dicen" también, en último término, de los modelos en función de transferencia de los sistemas. Ambos conceptos, entidades y características son tratados como propiedades (o atributos) de los conceptos matemáticos más básicos y, en último término, de los conceptos que representan a los modelos en función de transferencia de los sistemas. Todo el lenguaje de control se construye en base al uso, en las expresiones de este lenguaje, de estos atributos.

4.6.2.2 Clases e instancias, ontología y bases de conocimiento

La conceptualización realizada, reflejada en la figura 4.33, permite realizar una serie de comentarios acerca de la distinción entre ontología y base de conocimiento. Esta distinción, ya presentada como un aspecto problemático en el apartado 4.4.1, se estudia a continuación desde el punto de vista de la conceptualización realizada.

De acuerdo a (Guarino y Giaretta, 1995) una "base de conocimiento arbitraria" (o teoría lógica arbitraria) está formada por conocimiento que describe, al menos en alguna de sus estructuras, una situación o configuración concreta de los conceptos involucrados. Una ontología, por el contrario, sería una representación genérica de las posibles situaciones o configuraciones concretas que se puedan dar. De estas ideas surge la distinción clásica entre ontología, considerando que está formada por la estructura de clases y base de conocimiento, que englobaría a la ontología y a las instancias que describen situaciones o configuraciones concretas. En este sentido, todavía podría existir un tercer término para describir solamente a las estructuras que describen situaciones concretas: en (Guarino, 1998) se denomina "núcleo" de la base de conocimiento a estas estructuras de conocimiento.

Desde la distinción descrita en el párrafo anterior se puede tratar de encontrar la línea divisoria entre la ontología y el "núcleo" de la base de conocimiento. Más que realizar esta distinción mediante la separación clases/instancias se debe hacer la pregunta de si los correspondientes conceptos describen conocimiento genérico o sobre una situación concreta.

Bajo las consideraciones realizadas, los puntos 1, 2 y 3 de la figura 4.33 podrían considerarse parte de la ontología, de hecho, podrían asimilarse a lo que en (Guarino, 1998) se denomina "ontología de aplicación". De esta forma, el "núcleo" de la base de conocimiento estaría compuesto por el conocimiento relacionado con un problema concreto, es decir, por las instancias generadas en

los puntos 4, 5 y 6. En adelante, cuando se hable de "base de conocimiento" se hará referencia al conjunto formado por la ontología y el "núcleo" de la base de conocimiento, es decir, a las estructuras de los puntos 1 a 6 de la figura 4.33.

4.6.2.3 Uso de instancias en la ontología

La discusión sobre el uso de instancias en la ontología tiene diversas vertientes. Por un lado, ya se ha mencionado que las mismas facilitan el uso de los conceptos en posteriores procesos de adquisición del conocimiento (este aspecto, el de las ontologías como herramientas para facilitar la adquisición del conocimiento (Shadbolt et. al., 2004), se ha presentado como muy relevante en la experiencia realizada). Otro uso que se ha dado a las instancias es el hacer las veces de representantes de las características concretas que se pueden aplicar a los conceptos de la ontología. En este caso, el nivel de clase se deja para clasificar los tipos de características que existen y la estructura conceptual de estas clases permite definir la forma en la que la característica se puede obtener, ya sea mediante una expresión o mediante la llamada a funciones externas.

Las instancias se utilizan también para representar el valor de las características. En el caso de las características cuantitativas, éstas toman como valores instancias de la clase `RealNumber`, mientras que las características cualitativas toman como valores instancias que representan los símbolos en los que se divide el espacio cualitativo de cada característica. El uso de instancias como valores de características parece intuitivamente adecuado aunque diversos estudios, sobre todo provenientes del área de las lógicas descriptivas y más en concreto de OWL, preconizan el uso de clases para crear las particiones de valores que puede presentar una característica, al menos en el ámbito cualitativo (Rector ed., 2005). El utilizar esta estrategia permite aprovechar la expresividad de la lógica y poder realizar un mayor número de razonamientos automáticos, pero tiene problemas si se intenta construir una definición de características cualitativas asociadas a expresiones con características cuantitativas, tal como se hace en la presente ontología.

Detrás de los aspectos comentados en los últimos párrafos, incluyendo la difícil distinción entre ontología y base de conocimiento, está uno de los problemas clásicos en la creación de modelos conceptuales en ontologías: el de decidir si un concepto se modelará como instancia o como clase (Noy y McGuinness, 2001). En (Valente et. al., 1999), por citar una ontología del ámbito de la ingeniería, se encuentra descrito un problema similar.

4.6.2.4 Descripción y definición de conceptos

Otro aspecto relevante de la conceptualización realizada es el uso conjunto de la descripción y la definición de conceptos. La parte descriptiva aparece, en la ontología, en el esqueleto conceptual básico, es decir, la estructura de clases y slots. La parte de definición entra en juego en la representación de los conceptos de control: las entidades y las características.

La descripción es el mecanismo de los formalismos basados en marcos para construir conceptos, mientras que la definición es utilizada por los lenguajes basados en lógicas descriptivas. Este hecho hizo que, al elegir un formalismo basado en marcos, fuese necesario crear procesadores de la semántica para las estructuras que reflejan definiciones en la ontología. En cualquier caso, el haber elegido OWL para aprovechar las capacidades de definición del formalismo tampoco habría sido la solución porque OWL es capaz de representar definiciones de clases, mientras que en esta ontología lo que en realidad se está definiendo son atributos (representados como instancias). De esta forma, en los dos casos habría que construir o utilizar procesadores de semántica complementarios.

La estructura, creada en la presente ontología, para crear las definiciones de conceptos se asemeja, y puede traducirse, a un esquema similar al utilizado en la parte de "condición" de una regla de producción. Esta estructura es también fácilmente automatizable por medio de código en un lenguaje de programación (aproximación que se ha realizado en este trabajo y que se presenta en el siguiente capítulo).

4.6.3 Sobre la finalidad y uso de la ontología

Por último, otro aspecto importante que merece la pena mencionar y que se ha comentado de forma parcial anteriormente es la finalidad perseguida en el desarrollo de la ontología. La ontología realizada lo ha sido con el objetivo primordial de ser un modelo del conocimiento del dominio tratado que, aunque limitado e incompleto, pueda servir como base para mostrar la posibilidad de realización de aplicaciones informáticas con unas características que no se encuentran en las existentes en la actualidad. Un segundo objetivo es la localización de las estructuras de conocimiento más importantes en el dominio tratado. La reutilización de la ontología, por tanto, no ha sido el objetivo principal en este desarrollo. Por esta razón, tampoco se ha prestado una atención especial al problema de la interacción.

Capítulo

5

Experimentos y resultados

> Unless you can describe a specific problem and a specific ontology (or set of ontologies), and a specific methodology for using the ontology to address the problem, it is unlikely that you will get anyone to listen to you. This means that, for the most part, you must produce a successful application to demonstrate that the route you are suggesting will be worth pursuing. […] Unless you can provide specific answers to questions such as "Exactly what problem of ours is this going to help solve?" and "Exactly how is it going to help solve it?", you won't have much of a case. Generality and vagueness are the enemies of success.
>
> *- Foro de discusión de Protégé (post de Gary H. Merrill)*

En el presente capítulo se presentan las aplicaciones creadas para procesar las estructuras semánticas de la ontología y para comprobar las posibilidades del software CACE creado en base a modelos conceptuales representados en ontologías. En el primer caso se exponen los diferentes algoritmos utilizados para generar nuevo conocimiento a partir de las estructuras de la ontología y ante la aparición de datos sobre un problema concreto. Con la segunda aplicación se muestra la posibilidad de mejorar la interacción entre el usuario y el conocimiento del dominio a través del uso de una interfaz gráfica en la que todos sus elementos están asociados a conceptos de la ontología. Mediante el mantenimiento de esta relación entre la interfaz gráfica y la ontología el usuario puede obtener descripciones, definiciones y explicaciones sobre los diferentes conceptos que aparecen. De esta manera se comprueba la validez de la aproximación planteada para mejorar el software dedicado al control.

5.1 Introducción

En este capítulo se presenta la estructura y creación de un par de aplicaciones informáticas realizadas con dos objetivos concretos. La primera de ellas es un procesador de las estructuras semánticas creadas que van más allá de las capacidades del formalismo de marcos. Por ejemplo, la estructura de definición de

entidades (subclases de `NamedEntity`) necesita un procesamiento para poder ser aplicada. Esta aplicación es necesaria para el correcto funcionamiento de la ontología, independientemente del uso que se haga de la misma en una aplicación.

La segunda aplicación está encaminada a comprobar cómo el software puede hacer uso de las estructuras de conocimiento almacenadas en una ontología, así como la forma en la que la interfaz de usuario se relaciona con los conceptos almacenados y las posibilidades que esta aproximación ofrece. Además, esta aplicación servirá para comprobar el conocimiento almacenado y por tanto la validez de las diferentes definiciones y descripciones de los conceptos. La aplicación creada no tiene ningún uso específico pretendido, aunque puede utilizarse en tareas de educación ó de ayuda al diseño (sobre todo cuando se añada el conocimiento dinámico a la ontología). En todo caso, el objetivo es demostrar cómo puede ser la interacción entre el usuario y una aplicación CACE creada en base a una ontología. La figura 5.11 recoge un esquema general de las dos aplicaciones y los principales módulos que las componen.

A continuación se expone el funcionamiento de la aplicación procesadora de las estructuras ontología y de la aplicación CACE desarrollada, así como la forma en la que se han programado. En la sección 5.2 se describe la aplicación encargada de procesar la semántica de las estructuras construidas. Se presentan las estructuras y algoritmos más relevantes, introduciendo de forma breve la forma en la que se procesan. La sección 5.3 está dedicada a describir los aspectos relacionados con la aplicación que utiliza la ontología para presentar al usuario el conocimiento almacenado en ella. Finalmente se ofrecen una serie de discusiones y conclusiones sobre todos estos aspectos.

5.2 Procesador de estructuras semánticas

La aplicación dedicada a procesar las estructuras semánticas de la ontología es imprescindible para la creación y mantenimiento del modelo de conocimiento y la base de conocimiento. No tiene componente de interfaz gráfica y se encargará de implementar las estructuras semánticas, recogidas en el capítulo 4, que no tienen soporte en el formalismo de marcos.

La herramienta Protégé proporciona diversas funcionalidades para implementar la semántica que el formalismo de marcos contiene[105]. Entre ellas se pueden citar:

- Establecimiento y mantenimiento de la jerarquía de clases que representa la relación is-a, implementando el mecanismo de herencia de slots.

[105] Lo que se implementa es la especificación explicitada en el protocolo OKBC.

- Establecimiento y mantenimiento de la jerarquía de slots.
- Asociación de slots a clases. Herencia de slots, posibilidad de modificación (override) de las características de los slots a nivel de cada clase en la jerarquía.
- Posibilidad de establecer restricciones basadas en características (facetas, facets) de los slots como la cardinalidad, el dominio, el rango, valores por defecto, etc.
- Posibilidad de establecer restricciones complejas sobre los slot, restricciones que no se pueden representar con las facetas mencionadas en el apartado anterior.

Para establecer las restricciones complejas a las que se hace referencia en el último apartado se utiliza el lenguaje PAL (Protégé Axiom Language o Lenguaje de Axiomas de Protégé). Aunque se le denomina “lenguaje de axiomas”, con PAL no se establecen axiomas, es decir, no se dice las cosas que son verdad en el modelo sin necesidad de demostración, sino que se establece las cosas que deben ser verdad sobre el conocimiento existente. Dicho de otra forma, no se puede hacer programación lógica con PAL ni definir conceptos y, de hecho, el nombre del lenguaje debería contener la palabra "restricciones" en vez de "axiomas", es decir, debería denominarse "lenguaje de restricciones de Protégé" (Tu, 2001).

El resto de estructuras semánticas que Protégé no contempla y que ha sido necesario reflejar en la ontología deben implementarse en forma de código en la aplicación “procesadora de semántica” realizada a tal efecto. La mayor parte de las estructuras a procesar reflejan definiciones de conceptos.

En concreto, las estructuras semánticas utilizadas en la ontología que se deben procesar de esta forma son las siguientes:

- Mapeo entre bloques canónicos y función de transferencia (descrito en el apartado 4.5.2).
- Procesamiento y evaluación de expresiones matemáticas (descrito en el apartado 4.5.1).
- Procesamiento de las comparaciones (descrito en el apartado 4.5.5).
- Procesamiento de las precondiciones (descrito en el apartado 4.5.5).
- Procesamiento de las entidades y creación de triplas representándolas (descrito en el apartado 4.5.3).
- Procesamiento de características y creación de triplas representándolas (descrito en el apartado 4.5.6).
- Encadenamiento de propiedades. Hacia abajo y hacia arriba. - Con slots y con entidades (descrito en el apartado 4.5.3.1).
- Traducción entre representaciones: entre diagrama de bloques y diagrama de flujo de señal, entre diferentes formas de representar un polinomio y un número complejo (descrito en el apartado 4.5.1).

- Comunicación con aplicaciones externas para calcular valores de atributos (entidades y características), es decir, un mecanismo similar al de asignación de procedimientos (procedural attachment). (descrito en el apartado 4.5.4.1).

A continuación se verá, de forma breve, cómo se ha implementado el procesamiento de estas estructuras. Todo el procesamiento se realiza a través de la comunicación con la estructura de la ontología por medio de la API Java de Protégé.

5.2.1 Procesamiento del mapeo

El mapeo quiere decir que todo atributo que se aplique a un concepto puede ser aplicado a aquél al que éste concepto está mapeado. Por ejemplo, en realidad, la característica "tiempo de establecimiento" se aplica y se calcula a partir del modelo en función de transferencia de un sistema pero, como el modelo en función de transferencia es un sucedáneo del sistema real, existirá un mapeo entre ese modelo concreto y ese sistema real (representado en la ontología por una instancia de bloque canónico). Por lo tanto, se generarán hechos que reflejen las características referidas a los sistemas reales (a la instancia canónica). La noción de mapeo se introduce como una necesidad para poder utilizar expresiones del lenguaje de control (refiriendo características a los sistemas) mientras se mantiene la coherencia en el conocimiento (las características en realidad se refieren a un modelo determinado del sistema).

Otra forma de expresar el mapeo sería: Si una instancia iA tiene una propiedad pA y la instancia iA está relacionada con la instancia iB mediante la relación de mapeo entonces la instancia iB tiene también la propiedad pA. Esta expresión permite ver que la estructura de mapeo podría modelarse fácilmente por medio de reglas: habría que hacer un patrón que emparejase con cualquier objeto de los existentes en las triplas de la base de conocimiento. Si el elemento correspondiente al objeto en la tripla tiene un mapeo con otro elemento entonces habrá que crear una tripla nueva con este nuevo elemento y las mismas instancias para atributo y valor. Este tipo de reglas estaría dentro de la categoría de "reglas de derivación" en la división que suele hacerse en el campo de las reglas de negocio (The Bussiness Rules Group, 2000).

El procesamiento del mapeo consiste, pues, en duplicar las triplas que se refieran a instancias de la clase `TransferFunctionSystemModel` añadiendo triplas que se refieran a los sistemas a los que están mapeados (sólo existe este tipo de mapeo en la ontología).

5.2.2 Evaluación de las expresiones matemáticas

Las expresiones matemáticas (instancias de la clase `CompoundExpression`) se evalúan siempre a un número real. La notación utilizada en la cadena de caracteres que representa a la expresión matemática se ha hecho equivalente con la notación de la aplicación Maple, por lo que es éste programa de cálculo numérico el encargado de realizar la evaluación de la expresión (además, desde la versión 9.5 de Maple, existe una API basada en Java para acceder al motor matemático de esta aplicación).

Antes de hacer la llamada a Maple para evaluar la expresión es imprescindible resolver todas las asociaciones de variables que haya en esa expresión (sólo se evalúan expresiones totalmente numéricas, nunca simbólicas). Las asociaciones (instancias de la clase `VariableBinding`) se resuelven en una rutina separada. En último término el valor numérico será una característica cuantitativa de alguna entidad de las que existen en la ontología.

Una vez obtenido el valor numérico, resultado de la expresión, éste es capturado en Java y posteriormente introducido en la ontología como instancia de la clase `RealNumber`. Existen una serie de clases que centralizan la creación de instancias en la base de conocimiento así como la traducción entre diferentes lenguajes y tipos de datos (de Java a la ontología, por ejemplo).

5.2.3 Evaluación de las comparaciones

Las comparaciones cuantitativas se resuelven traduciendo (al lenguaje de programación con el que se implementa el procesador de la comparación) la expresión de la misma. En el caso implementado los números reales se transforman a tipos de datos float de Java y el comparador también a la correspondiente sintaxis Java.

Las comparaciones cualitativas consisten en verificar si una determinada característica cualitativa de una determinada entidad es o no es coincidente con uno de los posibles valores simbólicos que puede tomar esa característica. En este caso se comparan instancias, que son manejadas en el código como objetos Java (es decir, se comparan los objetos correspondientes a las instancias involucradas).

5.2.4 Evaluación de precondiciones

La estructura de la precondición permite crear un procesamiento mediante un código genérico que pueda ser ejecutado de forma recursiva hasta resolver los anidamientos que existan.

Dentro de las precondiciones puede haber comparaciones, en cuyo caso se utilizará el procesamiento descrito en el punto 5.2.3. La función que evalúa la precondición consiste en aplicar los operadores lógicos AND y OR a un conjunto de condiciones y/o precondiciones. El resultado final de la evaluación de precondiciones es un valor de verdad (TRUE o FALSE).

La estructura de la precondición también podría ser traducida a un formato similar al de la parte antecedente de las reglas de producción.

5.2.5 Procesamiento de entidades y creación de triplas representando a estas entidades

La creación de las triplas que reflejan las propiedades o atributos de los conceptos se realiza procesando la semántica de la definición de las entidades e insertando en la base de conocimiento la instancia tripla correspondiente. Los diferentes tipos de entidades y su procesamiento se describen a continuación:

Entidades que representan y dan nombre al contenido de determinados slots (es decir, instancias de las clases `NamedMultipleCardinalitySlotInClass` *y* `NamedSingleCardinalitySlotInClass`*)*. En este caso el procesamiento consiste en recorrer el camino de slot indicado en el slot hasPath. El camino puede estar explícitamente formado por una lista de slots o puede estar formado por una lista de entidades que, en último término, puede expandirse a la lista de slots correspondiente.

El procesamiento recorre las instancias existentes de la clase indicada en el slot `hasBaseClass`. Para cada una de ellas se accede al primer slot indicado en el camino y se obtiene la instancia que se encuentra en ese slot. De esa instancia se accede al siguiente slot indicado en el camino y se obtiene la instancia que se encuentra en dicho slot, y así sucesivamente. El resultado final será una instancia, o una colección de instancias (el recorrido de caminos de slot y entidades se ve con más profundidad en la sección 5.2.7).

En el paso final se añade a la base de conocimiento la instancia tripla correspondiente que tendrá como campo “objeto” la instancia que está en el slot

`hasBaseClass`, como campo "atributo" la instancia de entidad que se está procesando y como campo "valor" la instancia o colección de instancias obtenidas al llegar al último slot existente en el camino de slots.

El proceso se repite para cada una de las instancias de este tipo de entidades.

Entidades que representan a sub-colecciones creadas a partir de otras entidades existentes (instancias de la clase `NamedSubcollectionOfObjects`). El procesamiento de estos conceptos es una combinación del anterior más la comprobación de la precondición que define qué elementos de la colección original pueden pertenecer a la nueva.

El resultado final será la tripla correspondiente que se insertará en la base de conocimiento.

Mediante el uso de reglas, en este caso, podría simplificarse el procesamiento, al utilizar las triplas obtenidas en el caso anterior

Entidades que representan a la intersección de sub-colecciones (instancias de la clase `NamedIntersectionOfCollections`). Esta estructura se utiliza para representar colecciones de objetos que se forman a partir de la intersección de colecciones existentes, es decir, que las precondiciones que entrarían a formar parte de la definición de la nueva colección ya están recogidas de forma separada en la definición de otras colecciones. Es el caso, por ejemplo, de los polos reales y del semiplano negativo. El procesamiento en este caso se limita a obtener los elementos comunes a las colecciones sobre las que se calcula la intersección.

Entidades que representan a una colección o sub-colección cuyos elementos están ordenados según una característica cuantitativa que se pueda aplicar a los conceptos que forman esa colección. El procesamiento en este caso consiste en evaluar la característica especificada en el slot `hasQuantitativeCharacteristicForOrdering` y ordenar la colección de acuerdo a estos valores. La ordenación será ascendente o descendente según se especifique en el slot `hasNumericalOrderingType`.

Entidades que representan a elementos dentro de una colección ordenada. El procesamiento de estos conceptos consiste en obtener el elemento que está en una determinada posición dentro de una colección ordenada de elementos. El procesamiento es sencillo en este caso, se accede a la instancia que representa la posición, esta instancia, además de un slot para el nombre, tiene otro para indicar el índice dentro de la colección. Este índice es un número entero mediante el que se accederá al elemento en cuestión. Una vez obtenido el elemento se introducirá la correspondiente tripla en la base de conocimiento.

5.2.6 Procesamiento de características y creación de triplas representando a estas características

Las características a procesar pueden ser de varios tipos:

- Cuantitativas, con dos posibles variaciones:
  - Representadas en el valor de un slot de un concepto.
  - Calculadas mediante una expresión.
  - Calculadas mediante una llamada a una función externa.
- Cualitativas

En ambos casos, la primera operación que se realizará será la comprobación de la condición de aplicabilidad para que pueda llevarse a cabo el cálculo de la característica. Esta comprobación consiste en la evaluación de la instancia que representa a la precondición y que está recogida en el slot `hasApplicabilityCondition` de la instancia de característica que se está procesando. Si la precondición se evalúa a un valor FALSE no se procesará la característica y, por tanto, no se introducirá la nueva tripla. Si la precondición resulta con un valor de verdad TRUE el procesamiento de la característica se realiza como sigue.

Características cuantitativas que coinciden con el valor de un slot en un concepto (instancias de la clase `QuantitativeCharacteristicAsNamedSlot`). El procesamiento de estas características es igual que el de las "entidades que representan y dan nombre al contenido de determinados slots", visto anteriormente. Estas características son a la vez entidades y su tratamiento es el mismo.

Características cuantitativas que pueden ser descritas mediante una expresión matemática (instancias de `QuantitativeCharacteristicAsExpression`). El procesamiento de estas características se realiza evaluando la expresión matemática (instancia de `CompoundExpression`) que aparece en el slot `hasDefiningExpression` tal como se explicó en el apartado 4.5.4.1.

Características cuantitativas cuyo valor se calcula mediante llamadas a funciones externas (instancias de `QuantitativeCharacteristicFunctionCalculated`). El procesamiento de estas características consiste en varios pasos:

- Obtener los parámetros a pasar a la función.
- Traducir la representación en la ontología a la del programa cuya función va a ser llamada.

- Realizar la llamada a la función.
- Traducir la representación del resultado en el sentido inverso, es decir, desde la aplicación externa a la estructura de la ontología.

La obtención de los parámetros a pasar a la función se realiza procesando un camino de slots o entidades a partir de la instancia de la que se dice la característica (es decir, de una instancia de la clase que está reflejada en el slot `hasBaseClass`).

Una vez obtenidos los parámetros de la ontología se traducen al formato de la aplicación que contiene la función que los procesará. De forma general esta traducción consistirá en crear una cadena de texto que contenga la declaración e inicialización de una variable cuyo contenido represente a ese tipo de dato en esa aplicación. La interfaz con las aplicaciones Maple y Matlab se basan en la evaluación de cadenas de caracteres que contienen sentencias válidas de esos programas.

La llamada a la función se realiza también mediante la evaluación de una cadena de caracteres, almacenándose el resultado devuelto en una variable.

La última tarea es la recuperación de la variable que contiene el resultado y la traducción inversa a una instancia de la ontología.

Para la traducción entre formatos de datos existe una función que tiene en cuenta el par tipo de datos de la ontología – tipo de datos de la aplicación externa.

Por último, mencionar un caso *especial de característica* que no se aplica a instancias de la ontología sino a colecciones de las mismas. La única característica que existe en este caso es el "número de elementos" (ver sección 4.5.4.1). En este caso la característica se traduce a la expresión similar en el lenguaje de programación que se emplee (método sizeof() de las colecciones Java, por ejemplo).

5.2.7 Procesamiento de los caminos de slots y entidades

Los caminos de slots representan, en la ontología, la estructura de conocimiento denominada encadenamiento de propiedades, aunque la expresividad construida va más allá de la definición habitual de esta estructura.

El camino de slots se representa como una lista ordenada de slots. Existirá, siempre que se use el camino de slots en alguna construcción en la ontología, una referencia a la instancia base a partir de la cual se irá accediendo a los slots indicados.

El camino de slots puede ser de dos tipos, que en la ontología se han denominado "hacia abajo" y "hacia arriba", correspondiéndose, respectivamente, con las clases `DownWardsSlotPath` y `UpWardsSlotPath`.

El camino de slots "hacia abajo" se correspondería con la estructura de encadenamiento de propiedades mencionada anteriormente y consiste en la obtención de una instancia o conjunto de instancias que se encuentra en un slot de una instancia que se encuentra en un slot de una instancia… (así sucesivamente)... que se encuentra en un slot (el primero de la lista) de la instancia de partida.

Cada slot recorrido debe dar lugar a una nueva instancia, ninguno de ellos, exceptuando el último puede ser un slot múltiple.

El camino de slots "hacia arriba" consiste en buscar la referencia a la instancia inicial en algún slot de otra instancia que se encuentre en el slot indicado en el camino. A partir de la instancia de partida se busca la instancia en cuyo slot (el primero del camino) aparece esta instancia de partida. Un ejemplo sería encontrar la instancia de modelo en función de transferencia, que tenga la instancia de cociente de polinomios, que tenga la instancia de polinomio denominador, que tenga la instancia de descripción en raíces y término principal, que tenga la instancia de raíz que es la instancia de la que se parte. Dicho de otro modo, sería acceder a "la función de transferencia donde aparece 3+2j como raíz del denominador".

Según el esquema construido, la instancia no puede aparecer referenciada más que en otra instancia para cada slot que hay en el camino ya que, en caso contrario, surgiría más de un camino posible.

Además, en la ontología se ha implementado una posibilidad de definir caminos de slots de forma más compleja y a más alto nivel. Esta posibilidad es el utilizar caminos de entidades, que, al igual que los caminos de slots, son listas ordenadas de entidades. En el procesamiento de un camino de entidades existen dos posibilidades. Cuando todas las entidades involucradas en un camino de entidades consisten en dar nombre a un camino de slots, el procesamiento se reduce a procesar el camino de slots total que se formaría uniendo todos los caminos de slots que hay en las entidades de la lista. Cuando en la lista entran en juego entidades más complejas (como por ejemplo sub-colecciones) el procesamiento se complica ya que, además del recorrido de slots, hay que procesar las condiciones que se establecen en las precondiciones.

5.3 Descripción de la aplicación

Todas las aplicaciones contienen una representación del dominio sobre el que trabajan. Habitualmente esta representación está construida por las variables y sentencias diseminadas por el código fuente, así como en los objetos software utilizados para su construcción. Diversos paradigmas de programación preconizan la separación entre los datos, el procesamiento de los mismos y la presentación al usuario. La aproximación de la ingeniería del conocimiento es separar y hacer explícito el modelo de conocimiento (no sólo datos, sino abstracción de los mismos) por medio de una, o varias, ontologías.

Los modelos de conocimiento reflejados en las ontologías están pensados para ser utilizados en las aplicaciones informáticas en tiempo de ejecución. En ciertos casos, como en las aplicaciones más comunes de la Web Semántica, este uso se basa en ser la fuente de conceptos con los que etiquetar (anotar) documentos distribuidos a través de Internet, así como facilitar los procesos de búsqueda y recuperación de la información. Existe otro tipo de aplicaciones en los que la ontología es la estructura donde están reflejados tanto los conceptos que utilizará la aplicación como la propia estructura dinámica que la aplicación ejecutará para resolver un problema dado. Esta segunda aproximación es la de los sistemas basados en el conocimiento, sucesores de los sistemas expertos. Entre uno y otro extremo existe una amplia variedad de tipos de aplicaciones que utilizan las ontologías de forma muy diferente.

Para que las ontologías salgan del ámbito académico y tengan aplicación práctica debe existir la posibilidad de acceder a su estructura de forma flexible desde el código de un programa informático. De esto depende incluso el éxito de los lenguajes y formalismos de representación del conocimiento ya que, desde el punto de vista práctico, la implantación de una tecnología informática depende de que se alcance una comunidad de usuarios y desarrolladores lo suficientemente grande como para mantenerla viva y hacerla evolucionar.

Los esfuerzos realizados durante los años 1990s dieron como fruto sistemas y lenguajes que nunca salieron del ámbito académico. La razón principal para que ocurriese esto es que no era sencillo el uso e integración de estos lenguajes y herramientas en una aplicación práctica (el énfasis en esa época estaba en los estudios teóricos). El punto de inflexión, en el paso de las ontologías al ámbito práctico, fue el desarrollo de interfaces de programación de aplicaciones (APIs) para el acceso a las estructuras de la ontología, en un primer momento, y a los motores de razonamiento posteriormente. En este sentido se pueden citar la API de Protégé para el formalismo de marcos y, posteriormente, la desarrollada para ontologías basadas en OWL. Del ámbito de la Web Semántica puede mencionarse Jena, la API más extendida para acceder a documentos RDF(S). En cuanto a APIs

para motores de razonamiento, se pueden citar la de Jess (basado en un sistema de reglas de producción) o la de Pellet (Sirin et. al., 2007) o RACER (Haarslev y Möller, 2003) (razonadores para lógicas descriptivas)[106].

Mediante la aplicación, descrita a continuación, se han probado varias de las posibilidades de interacción entre una aplicación informática y la ontología que recoge el modelo de conocimiento del dominio tratado. La aplicación construida utiliza la API de Protégé para que la interacción entre el usuario y los elementos de la interfaz gráfica sea, en realidad, una interacción con los conceptos de la ontología. En concreto, la aplicación permitirá las siguientes operaciones:

- Permitir la introducción de datos correspondientes a los sistemas involucrados y a las especificaciones de diseño. Los datos introducidos tendrán que ser recogidos como instancias de la ontología.
- Presentación de los conceptos introducidos y los creados mediante las definiciones de la ontología. A partir de la introducción de datos, todo el conocimiento de la ontología se aplicará a los mismos, de forma que se creará nuevo conocimiento (en forma de triplas) que consistirá en diversas afirmaciones sobre los elementos del diagrama de bloques en base datos concretos.
- Interactuación con el usuario, ofreciendo explicaciones sobre el por qué de las afirmaciones creadas, la definición y descripción de los conceptos, etc.

Para crear esta aplicación será necesario que las estructuras de la ontología estén asociadas a los elementos de la interfaz gráfica de usuario. A continuación se describen estas tres partes, indicando cómo se ha implementado esta relación y cómo se produce la interacción con el usuario. La figura 5.11 ofrece una visión general de la aplicación y los diferentes módulos que la forman.

5.3.1 Introducción de datos.

La introducción de datos al sistema consiste en explicitar las funciones de transferencia que describen a los sistemas involucrados en la topología de control. En este caso se introducirán las correspondientes al sistema a controlar y al controlador[107], según el esquema de la figura 5.1. La introducción de estos datos consistirá en dar de alta en la ontología instancias para los siguientes elementos (del más general al más específico y para cada función de transferencia):

[106] La creación del estándar DIG (DL Implementation Group) para el acceso a un razonador basado en lógicas descriptivas ha venido a facilitar todavía más el acceso y manejo por código de estos razonadores.

[107] Al no existir conocimiento dinámico que pueda realizar el diseño, se introduce manualmente la expresión del controlador. El lugar de las raíces representado será el obtenido al variar la constante de proporcionalidad del controlador.

- Instancia de `TransferFunctionSystemModel`, representando al modelo en función de transferencia del sistema.
- Instancia de `PolynomialQuotient`, que hará las veces de función de transferencia.
- Instancias de `Polynomial`, uno para el numerador y otro para el denominador de la función de transferencia.
- Instancias de `DescendingPowersOfVariablePolynomialDescription`, una por cada polinomio citado anteriormente.
- Instancias de `RealNumber`, tantas como coeficientes existan en los polinomios

La introducción de datos se hará presentando el diagrama de bloques de forma gráfica de forma que, pinchando en las cajas correspondientes a los sistemas mencionados, aparecerá la instancia de `TransferFunctionSystemModel` correspondiente.

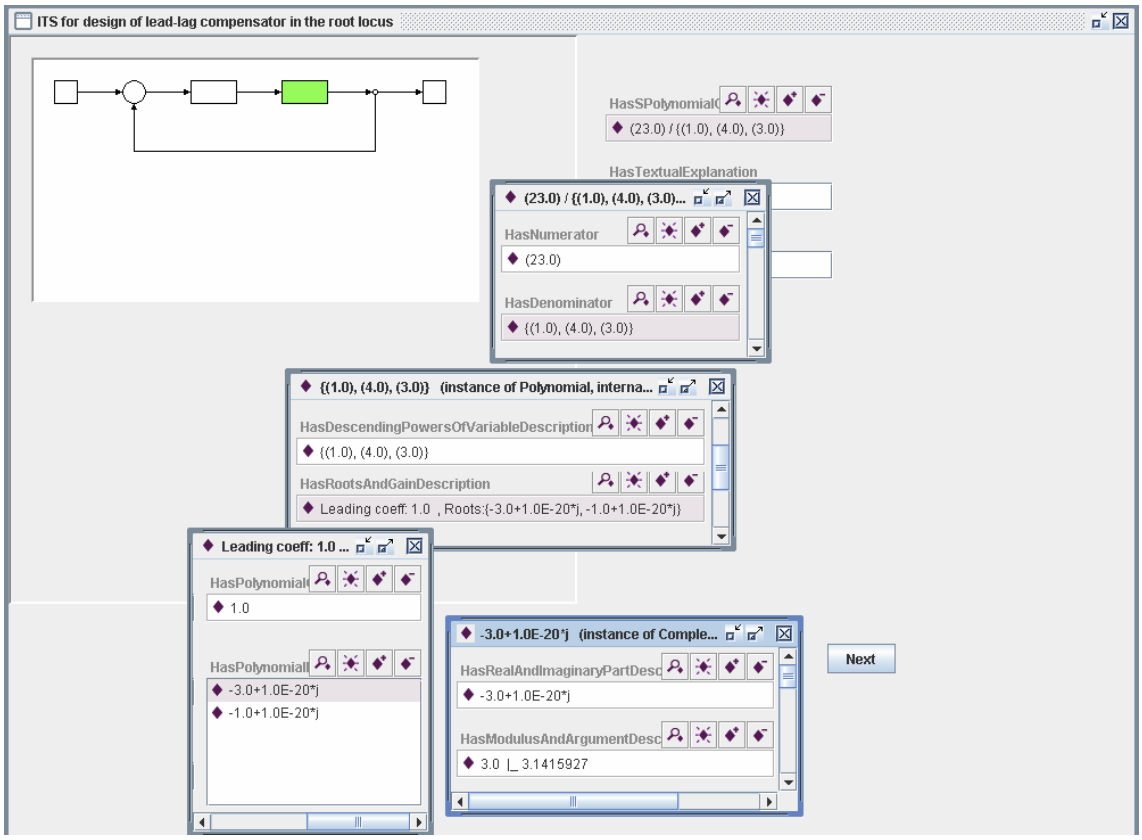


Figura 5.1. Formulario inicial de la aplicación.

Para representar gráficamente la topología de control se ha almacenado en la ontología la forma en la que los diferentes elementos de la topología deben aparecer gráficamente. Así, los bloques correspondientes a sistemas aparecen como cajas de un determinado tamaño, los enlaces como flechas, el sumador como un círculo y el punto de bifurcación como un círculo pequeño. Mediante la gestión de eventos, cada vez que se pincha en uno de estos elementos, aparece el formulario de Protégé asociado a la instancia correspondiente. Esto se ha hecho así aprovechando una de las funcionalidades de Protégé que consiste en la posibilidad de incluir los propios formularios utilizados en la herramienta al crear la ontología dentro de aplicaciones informáticas.

Una vez introducidos los datos de los sistemas se pasa al siguiente punto (se cambia de formulario Java en la aplicación), consistente en la introducción de los requerimientos de diseño.

La introducción de los requerimientos de diseño se hace presentando el formulario correspondiente a este concepto en la ontología. Los requerimientos son la base para la representación gráfica de la denominada "área de diseño". Serían también el punto de partida para la ejecución del proceso de diseño del controlador (no incluido en la ontología y aplicación presentes).

5.3.2 Presentación de conceptos.

Una vez introducidos los datos de los sistemas y las especificaciones de diseño se pasará a mostrar (en una ventana nueva) los conceptos existentes en la ontología que se han creado a partir de las definiciones recogidas en la misma y de estos datos introducidos. Entre este punto y el anterior se producirá la generación de nuevo conocimiento a partir de las estructuras de la ontología, proceso que será transparente para el usuario.

El proceso de creación de este nuevo conocimiento consistirá en calcular, en primer término, las funciones de transferencia de los denominados "bloques calculados" (es decir, todos los recogidos como instancias bajo la clase `CalculatedCanonicalBlock`). Una vez conocidos todos los bloques involucrados en la conceptualización se aplicarán las definiciones sobre las entidades y características existentes en la ontología, creando las triplas correspondientes (este proceso se realiza mediante el procesador descrito en el apartado anterior).

La ventana en la que se presentan los resultados al usuario está dividida en tres zonas, según se puede ver en la figura 5.2:

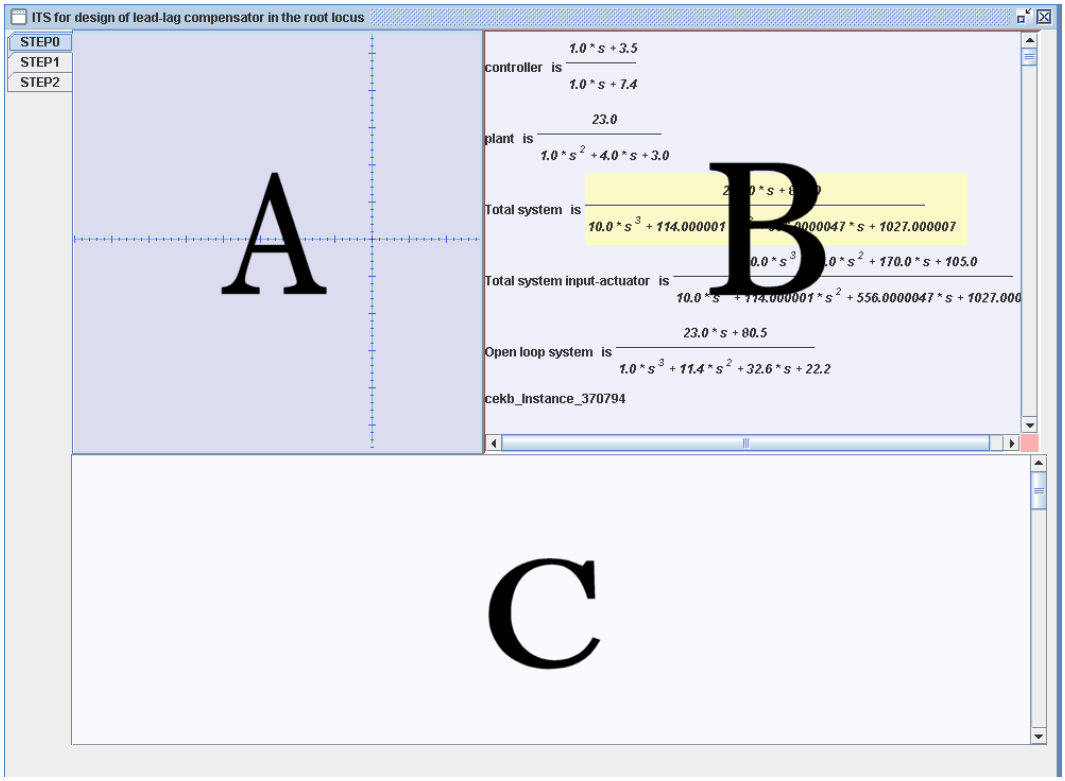


Figura 5.2. Ventana principal de presentación de información y conocimiento.

La parte de arriba de la ventana recoge dos de estas zonas. A la izquierda existe una zona (A) donde se representarán los elementos gráficos (polos, ceros, lugar de las raíces, área de diseño, etc.). En la parte derecha existe una zona (B) en la que, inicialmente, aparecerán los nombres de los sistemas involucrados, ya que éstos son el punto de partida para la obtención de cualquier conocimiento. Estos sistemas serán tanto los introducidos por el usuario como los definidos en la ontología bajo la clase de "bloques calculados" (`CalculatedCanonicalBlock`). Lo que se representa en esta zona son, pues, las instancias de la clase `Mapping`.

La representación de la información textual se realizará por medio de etiquetas de texto que tendrán asociadas las instancias correspondientes de la ontología de forma que, al interactuar con ellas, se pueda obtener el conocimiento asociado a estos conceptos. En la figura 5.3[108] se puede observar un ejemplo de esta asociación.

[108] Se ha omitido de la figura la información sobre los slots de las instancias para simplificar la representación.

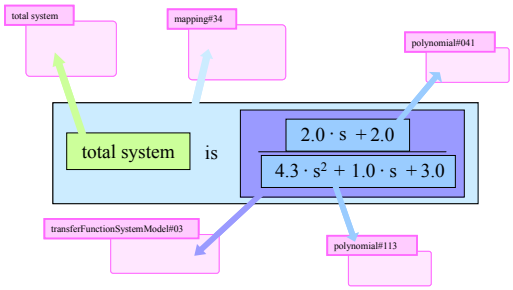


Figura 5.3. Estructura de una etiqueta y relaciones con instancias en la ontología.

Como puede verse, la estructura de una etiqueta puede ser compleja, incluyendo otras etiquetas dentro de ella (cada rectángulo con color de relleno representa una etiqueta) que estarán asociadas a otras instancias de la ontología. Esta asociación entre etiquetas e instancias de la ontología es la base para la interacción del usuario con los conceptos de la interfaz gráfica.

La interacción descrita anteriormente se llevará a cabo mediante el uso de menús contextuales, que presentarán al usuario el posible conocimiento que puede extraerse de cada concepto. En concreto, al pulsar el botón derecho sobre cada etiqueta se obtendrá un menú contextual que podrá contener (dependiendo del concepto que se trate):

- Una opción para obtener una descripción ó definición de ese concepto.
- Una opción para obtener una explicación sobre las razones para llegar a la afirmación que aparece expresada en la etiqueta.
- Opciones para obtener las entidades relacionadas con ese concepto.
- Opciones para obtener las características que se pueden aplicar al concepto.
- Opciones para mostrar/ocultar la representación gráfica (o resaltarlos si ya está mostrada) de los elementos en la zona dedicada a ello.

La información que aparezca en cada menú dependerá del tipo de conocimiento que el concepto sobre el que se ha pulsado el botón del ratón tenga asociado dentro de la ontología.

En la figura 5.4 puede verse un ejemplo de menú contextual creado sobre una instancia de la clase TransferFunctionSystemModel:

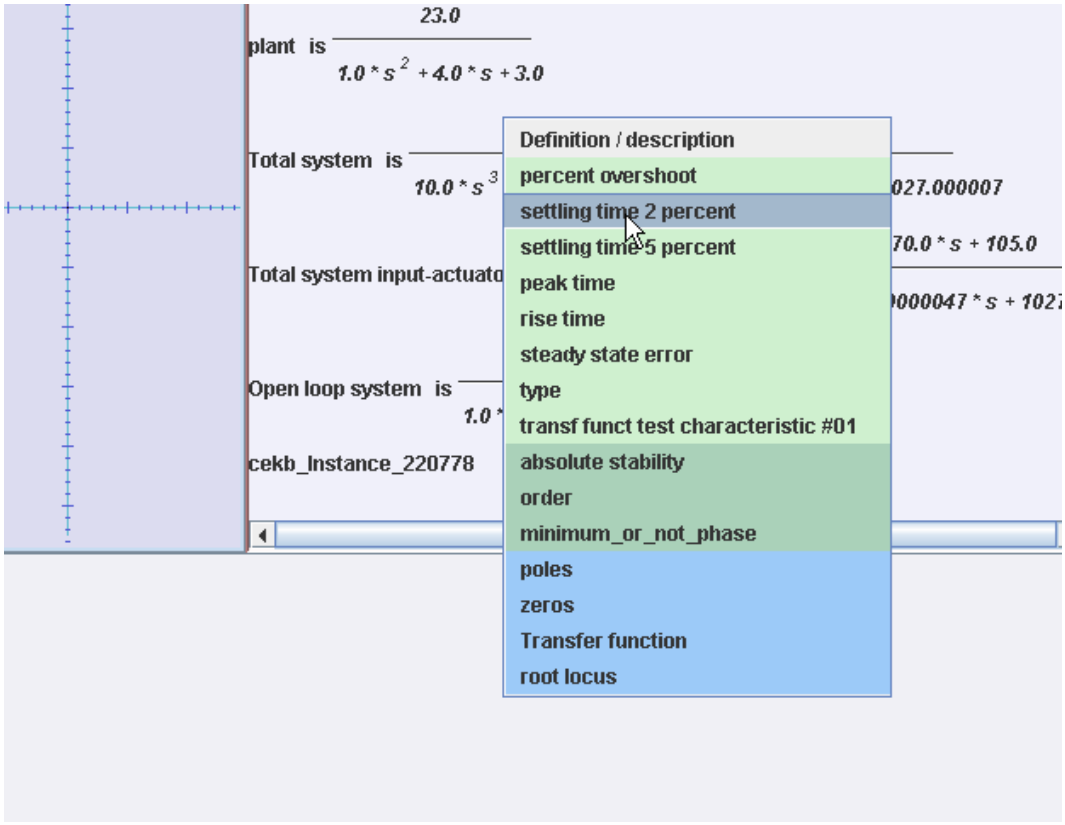


Figura 5.4. Menú contextual generado a partir de la ontología

Se han obtenido de la ontología las entidades y características (tanto cuantitativas como cualitativas) que pueden aplicarse a esas instancias y se han creado los correspondientes items para el menú.

El resultado de la interacción con las etiquetas descritas anteriormente aparecerá en la tercera zona de la ventana (zona C en la figura 5.2), la que ocupa la parte inferior. En esta zona la información aparecerá de forma similar, en forma de etiquetas, representando sentencias sobre el conocimiento almacenado en la ontología que, a su vez, tendrán el mismo modo de interacción que las descritas anteriormente.

La información que aparecerá en esta zona podrá ser de alguno de los siguientes tipos:

- Representación de triplas, provenientes de la base de conocimiento, que describen el valor de una cierta entidad asociada a un concepto.
- Representación de triplas que describen el valor de una cierta característica asociada a un concepto.

- Representación de la descripción ó definición de un concepto según esté especificado en la ontología.
- Representación de la explicación sobre las razones y razonamientos con las que se ha llegado a cierta afirmación.

Por ejemplo, si sobre la etiqueta que representa la función de transferencia de la planta "plant" se pide mostrar la entidad poles (ver figura 5.5), se tendrá un nuevo conjunto de etiquetas expresando:

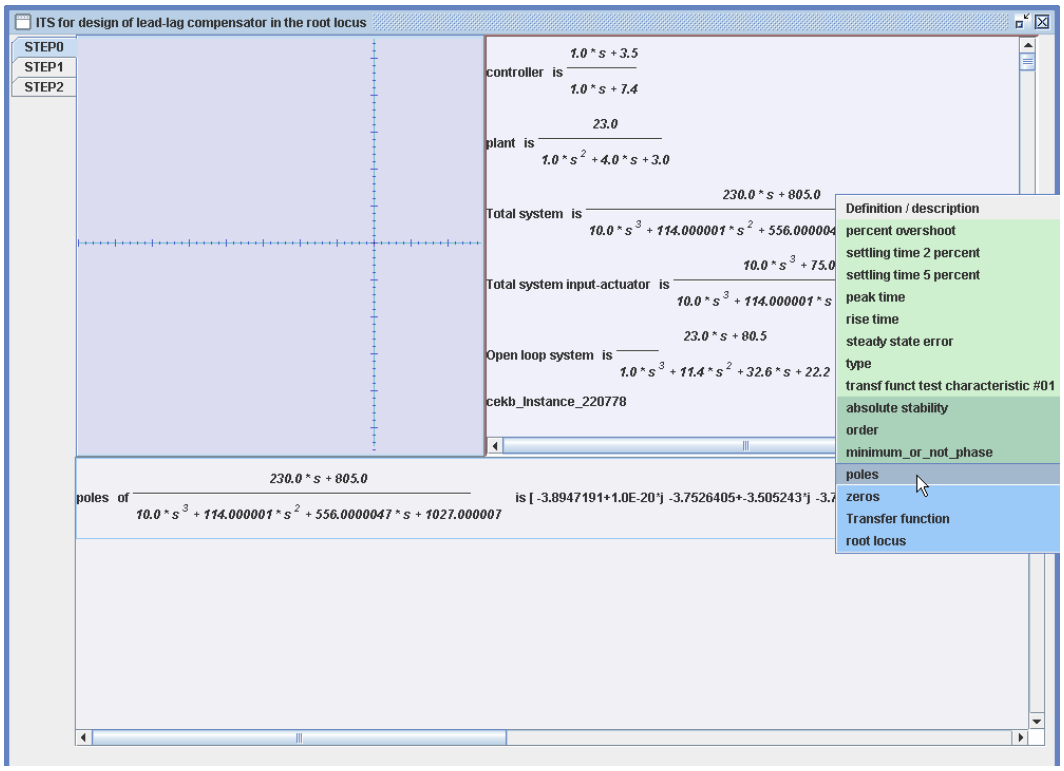


Figura 5.5. Representación de la tripla correspondiente a "polos del sistema total", generado a partir de la interacción con el menú contextual.

Esto es una afirmación sobre un concepto función de transferencia dado, es, en último término, el reflejo en la interfaz de una tripla de la base de conocimiento. La figura 5.6 muestra la instancia correspondiente a la tripla mostrada.

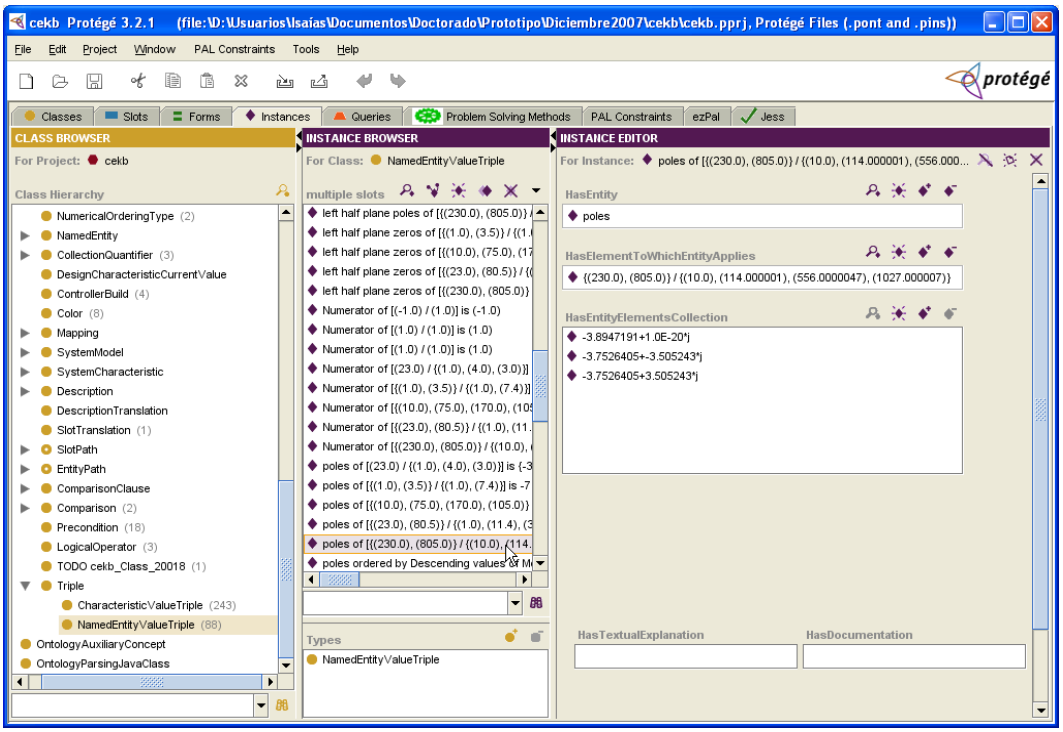


Figura 5.6. Instancia de tripla en la ontología.

La figura 5.7 muestra la representación de una tripla correspondiente a la característica cualitativa "estabilidad absoluta".

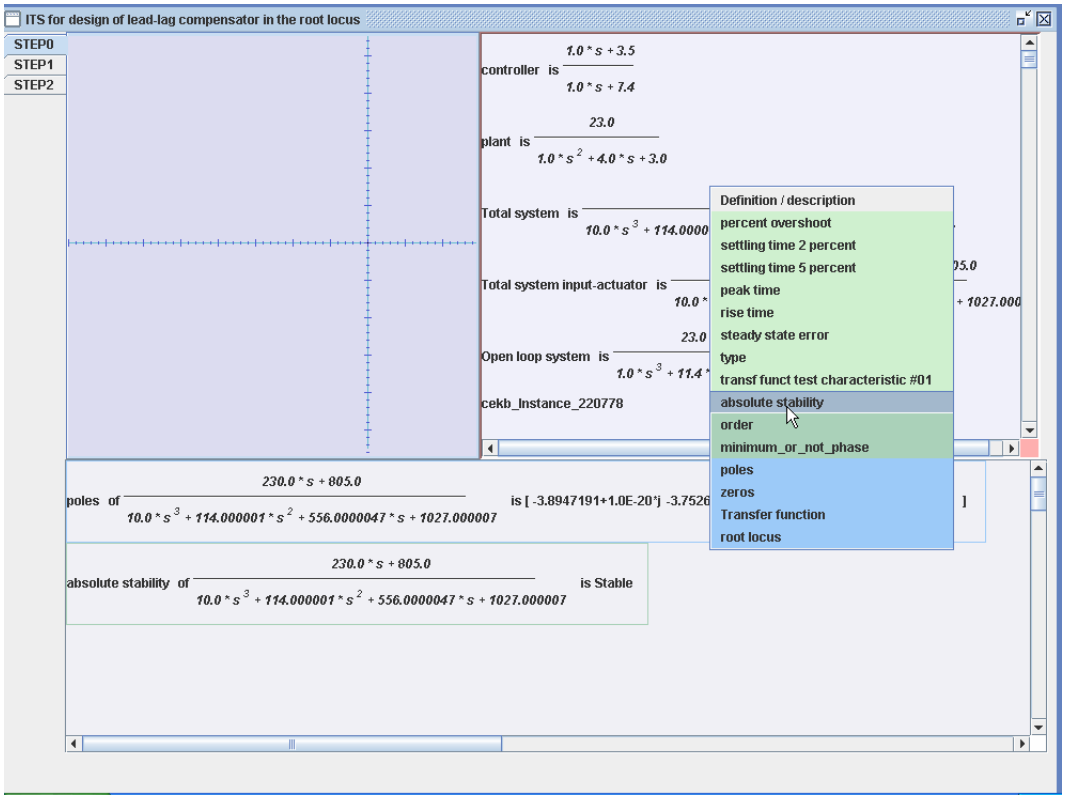


Figura 5.7. Representación de la tripla que recoge el valor de la característica cualitativa `absolute stability` aplicada a la instancia `total system`.

Sobre las afirmaciones obtenidas se puede, a su vez:

- Realizar preguntas sobre cómo se llega a esta afirmación.
- Interactuar con cada concepto de los que aparecen en la afirmación de forma similar a la descrita anteriormente.

Para implementar esta relación entre los conceptos de la ontología y los componentes en la interfaz gráfica de usuario se utilizan dos módulos dedicados a estos cometidos. El primero de ellos, el encargado de generar los componentes a partir de los conceptos en la ontología se denomina "generador de componentes". El segundo de ellos, dedicado a exponer las estructuras y/o definiciones de los conceptos a partir de la ontología, se denomina "generador de explicaciones".

5.3.2.1 Generador de componentes

El módulo "generador de componentes" se encarga de crear los siguientes elementos dentro de la interfaz gráfica de usuario:

- Etiquetas, que representan a los conceptos de la ontología. Las etiquetas son objetos cuya clase extiende a la clase JLabel de la librería Java swing, añadiendo propiedades para poder incluir una referencia a una instancia de la ontología. Todos los elementos de la ontología tendrán una representación textual y, por tanto, una representación por medio de etiquetas.
- Objetos gráficos, dedicados a los conceptos de la ontología que pueden ser representados en el diagrama de Argand (plano complejo). Estos objetos pueden ser:
    - Polos, representados mediante un aspa.
    - Ceros, representados mediante un círculo.
    - Requerimiento de diseño sobre tiempo de establecimiento, representado por una línea vertical.
    - Requerimiento de diseño sobre frecuencia amortiguada, representado por una línea horizontal.
    - Requerimiento de diseño sobre factor de amortiguación, representado una línea que pasa por el origen del plano.
    - Requerimiento de diseño sobre la frecuencia natural, representado por un círculo con centro en el origen del plano.
    - Conjunto de requerimientos de diseño, representado por un área de diseño formada por la combinación de las áreas del plano complejo que definen los diferentes requerimientos de diseño.
    - Lugar de las raíces, representado por un spline cúbico obtenido a partir de la información almacenada en el cálculo del lugar de las raíces. Existe la posibilidad de representar también segmentos del lugar de las raíces, formados por las ramas del mismo que existen entre dos valores de la constante de proporcionalidad determinados.
- Menús contextuales, que recogen la información que se puede obtener a partir de cada etiqueta tal como se indicó anteriormente.

La generación de etiquetas, junto con sus menús contextuales asociados, se realiza en el momento en que es necesario mostrarlas, es decir, al existir algún tipo de interacción por parte del usuario con los objetos que hay en la interfaz.

La función principal encargada de la generación de los elementos de la interfaz gráfica recibe como uno de sus parámetros la instancia de la ontología que debe ser representada. Dependiendo de la complejidad de esa instancia será necesario procesar otras instancias que forman parte de la principal. En la figura 5.3 se puede ver la representación de una instancia de tripla, distinguiendo las diferentes etiquetas que corresponden a las diferentes instancias que entran en juego.

Como puede comprobarse, la generación de etiquetas se encarga también de añadir los conectores adecuados ("of", "is", etc.) para que el resultado final sea una sentencia en "pseudo lenguaje natural".

Al construir cada etiqueta se asocia el menú contextual correspondiente, realizando consultas a la ontología con el fin de determinar las entidades y características relacionadas con ese concepto. Los gestores de eventos de los items de los menús son los encargados de lanzar los procesos relacionados con la pulsación en cada uno de ellos. Habitualmente, la consecuencia de la pulsación en un item será la generación de una nueva etiqueta mostrando la información requerida.

Generador de elementos gráficos

Los elementos susceptibles de tener una representación gráfica son tratados por este submódulo del módulo generador de componentes. A partir de la instancia a representar se genera el tipo de elemento correspondiente, tal como se ha citado anteriormente. Cada uno de los elementos gráficos lleva también asociada la instancia a la que representa.

El panel gráfico tiene asociado un array de elementos gráficos que es recorrido y representado de acuerdo a los tipos de elementos que aparecen en él. Merece especial mención el lugar de las raíces, que, antes de ser representado, necesita una transformación para generar los spline cúbicos a partir de la información almacenada en la ontología.

El ocultar un elemento en el panel gráfico consiste en marcarlo como invisible (una propiedad que se ha asociado a todos los elementos gráficos) y repintar el panel.

La figura 5.8 muestra la representación gráfica del lugar de las raíces del sistema total, incluyendo la representación de los polos complejos del sistema total para el valor de la ganancia actual. También se muestra una ventana para aplicar un zoom sobre la imagen.

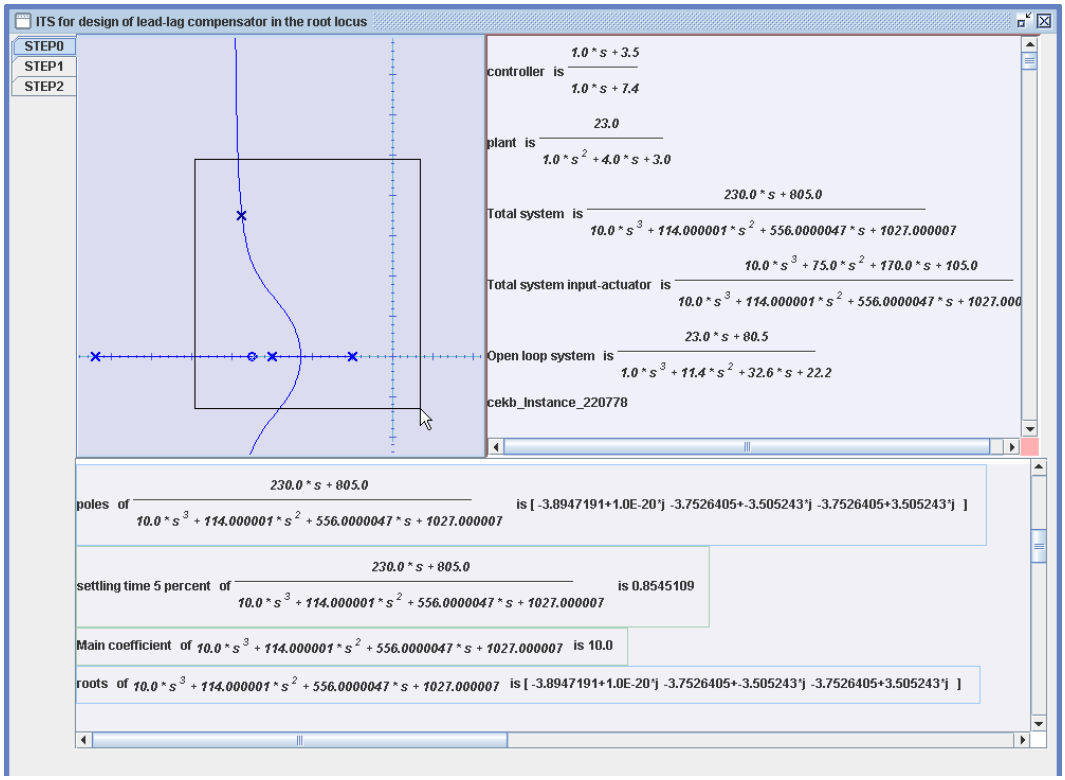


Figura 5.8. Representación gráfica de algunas instancias de la ontología.

5.3.2.2 Generador de explicaciones

El módulo "generador de explicaciones" se encarga de ofrecer explicaciones sobre las afirmaciones que aparecen en la interfaz gráfica de usuario. El tipo de explicaciones que se puede ofrecer puede ser:

- Sobre la descripción ó definición de los conceptos que aparecen (entidades, características, etc.).
- Sobre la forma en la que se calcula una determinada característica ó entidad (explicaciones sobre las triplas generadas).

Las explicaciones se generan a partir de la instancia que debe ser explicada, instancia que estará asociada a la etiqueta sobre la que se interactúa en la interfaz gráfica. La figura 5.9 es un ejemplo de explicación generada para definir el concepto `poles`.

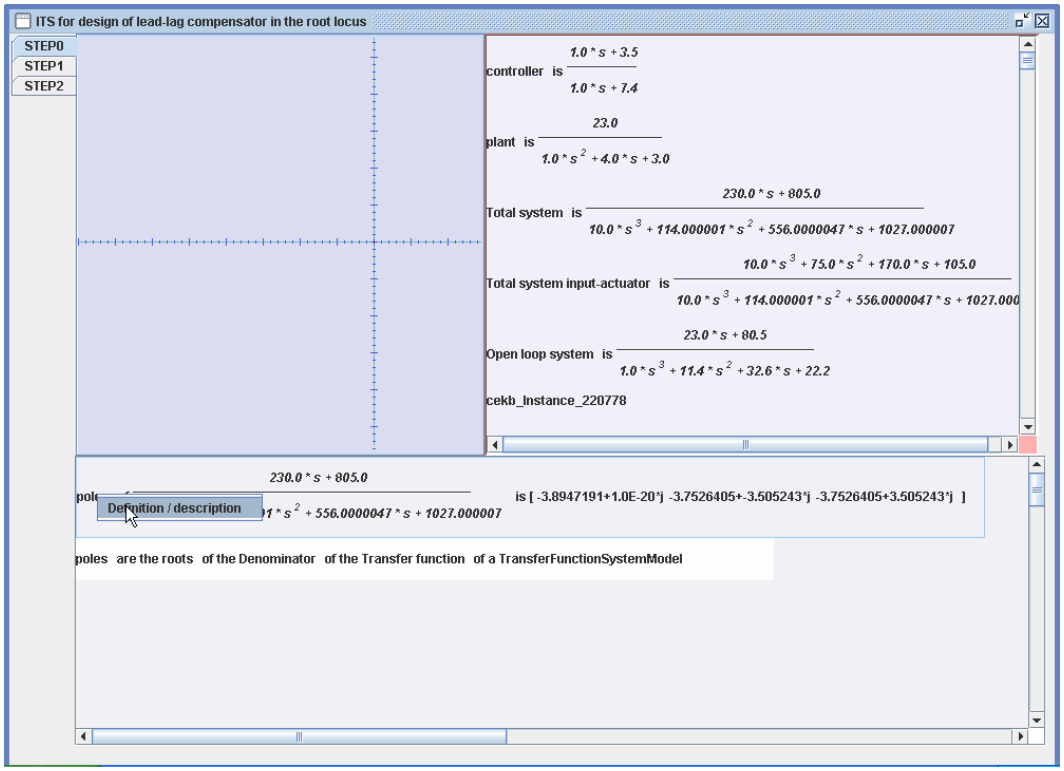


Figura 5.9. Explicación generada sobre el concepto poles a partir de la estructura del concepto en la ontología.

Esta explicación se genera a partir de la estructura creada para la clase `NamedMultipleCardinalitySlotInClass` y, en concreto, para la instancia `poles`, que pertenece a dicha clase.

La figura 5.10 recoge una explicación que es la respuesta a la pregunta "why?" efectuada sobre la afirmación correspondiente a la tripla que refleja el valor de la característica cualitativa "estabilidad absoluta" aplicada sobre el sistema total. La explicación consiste en explicitar las precondiciones que se cumplen y que están asociadas al valor cualitativo `stable` dentro de la instancia `absolute stability`.

La generación de explicaciones realizada es bastante básica y se basa, en parte, en la forma en la que los distintos conceptos están representados en la ontología. La conceptualización y formalización presentadas para las entidades y características permite obtener una expresión de los mismos en pseudo lenguaje natural realizando poco procesamiento.

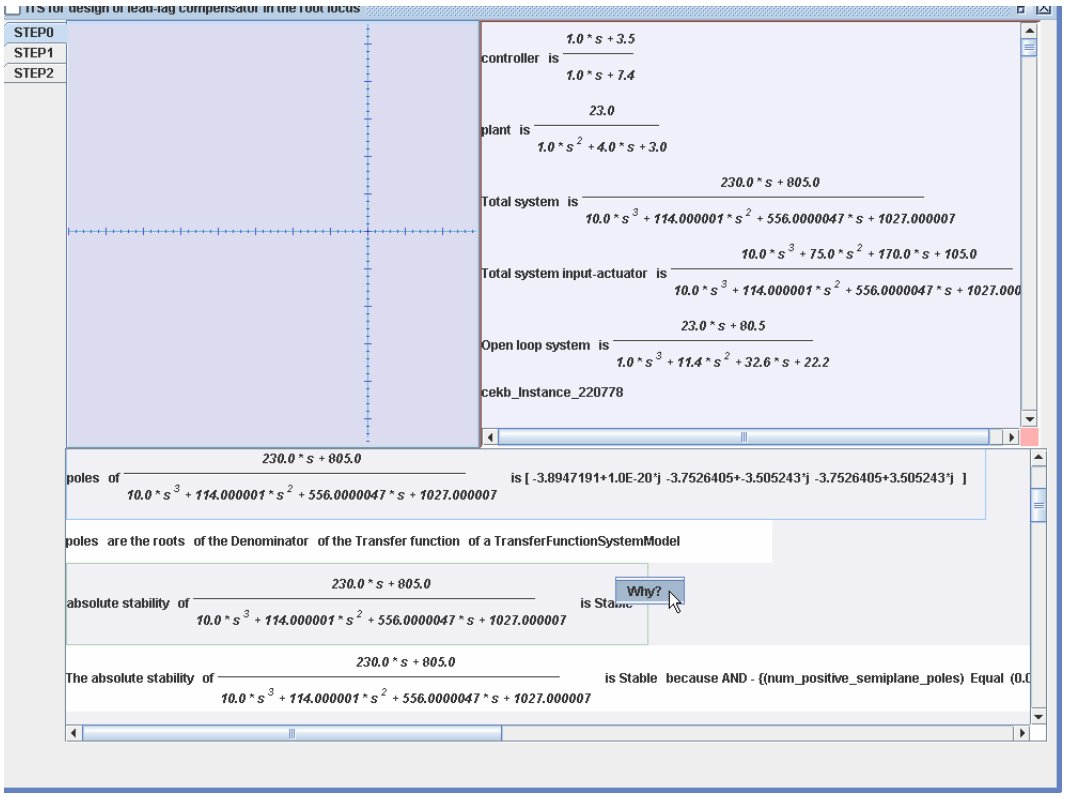


Figura 5.10. Explicación sobre la tripla que representa la estabilidad absoluta del sistema total.

En la figura 5.11 puede verse un diagrama simplificado de los diferentes módulos que componen las aplicaciones. Sobre fondo azul (con borde punteado grueso) se muestra la ontología, la base de conocimiento total y la aplicación para el procesamiento de las estructuras de conocimiento. Sobre fondo anaranjado (con borde continuo grueso) se muestra la aplicación que utiliza la ontología para presentar la información al usuario. También se han incluido en la figura los roles del ingeniero del conocimiento y del experto del dominio, que se encargarían de las tareas de creación y mantenimiento de la ontología, tal como se explicó en el apartado 1.1.1.

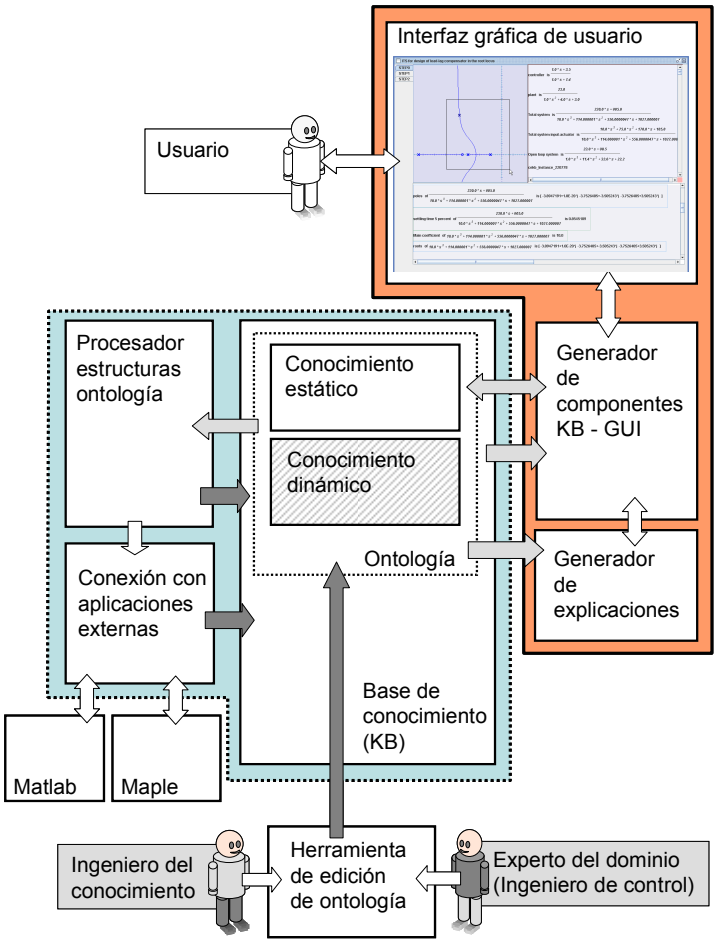


Figura 5.11. Esquema general de la aplicación procesadora de la ontología y la aplicación CACE (con trama rayada aparece la estructura dinámica, no descrita en esta tesis).

5.4 Conclusiones parciales

5.4.1 Sobre el procesamiento de las estructuras de la ontología.

La decisión de utilizar el formalismo de los marcos (u objetos estructurados) ha supuesto la necesidad de implementar la semántica de ciertas estructuras de conocimiento mediante un procesador programado a ese efecto[109]. Este hecho se produce por utilizar un cierto tipo de definición de conceptos, y no sólo ceñirse a

[109] La idea de procesador de estructuras de conocimiento (knowledge processor) está siempre presente (Devedžić, 1999), la diferencia es que, en el caso de las lógicas descriptivas, el formalismo lógico declarativo permite que la interpretación de la semántica sea bien conocida y el procesador sea siempre el mismo.

sus descripciones (caso en el que los marcos, sin más extensiones, podrían haber representado todo el conocimiento).

En el caso de la presente aplicación el procesador se ha programado en Java, aunque una aproximación más automatizada y que conservase la naturaleza declarativa del conocimiento sería la traducción de esas estructuras de conocimiento a la sintaxis de un lenguaje de reglas (Jess, por ejemplo). De hecho, la estructura de la definición de conceptos se ha hecho pensando en que ese tipo de traducción sea sencilla.

La alternativa, desde el punto de vista práctico, al formalismo de los marcos es, como se ha mencionado, el uso de OWL. Sin embargo, el haber utilizado OWL como lenguaje de representación también habría supuesto la necesidad de haber implementado parte de la semántica de forma separada, ya que la expresividad y los mecanismos automatizados de razonamiento con que cuenta este lenguaje no son de mucha ayuda para el tipo de conocimiento del dominio tratado, tal como se explicó en la sección 3.6.

En muchas aplicaciones realizadas con OWL también se utilizan reglas para extender su expresividad. El lenguaje SWRL (Semantic Web Rule Language) (Horrocks et. al., 2004) proporciona la posibilidad de construir expresiones de conocimiento que no son posibles en OWL. Esta estrategia tiene varios aspectos sensibles a tener en cuenta: en primer lugar, la expresividad construida en SWRL hace que el lenguaje sea indecidible (Rosati, 2006) y, por otro lado, la expresividad de SWRL se solapa en ciertas construcciones semánticas con la propia expresividad de OWL (se dice que ambos lenguajes son ortogonales), por lo que surge una nueva fuente para la existencia de posibles diferentes conceptualizaciones de las mismas estructuras así como posibles problemas de conceptualización.

En cualquier caso (se elijan marcos u OWL como formalismo), el uso de reglas de derivación es obligatorio para conseguir la expresividad necesaria. Puede considerarse que lo que se ha hecho en la ontología es construir (mediante las estructuras creadas para las entidades, características, comparaciones y precondiciones) un mecanismo para facilitar la introducción y mantenimiento de estas reglas, utilizando una conceptualización que aísla de la sintaxis de las mismas, facilitando su edición y la generación de explicaciones.

Las estructuras de conocimiento encontradas en el desarrollo de esta ontología muestran el tipo de expresividad que se requeriría de un lenguaje de representación del conocimiento para abordar la representación del conocimiento en el dominio de estudio. Desde este punto de vista, el tipo de formalismo utilizado para explicitar ese conocimiento no es tan importante. Además, el uso de

marcos como formalismo facilita la creación de estructuras ad-hoc que, en el caso de OWL, habría que haber adaptado a las expresiones del lenguaje.

Previsiblemente, la nueva versión de OWL permitirá representar ciertas estructuras que hasta ahora no pueden ser representadas aunque, de acuerdo a las previsiones conocidas[110], no todas las estructuras descritas en este trabajo podrán ser abordadas directamente en el lenguaje y, en el caso de las que sí lo serán, habría que cambiar la estrategia de representación, por ejemplo en el caso de las entidades definidas, que deberían ser tratadas como clases en vez de instancias, las particiones de valores de características, que tendrían que ser clases, etc. Por otro lado, existiría también la necesidad de implementar una estructura similar a la realizada para facilitar la tarea de adquisición del conocimiento correspondiente a las reglas SWRL, tal como se ha mencionado anteriormente.

En cualquier caso, el cambio de formalismo de representación es un hecho habitual en el ciclo de vida de las ontologías. Por ejemplo, una de las ontologías más influyentes en el ámbito de la biomedicina, desarrollada en el proyecto Foundational Model of Anatomy (FMA) (Rosse y Mejino, 2003), se está migrando desde el formalismo de marcos hacia OWL (Dameron et. al., 2005). El trabajo citado es también un ejemplo de la dificultad de traducción de las estructuras de conocimiento entre uno y otro formalismo. En el artículo se recoge el hecho de que no es posible pasar toda la expresividad existente en los marcos a OWL DL, debiendo utilizar OWL Full como lenguaje de representación. Además, se puntualiza que, debido a la intratabilidad computacional de este sublenguaje, el uso de la ontología en aplicaciones prácticas debe producirse extrayendo partes de la misma que, con una expresividad reducida, permitan el buen comportamiento en cuanto a complejidad computacional.

5.4.2 Sobre la aplicación con acceso a la ontología

La aplicación CACE desarrollada no se ha construido con una utilidad concreta en mente (aunque puede ser utilizada como herramienta de ayuda a la educación en control, sobre todo si se completa la parte dinámica de la ontología y se incluyen más conceptos en la misma), pero sí permite mostrar las posibilidades de la programación basada en modelos de conocimiento reflejado en ontologías.

La asociación de los elementos de la interfaz gráfica a las instancias existentes en la ontología es la base para la interacción del usuario con la aplicación. Esta asociación permite que el usuario interactúe con los conceptos de control, de

[110] En septiembre de 2007 se ha creado un nuevo grupo de trabajo (OWL working group) en el W3C que será el encargado de desarrollar la nueva versión de OWL, conocida como OWL 1.1. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

forma que toda respuesta recibida será construida a partir de la estructura de conocimiento subyacente.

Este tipo de interacción no aparece en ninguna aplicación CACE actual, aunque algunas herramientas, como Easy Java Simulations (EJS)[111] ó "interactive learning modules" (Guzmán et. al., 2006) intentan ofrecer un cierto nexo de unión entre los elementos de la interfaz y los conceptos subyacentes. En cualquier caso, estos nexos se establecen entre las variables de las ecuaciones diferenciales y la representación gráfica de una magnitud física (en el caso de Ejs) o mediante la inclusión de enlaces (hiperenlaces Web) a contenidos teóricos externos a la aplicación (en el caso de los interactive learning modules). En ambos casos la implementación de las conexiones conceptuales entre los elementos de la interfaz gráfica y los conceptos relacionados se realiza a un nivel diferente al aquí presentado.

Como puede comprobarse en las figuras que reflejan la aplicación, no existen botones ni menús en la misma, ya que toda la interactividad se implementa en base a los conceptos que aparecen en las etiquetas. De esta forma, cada usuario puede utilizar la aplicación de una forma muy diferente a la de otro usuario. De hecho, el conocimiento que se muestra como respuesta a las acciones del usuario acabará generando una especie de documento en el que se refleja el "historial de respuestas" ofrecidas por la aplicación, que será muy diferente para cada usuario.

Todas estas consideraciones cobran mayor importancia si se piensa en una ontología que recoja mayor cantidad de conceptos y/o incluya la estructura de conocimiento dinámico sobre el proceso de diseño de compensadores.

Las posibilidades de utilización de este tipo de software en el campo de la educación son numerosas. La interacción que el usuario (estudiante) realice con la aplicación puede ser utilizada para obtener un perfil del conocimiento del mismo, así como para saber en qué conceptos está más interesado. Mediante otro tipo de software más complejo, y utilizando la ontología de forma similar a la descrita, podrían evaluarse el tipo de conceptos que el estudiante no conoce y generar, de forma automática, problemas que sirvan para aclarar esos conceptos.

La conclusión más importante y más genérica es que el modelado mediante ontologías es la forma más adecuada de conseguir esta interrelación entre las interfaces de las aplicaciones CACE y los conceptos subyacentes. Esto no quiere decir que el mismo objetivo pudiese llevarse a cabo con otras herramientas, pero sí queda claro que mediante las descritas es como más adecuadamente se puede conseguir este objetivo.

[111] http://fem.um.es/Ejs

El uso de ontologías permite conservar de forma totalmente separada el conocimiento del dominio de aplicación informática que lo utiliza. Además, permite que ese conocimiento incluya la descripción y definición de los conceptos, así como las relaciones entre ellos, de forma que se facilite la generación automática de las representaciones de los conceptos en las interfaces y las explicaciones acerca de esas descripciones y definiciones.

Las aplicaciones CACE más elaboradas en cuanto a representación de la información utilizan, a menudo, bases de datos (orientadas a objetos o no) para representar esa información. La gran diferencia entre una base de datos y una ontología radica en el tipo de información que se almacena. Puede decirse que el conocimiento es la abstracción y generalización de grandes cantidades de datos. Así, un modelo de conocimiento recogido en una ontología sería un paso más allá que el diagrama entidad-relación de una base de datos.

Finalmente, debe decirse que la aplicación descrita en este capítulo es bastante básica en cuanto al uso que hace de las estructuras de la ontología. Existen varias posibilidades de aumentar los aspectos de interacción con el usuario. Una posible mejora sería crear una ontología aparte para definir cómo se representan los diferentes conceptos dentro de una interfaz gráfica de una aplicación. Esta ontología recogería los conceptos que aparecen en la construcción de interfaces gráficas (ventana, panel, etiqueta, etc.) y relacionaría los conceptos de la ontología de control con éstos, de forma que la generación de los componentes de la interfaz estaría también definida dentro de una ontología y no en un módulo de la aplicación como ocurre en los ejemplos mostrados (ver figura 5.11).

En cuanto a la aplicación en el campo de control, se pueden realizar diferentes mejoras tanto en la ontología como en la aplicación. Por ejemplo, se puede aplicar el modelado del conocimiento en ontologías a problemas de control más relevantes desde el punto de vista del diseño práctico. También puede mejorarse la cantidad de conocimiento que existe en la ontología actual, comenzando por complementarla mediante la conceptualización de la parte dinámica y generalizando el caso de la topología de realimentación unitaria y negativa a otras más complejas.

5.4.3 Sobre la evaluación de los resultados

La construcción de estas aplicaciones se ha realizado como forma de evaluar y verificar la aproximación defendida en esta tesis, así como la conceptualización realizada y reflejada en la ontología. Se ha comprobado, mediante pruebas exhaustivas, que es posible automatizar el procesamiento de las estructuras y obtener las explicaciones pertinentes acerca del conocimiento inferido, así como del conocimiento factual de la ontología. También se ha demostrado que es

posible construir una aplicación que muestre todo este contenido semántico al usuario a un nivel y mediante una forma de interacción novedosa para este tipo de software.

Debe decirse que las comprobaciones permiten afirmar que las estructuras de conocimiento creadas pueden recoger la definición, descripción y forma de cálculo de las entidades y características utilizadas en la teoría clásica de control; también que estas estructuras de conocimiento se comportan de forma correcta y su semántica puede ser procesada y presentada de forma automatizada. Pero es importante puntualizar también que no se ha pretendido reflejar de forma exhaustiva todas y cada una de las entidades y características particulares que aparecen en el dominio. El objetivo de la tesis es crear el modelo del conocimiento y demostrar su adecuación y bondades para construir software CACE, pero no el construir un modelo completo del dominio tratado.

Capítulo

6

Conclusiones finales y trabajos futuros

> El abuelo de Margie contó una vez que, cuando él era pequeño, su abuelo le había contado que hubo una época en que los cuentos siempre estaban impresos en papel.
>
> Uno pasaba las páginas, que eran amarillas y se arrugaban, y era divertidísimo ver que las palabras se quedaban quietas en vez de desplazarse por la pantalla. Y, cuando volvías a la página anterior, contenía las mismas palabras que cuando la leías por primera vez.
>
> *Isaac Asimov - Cuánto se divertían*

En el presente capítulo se detallan las conclusiones finales y las principales aportaciones realizadas en la realización de la presente tesis. Se indican también algunas de las posibles líneas futuras de investigación que pueden llevarse a cabo.

6.1 Conclusiones finales

Las principales conclusiones que han resultado del estudio llevado a cabo en la presente tesis son las siguientes:

- En el campo de la informática se está produciendo una evolución hacia el uso de modelos de conocimiento, no solo en las tareas del desarrollo de software, sino formando parte central de las propias aplicaciones y siendo la estructura fundamental en tiempo de ejecución. Se aúnan los esfuerzos e ideas de los campos de la ingeniería del software y la inteligencia artificial para poder conseguir esta evolución.
- El software para ingeniería de control necesita incrementar el nivel de representación de la información para hacer frente a las demandas actuales y futuras.

- Las técnicas y herramientas relacionadas con las ontologías son adecuadas para conseguir este nivel de representación necesario. Ningún otro paradigma o estrategia permite obtener los mismos resultados de manera más adecuada.
- El contenido semántico de la ingeniería de control se comunica mediante el uso de un lenguaje propio consistente en ofrecer términos y construcciones nuevas para fenómenos y entidades ya existentes en dominios como las matemáticas. Este es el hecho más relevante en la caracterización del conocimiento en control.
- Es posible realizar modelos de conocimiento del dominio de la ingeniería de control recogidos en ontologías, aunque los lenguajes existentes hoy en día no permiten una total representación de las estructuras de conocimiento necesarias, debiendo implementar parte de la semántica mediante procesadores externos.
- Es posible realizar aplicaciones en las que los elementos que aparecen en la interfaz gráfica con la que el usuario interactúa estén relacionados con los conceptos teóricos subyacentes si éstos están recogidos en una ontología. Además, se pueden generar explicaciones sobre las definiciones de estos conceptos y utilizar toda la estructura de conocimiento en la interacción con el usuario.

Las principales aportaciones concretas del presente trabajo pueden resumirse en los siguientes puntos:

- Se ha efectuado un estudio de la evolución y situación de las herramientas CACE desde el punto de vista de la representación del conocimiento utilizada en las mismas, abarcando desde los primeros sistemas hasta las propuestas de aplicación de técnicas de modelado del conocimiento.
- Se ha realizado un estudio de la evolución de la representación del conocimiento desde los sistemas expertos hasta las lógicas descriptivas y la Web Semántica, relacionando los aspectos claves que desde aquellos sistemas marcaron la evolución de los formalismos hasta la actualidad.
- Se ha realizado un estudio del conocimiento existente en la teoría clásica de la ingeniería de control con el fin de poder conceptualizarlo. El estudio va desde la caracterización general del mismo hasta la descripción de las estructuras conceptuales que lo puedan recoger en una ontología.
- Se ha creado una ontología para el dominio de estudio elegido, representando los conceptos más relevantes en ese ámbito.
- Se han identificado algunas estructuras de conocimiento importantes a la hora de conceptualizar el dominio de la ingeniería de control u otros similares. Ejemplos de estas estructuras pueden ser la representación de caminos de entidades y su recorrido hacia arriba y hacia abajo, el uso de

instancias canónicas, la representación de llamadas a funciones externas, etc.

- Se ha creado una aplicación que, utilizando la ontología, permite alcanzar un nuevo nivel de interacción entre la herramienta informática y el usuario de la misma, más allá de las capacidades multimedia y de interacción habitualmente implementadas. Estos resultados pueden ser aplicados directamente al campo de la educación en control y, de forma genérica, al campo del software CACE.

6.2 Trabajos futuros

Existe una amplia variedad de trabajos que pueden llevarse a cabo a partir de las ideas y experiencias obtenidas en el presente. Entre ellos pueden citarse los siguientes:

- El trabajo más importante, para el desarrollado en esta tesis, es complementar la estructura estática realizada con la conceptualización de la parte dinámica del dominio, consistente en la representación en la ontología de la estructura de tareas y sub tareas que describen el proceso de diseño de controladores de adelanto/retraso de fase. Este trabajo se está llevando a cabo ya en una tesis doctoral.
- Desde el punto de vista del desarrollo de la ontología ésta puede ser traducida y representada mediante el lenguaje OWL. Como se ha visto en los capítulos anteriores este lenguaje no tiene actualmente la expresividad suficiente para representar todas las estructuras necesarias en la ontología, por lo que habrá que complementarlo con representaciones del conocimiento basadas en reglas (que pueden ser escritas en SWRL, por ejemplo). La ventaja de llevar a cabo esta traducción entre formalismos es la mayor posibilidad de reutilización de la ontología, ya que OWL es actualmente el único lenguaje estandarizado para desarrollar ontologías (en Internet) además de contar con semántica basada en teoría de modelos (lo que facilita su reutilización). Las funcionalidades y nueva expresividad que se espera en la versión 1.1 de OWL puede hacer que sea viable la representación de la ontología desarrollada en este nuevo lenguaje (o al menos de parte de sus estructuras).
- Desde el punto de vista del conocimiento modelado se puede profundizar en el nivel de granularidad de los conceptos, creando representaciones más completas que puedan ser reutilizadas en otras conceptualizaciones (por ejemplo sobre los conceptos del dominio matemático). Puede también aumentarse la conceptualización en la dirección de los sistemas físicos, de forma que el modelo recoja el conocimiento desde los diagramas de parámetros agrupados y no directamente desde las funciones de

transferencia. Se pueden abarcar otras configuraciones de control (otras topologías), considerar sistemas con múltiples entradas y salidas, etc. Respecto al lugar de las raíces se podría trabajar también con el lugar de las raíces complementario, admitir la existencia de polos complejos en el controlador, utilizar graficas de ganancia y esquemas de fase, líneas de flujo de fuerza, etc.

- También puede ampliarse la conceptualización escogiendo otro dominio y desarrollando o complementando la ontología abarcando los controladores PID, conceptos y métodos de diseño mediante la respuesta en frecuencia u otros más modernos pero con gran contenido semántico como las técnicas de loop shaping, LQG/LTR, diseño de controladores H_∞, etc.
- Desde el punto de vista de las aplicaciones prácticas pueden generarse herramientas muy útiles en base a la ontología desarrollada. Entre ellas se pueden citar:
    - Construir tutores inteligentes completos, añadiendo modelos de conocimiento para el proceso de aprendizaje de la materia y el modelo del alumno.
    - Construir aplicaciones que funcionen en Internet, lo cual sería la aplicación más natural ya que las ontologías son la base de la Web Semántica. En este sentido se puede pensar en aplicaciones educativas que utilicen un mismo modelo de datos y por lo tanto puedan ser utilizadas de forma integrada por cualquier estudiante. Las ontologías pueden ser también la base para construir una capa unificadora para la descripción de los servicios educativos relacionados con el control en Internet, desde repositorios de materiales y aplicaciones hasta laboratorios virtuales y remotos, se utilizarían los emergentes servicios Web semánticos para describir todos los recursos mediante los términos almacenados en la ontología

Referencias

Nota: Todos los enlaces a los documentos electrónicos que aparecen en las siguientes referencias han sido comprobados con fecha de diciembre de 2007

(Alberts, 1994) Alberts, L. K. (1994) YMIR: A sharable ontology for the formal representation of engineering design knowledge. In J. S. Gero and E. Tyugu, editors, Formal Design Methods for CAD, IFIP (International Federation for Information Processing) Transactions, pp 3 - 32

(Alberts y Dikker, 1994) Alberts, L. K.; Dikker, F. (1994) Integrating standards and synthesis knowledge using the YMIR ontology. Artificial Intelligence in Design `94, J. S. Gero (ed.), Kluwer Academic Publishers, Boston. Online. Available:
http://citeseer.ist.psu.edu/rd/18513319%2C470814%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/24031/http:zSzzSzwwwkbs.cs.utwente.nlzSzPublicationszSz.zSz1994zSzalberts-aid94.pdf/alberts94integrating.pdf

(Alonso et. al., 1996) Alonso, F.; Juristo-Juzgado, N.; Maté, J. L.; Pazos, J. (1996) Software engineering and knowledge engineering: Towards a common life cycle. Journal of Systems and Software, 33(1), pp. 65 - 79

(Antoniou et. al., 2005) Antoniou, G.; Viegas Damásio, C.; Grosof, B.; Horrocks, I.; Kifer, M.; Małuszyński, J.; Patel-Schneider, P. F. (2005) Combining Rules and Ontologies: a survey. Deliverable I3-D3. Online. Available: <http://rewerse.net/deliverables/m12/i3-d3.pdf>

(Antoniou y van Harmelen, 2004a) Antoniou, G.; van Harmelen, F. (2004) Web Ontology Language: OWL. In S. Staab, R. Studer (eds.) Handbook on ontologies, Springer, pp. 67 – 92

(Antoniou y van Harmelen, 2004b) Antoniou, G.; van Harmelen, F. (2004) A Semantic Web Primer. The MIT Press.

(Antsaklis et. al., 1999) Antsaklis, P.; Basar, T.; DeCarlo, R.; McClamroch, N.H.; Spong, M.; Yurkovich, S. (1999) Report on the NSF/CSS Workshop on new directions in control engineering education. IEEE Control Systems Magazine, 19(5), pp. 53 - 58

(Artisan Software, 2007) Artisan Software. (2007) “Artisan Studio Tutorial” http://www.artisansw.com/support/downloads/tutorials/StudioTutorial62.zip

(Baader et. al. eds., 2007) Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; Patel-Schneider, D. F. (eds.) (2007) The description logic handbook: theory, implementation, and applications, Cambridge University Press

(Baader y Sattler, 2001) Baader, F.; Sattler, U. (2001) An overview of tableau algorithms for description logics, Studia Logica, 69(1), Springer-Verlag, pp. 5 - 40

(Baclawski et. al., 2002) Baclawski, K.; Kokar, M.; Kogut, P.; Hart, L.; Smith, J.; Letkowski, J.; Emery, P. (2002) Extending the Unified Modeling Language for ontology development. Software and Systems Modeling Journal, SoSyM, 1(2), pp. 142 - 156

(Barker, 1994) Barker, H. A. (1994) Open environments and object-oriented methods: the way forward in Computer-Aided Control System Design, in Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, 1994, pp. 3 - 12

(Batres et. al., 2007) Batres, R. West, M.; Leal, D.; Price, D.; Masaki, K.; Shimada, Y.; Fuchino, T.; Nakag, Y. (2007) An upper ontology based on ISO 15926. Computers and Chemical Engineering, 31(5-6), pp. 519 - 534

(Bechhofer et. al., 2000) Bechhofer, S.; Broekstra, J.; Decker, S.; Erdmann, M.; Fensel, D.; Goble, C.; van Harmelen, F.; Horrocks, I.; Klein, M.; McGuinness, D. L.; Motta, E.; Patel-Schneider, P.; Staab, S.; Studer, R. (2000) An informal description of standard OIL and instance OIL, online: http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf.

(Berneers-Lee, 1998) Berneers-Lee, T. (1998) Semantic Web Roadmap, http://www.w3.org/DesignIssues/Semantic.html

(Bilqees, 1996) Bilqees, A. (1996) Computing environments for control engineering. PhD. Thesis, Universidad de Cambridge. Online. Available: http://citeseer.ist.psu.edu/rd/75021831%2C610741%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/29125/http:zSzzSzwww-control.eng.cam.ac.ukzSzabszSzthesis.pdf/computing-environments-for.pdf

(Bissell, 1993) Bissell, C. (1993) A new way of talking: aspects of the creation of the language of control engineering, Faculty of Technology/Systems Architecture Group Internal Report (SAG/1993/RR31/CCB), November 1993

(Bissell, 1999) Bissell, C. C. (1999) Control education: time for radical change? IEEE Control Systems Magazine, 19(5), pp. 44 - 49

(Bissell, 2004) Bissell, C. (2004) Mathematical 'meta-tools' in 20th century information engineering. Hevelius, Vol. 2, pp. 11-21 (Published version of a paper presented at: ICOTEC2003 (XXX Symposium of the International Committee for the History of Technology), 21-26 August 2003, St. Petersburg/Moscow

(Bissell y Dillon, 2000) Bissell, C.; Dillon, C. (2000) Telling tales: models, stories and meanings. For the Learning of Mathematics, 20(3), pp. 3 - 11

(Bobrow y Winograd, 1976) Bobrow, D. G.; Winograd, T. (1976) An overview of KRL, a knowledge representation language. Stanford Artificial intelligence Laboratory. Memo AIM-293. Computer Science Department. Report No. STAN-E-76-58 1. Online: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/76/581/CS-TR-76-581.pdf. También publicado en: Cognitive Science, 1(1), 1977

(Booch et. al., 2005) Booch, G.; Rumbaugh, J.; Jacobson, I. (2005) The Unified Modeling Language user guide, 2nd ed. Pearson

(Borgida et. al., 1989) Borgida, A.; Brachman, R. J.; McGuiness, D. L.; Resnick, L. A. (1989) CLASSIC: A structural data model for objects. In Proceedings of the 1989 ACM SIGMOD international conference on Management of data, pp. 58-67

(Borst, 1997) Borst, W.N. (1997) Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD Thesis, Universidad de Twente, Holanda. Online: http://doc.utwente.nl/17864/1/t0000004.pdf

(Borst et. al., 1997) Borst, P.; Akkermans, H.; Top, J. (1997) Engineering ontologies. International Journal of Human-Computer Studies, 46(2-3), pp. 365 - 406

(Brachman, 1977) Brachman, R. J. (1977) What's in a concept: Structural foundations for semantic networks. International Journal of Man-Machine Studies, 9, pp. 127 - 152,

(Brachman, 1978) Brachman, R. J. (1978) A structural paradigm for rep- resenting knowledge. Technical Report 3605, Bolt Beranek and Newman, Cambridge, MA, 1978

(Brachman y Levesque, 1985) R. J. Brachman & H. J. Levesque, eds. (1985) Readings in Knowledge Representation, Morgan Kaufmann

(Brachman y Schmolze, 1985) Brachman, R. J.; Schmolze, J. (1985) An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2), pp. 171-216

(Brachman et. al., 1991) Brachman, R. J.; McGuiness, D. L.; Patel-Schneider, P. F.; Resnick, L. A. (1991) Living with CLASSIC: when and how to use a KL-ONE -like language. In Principles of semantic networks, John Sowa ed., Morgan Kaufmann, San Mateo, US

(Breunese et. al., 1997) Breunese, A. P. J.; Top, J. L.; Broenink, J. F.; Akkermans J. M. (1998) Libraries of Reusable models: theory and application. Simulation, 71(1), pp. 7 - 22

(Brewster y O'Hara, 2004) Brewster, C.; O' Hara, K. (2004) Knowlege representation with ontologies: the present and future. IEEE Intelligent Systems, 19(1), pp. 72 - 81

(Brickley y Guha eds., 2004) Brickley, D.; Guha, R. V. (editors) (2004) RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, 10 February 2004, Online: <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>

(Brown, 2004) Brown, A. W. (2004) Model driven architecture: Principles and practice. Software and System Modeling 3(4), pp. 314 – 327

(Burek, 2004) Burek, P. (2004) Adoption of the Classical Theory of Definition to Ontology Modeling. In Proceedings of the 11th International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA), 2-4 September, Varna
(Bulgaria), Lecture Notes in Computer Science, Springer, Volume 3192/2004, pp. 1 - 10

(Burek, 2005) Burek, P. (2005) Essentialized conceptual structures in ontology modeling. In Proceedings of the KES International Conferences in Knowledge-Based and Intelligent Engineering Systems, Lecture Notes y Computer Science, Springer, Volume 3682/2005, pp. 880-886

(Butz et. al., 1990) Butz, B. P.; Palumbo, N. F.; Unterberger, R. C. Jr. (1990) An expert system for control system design. In: Proceedings of the 5th IEEE International Symposium on Intelligent Control, vol 2., pp. 1156-1162

(Bylander y Chandrasekaran, 1988) Bylander, T.; Chandrasekaran, B. (1988) Generic tasks in knowledge-based reasoning: the right level of abstraction for

knowledge acquisition. In B. R. Gaines and J. H. Boose (eds.), Knowledge Acquisition for Knowledge Based Systems. Academic Press, London, pp. 65 - 77

(Chandra y Tumanyan, 2007) Chandra, C.; Tumanyan, A. (2007) Organization and problem ontology for supply chain information support system. Data & Knowledge Engineering, 61(2), pp. 263 - 280

(Chandrasekaran et. al., 1999) Chandrasekaran, B.; Josephson, J.R.; Benjamins, V.R. (1999) What are ontologies, and why do we need them? Intelligent Systems, 14(1), pp. 20-26

(Chaudhri et. al., 1998a) Chaudhri, V. K.; Farquhar, A.; Fikes, R.; Karp, P. D.; Rice, J. P. (1998) OKBC: a programmatic foundation for knowledge base interoperability. Proceedings of AAAI-98, Madison, Wisconsin

(Chaudhri et. al., 1998b) Chaudhri, V. K.; Farquhar, A.; Fikes, R.; Karp, P. D.; Rice, J. P. (1998) Open Knowledge Base Connectivity 2.0.3. Online: http://www.ai.sri.com/~okbc/okbc-2-0-3.pdf

(Chella et. al., 2002) Chella, A.; Cossentino, M.; Pirrone, R.; Ruisi, A. (2002) Modeling ontologies for robotic environments. In Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM International Conference Proceeding Series; Vol. 27, Ischia, Italy, pp. 77 - 80.

(Corcho et. al., 2003) Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. (2003) Methodologies, tools and languages for building ontologies. Where is their meeting point? Data and Knowledge Engineering, 46(1), pp. 41 – 64.

(Cranefield y Pan, 2007) Cranefield, S.; Pan, J. (2007) Bridging the gap between the model-driven architecture and ontology engineering. International Journal of Human-Computer Studies, 65(7), pp. 595 – 609.

(Cranefield y Purvis, 1999) Cranefield, S.; Purvis, M. (1999) UML for ontology development. In Proceedings of the Workshop III-99 at IJCAI-99, Stockholm, Sweden.

(Cutkosky et. al., 1993) Cutkosky, M.; Engelmore, R. S.; Fikes, R. E.; Gruber, T. R.; Genesereth, M. R.; Mark, W. S.; Tenenbaum, J. M.; Weber, J. C. (1993) PACT: An experiment in integrating concurrent engineering systems. IEEE Computer, 26(1), pp. 28 – 37.

(Dameron et. al., 2005) Dameron, O.; Rubin, D. L.; Musen, M. A. (2005) Challenges in converting frame-based ontology into OWL: the Foundational Model of Anatomy case-study. In Proceedings of the AMIA - American Medical Informatics Association - 2005, pp. 181 – 185.

(Damjanović et. al., 2004) Damjanović, V.; Devedžić, V.; Djurić, D.; Gašević, D. (2004) Framework for analyzing ontology development tools. AIS SIGSEMIS Bulletin, 1(3), pp. 43 – 47.

(Devedžić, 1999) Devedžić, V. (1999) A survey of modern knowledge modelling techniques. Expert Systems with Applications, 17, Elsevier, pp. 275 – 294.

(Djuric et. al., 2005) Djuric, D.; Gaševic, D.; Damjanović, V.; Devedžić, V. (2005) MDA-Based Ontological Engineering", in: Chang, S.K. (ed.), Handbook of Software Engineering and Knowledge Engineering Vol.3 - Recent Advances, World Scientific Publishing Co., Singapore, pp. 203-231

(Djuric et. al., 2007) Djuric, D.; Devedžić, V.; Gaševic, D. (2007) Adopting software engineering trends in AI, IEEE Intelligent Systems, 22(1), pp. 59 – 66.

(Donini et. al., 1996) Donini, F. M.; Lenzerini, M.; Nardo, D.; Schaerf, A. (1996) Reasoning in description logics. In Brewka, G. (ed.) Principles of knowledge representation, {CSLI} Publications, Stanford, California, pp. 191 - 236. Online. Available: < http://citeseer.ist.psu.edu/rd/75021831%2C131764%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/2710/http:zSzzSzwww.cs.man.ac.ukzSz%7EfranconizSzdlzSzcoursezSzarticleszSzreasoning-survey.pdf/donini97reasoning.pdf >

(Dormido, 2002) Dormido, S. (2002). Control learning: present and future. In: Proceddings of the 15th IFAC World Congress, Barcelona (Spain).

(Dormido, 2004) Dormido, S. (2004) Control learning: present and future, Annual Reviews in Control, 28, pp. 115 – 136.

(dos Santos y Vrancken, 2007) dos Santos, M.; Vrancken, J. (2007) An Integrated method based on multi-models and levels of modeling for design and analysis of complex engineering systems. In Proceedings of the 21st European Conference on Object-Oriented Programming, Doctoral Symposium and PhD Workshop. July 30 - August 3, 2007, Berlin, Germany, pp. 35 – 45.

(Drummond et. al., 2006) Drummond, N.; Rector, A.; Stevens, R.; Moulton, G. (2006) Putting OWL in order: patterns for sequences in OWL. In Proceedings of

OWLED 2006 - OWL: Experiences and Directions Second International Workshop, Athens, Georgia, USA.

(Duineveld et. al., 1999) Duineveld, A.; Stoter, R.; Weiden, M. R.; Kenepa, B.; Benjamins, V. R. (1999) Wonder Tools? A comparative study of ontological engineering tools. In Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling, and Management (KAW99), 16-21 October, Banff, Canada.

(Dutton et. al., 1997) Dutton, K.; Thompson, S.; Barraclough, B. (1997) Beautiful theories and ugly facts. In The art of control engineering, Prentice Hall, pp. 25 - 26 (apartado 1.3.7).

(Elmqvist, 1978) Elmqvist, H. (1978) A Structured Model Language for Large Continuous Systems. Phd thesis, Department of Automatic Control, Lund Institute of Technology, Lund.

(Farquhar et. al., 1996) Farquhar, A. Fikes, R.; Rice. J. (1996) The ontolingua server: a tool for collaborative ontology construction. Technical Report KSL 96-26, Stanford University, Knowledge Systems Laboratory, 1996.

(Feigenbaum, 1977) Feigenbaum, E. A. (1977). The art of artificial intelligence: themes and case studies of knowledge engineering. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA.

(Feigenbaum, 1980) Feigenbaum, E. (1980) Knowledge engineering: the applied side of artificial intelligence, STAN-CS-80-812 Department of Computer Science, Stanford University.

(Feigenbaum, 1992) Feigenbaum, E. (1992) A personal view of expert systems: looking back and looking ahead. Expert Systems with Applications, 5, pp. 193 – 201.

(Fernández et. al., 1997) Fernández, M., Gómez-Pérez, A. Juristo, N. (1997) METHONTOLOGY: from ontological art toward ontological engineering. Spring Symposium Series on Ontological Engineering. AAAI'97. Stanford. USA.

(Fernández, 1999) Fernández, M. (1999) Overview of methodologies for building ontologies. In proceedings of IJCAI99's Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends, pp. 4.1 - 4.13.

(Fikes et. al., 1991) Fikes, R.; Gruber, T.; Iwasaki, Y.; Levy, A.; Nayak, P. (1991) How things work - project overview - Knowledge Systems Laboratory Technical

Report KSL-91-70; Computer Science Department, Stanford University. Online: http://citeseer.ist.psu.edu/rd/75021831%2C322066%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/14917/ftp:zSzzSzksl.stanford.eduzSzpubzSzKSL_ReportszSz.zSzKSL-91-70.pdf/how-things-work-project.pdf

(Fikes y Kehler, 1985) Fikes, R.; Kehler, T. (1985) The role of frame-based representation in reasoning, Communications of the ACM, 28(9), pp. 904 – 920.

(Finin et. al., 1992) Finin, T.; Weber, J.; Wiederhold, G.; Genesereth, M.; Fritzson, R.; McKay, D.; McGuire, J.; Pelavin, P.; Shapiro, S.; Beck, C. (1992) Specification of the KQML Agent-Communication Language. Enterprise Integration Technologies, Palo Alto, CA, Technical Report EIT TR 92-04. Online: http://www-ksl.stanford.edu/knowledge-sharing/papers/kqml-spec.ps

(Friedman-Hill, 2007) Friedman-Hill, E. et al. (2007) JESS (Java Expert System Shell) Online: http://herzberg.ca.sandia.gov/jess

(Gamma et. al., 1995) Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995) Design patterns: elements of reusable object-oriented software, Addison-Wesley.

(García et. al., 2006) Garcia, A.; Berenguel, M.; Guzman, J. L.; Dormido, S.; Domínguez, M. (2006) Remote laboratory for teaching multivariable control techniques. Proceedings of the 7th IFAC Symposium on Advances in Control Education.

(Gaševic et. al., 2004) Gaševic, D.; Djuric, D.; Devedžić, V.; Damjanović, V. (2004). Approaching OWL and MDA through technological spaces. In Proceedings of the 3rd Workshop in Software Model Engineering (WiSME 2004)

(Gaševic y Devedžić, 2006) Gaševic, D.; Devedžić, V. (2006) Petri net ontology. Knowledge-Based Systems, 19(4), pp. 220 - 234.

(Genari et. al., 2002) Gennari, J. H.; Musen, M. A.; Fergerson, R. W.; Grosso, W. E.; Crubézy, M.; Eriksson, H.; Noy, N. F.; Tu. S. W. (2002) The evolution of Protégé: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford Medical Institute.

(Genesereth y Fikes, 1992) Genesereth, M. R.; Fikes, R. (in association with Bobrow, D; Brachman, R.; Gruber, T.; Hayes, P.; Letsinger, R.; Lifschitz, V.; MacGregor, R.; McCarthy, J.; Norvig, P.; Patil, R.; Schubert, L.) (1992) Knowledge Interchange Format (KIF) ver. 3.0 Reference Manual. Online: http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps

(Genesereth y Nilsson, 1987) Genesereth, M. R., Nilsson, N. J. (1987) Logical foundations of artificial intelligence, Morgan Kauffman, Los Altos, California.

(Gómez-Pérez y Corcho, 2002) Gómez-Pérez, A.; Corcho, O. (2002) Ontology languages for the Semantic Web, IEEE Intelligent Systems, 17(1), pp. 54 – 60.

(Gómez-Pérez, 2004) Gómez-Pérez, A. (2004) Ontology evaluation. In S. Staab, R. Studer (eds.) Handbook on ontologies, Springer, pp. 251 – 274.

(Grübel, 1994) Grübel, G. (1994) The ANDECS-CACE framework A-RSYST for integrated analysis and design of controlled systems. In Proceedings of IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, pp. 389 – 394.

(Grübel, 1995) Grübel, G.(1995) The ANDECS CACE Framework. IEEE Control Systems, 15(2), pp. 8 – 13.

(Gruber, 1993) Gruber, T. R. (1993) A translation approach to portable ontology specifications. Knowledge Adquisition, 5(2), pp 199 – 220.

(Gruber, 1995) Gruber, T. R. (1995) Towards principles for the design of ontologies used for knowledge sharing. International Journal of Human and Computer Studies, vol. 43, pp. 907 – 928.

(Gruber, 2004) Gruber, T. (2004) Every ontology is a treaty - a social agreement - among people with some common motive in sharing. Entrevista al autor en Bulletin of AIS Special Interest Group on Semantic Web and Information Systems (SIGSEMIS), 1(3).

(Gruber e Iwasaki, 1990) Gruber, T.; Iwasaki, Y. (1990) How things work: knowledge-based modeling of physical devices. Informe técnico KSL-90-51, Knowledge Systems Laboratory, Universidad de Stanford. Online: ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-90-51.ps.gz

(Gruber y Olsen, 1994) Gruber, T. R.; Olsen, G. R. (1994) An ontology for engineering mathematics. In Jon Doyle, Piero Torasso, & Erik Sandewall, eds., Fourth International Conference on Principles of Knowledge Representation and Reasoning, Gustav Stresemann Institut, Bonn, Germany, Morgan Kaufmann.

(Guarino, 1997) Guarino, N. (1997) Understanding, building and using ontologies. International Journal of Human Computer Studies, 46(2-3), pp. 293 - 310.

(Guarino, 1998) Guarino, N. (1998) Formal Ontology in Information Systems. (Amended version of a paper appeared) in N. Guarino (ed.), Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998. Amsterdam, IOS Press, pp. 3 - 15.

(Guarino et. al., 1997) Guarino, N.; Borgo, S.; Masolo, C. (1997) Logical modelling of product knowledge: towards a well-founded semantics for STEP. In PDTAG-AM and GB Quality Marketing Services QMS, Sandhurst, editors, Proceedings of the European Conference Product Data Technology Days 1997, pp. 183 - 190. Online. Available: http://citeseer.ist.psu.edu/rd/75021831%2C111284%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/297/http:zSzzSzwww.ladseb.pd.cnr.itzSzinforzSzOntologyzSzPaperszSzPDT97.pdf/guarino97logical.pdf

(Guarino y Giaretta, 1995) Guarino, N.; Giaretta, P. (1995) Ontologies and knowledge bases: towards a terminological clarificacion. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, pp. 25 - 32, IOS Press, Amsterdam.

(Guarino y Welty, 2004) Guarino, N.; Welty, C. A. (2004) An overview of OntoClean. In S. Staab, R. Studer (eds.) Handbook on ontologies, Springer, pp. 151 - 171.

(Guizzardi, 2005) Guizzardi, G. (2005) Ontological foundations for structural conceptual models, PhD. Thesis, Centre for Telematics and Information Technology, ISSN 1388-3617; No. 05-74.

(Guzmán et. al., 2006) Guzmán, J. L., Åström, K. J., Dormido, S., Hägglund, T., Piguet, Y. (2006), "Interactive learning modules for PID control" In 7th IFAC Symposium in Advances in Control Education, Madrid, Spain

(Haarslev y Möller, 2003) Haarslev, V.; Möller, R. (2003) Racer: A Core Inference Engine for the Semantic Web. In Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20, pp. 27 – 36.

(Hayes, 1979) Hayes, P. J. (1979) The logic of frames, Frame Conceptions and Text Understanding, Walter de Gruyter and Co. pp 46-61, reprinted in Readings in Knowledge Representation, R. J. Brachman & H. J. Levesque, eds., Morgan Kaufmann, pp. 287 - 295.

(Hayes, 1985) Hayes, P. J. (1985) Naive Physics I: Ontology for liquids, in Hobbs and Moore (eds.) Formal Theories of the Commonsense World, Norwood, NJ; Ablex, pp. 71 - 108.

(Heflin ed., 2004) Heflin, J. (editor) (2004) OWL Web Ontology Language use cases and requirements, W3C Recommendation 10 February 2004. Online: http://www.w3.org/TR/webont-req/

(Hendler y McGuinness, 2000) Hendler, J.; McGuinness, D. L. (2000) The DARPA Agent Markup Language. IEEE Intelligent Systems, 15(6) (Trends and Controversies), pp. 72 - 73.

(Herzog, 2005) Herzog, E. (2005) SysML – an assessment, In Proceedings of the 15th International Council on Systems Engineering INCOSE International Symposium.

(Herzog y Törne, 2001) Herzog, E.; Törne, A. (2001) Information modelling for system specification representation and data exchange. In Proceedings of ECBS 2001, Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 136 - 143.

(Hodgson y Coyne, 2006) Hodgson, R. Coyne, R. (2006) Ontology engineering – the role of ontology architecture in an integrated lifecycle approach. In Proceedings of the Semantic Technology Conference 2006, San Jose, CA, USA, 6-9 de Marzo de 2006, pp. 55 – 60.

(Horrocks, 1998) Horrocks, I. (1998) FaCT and iFaCT. In Proceedings of the International Workshop on Description Logics - DL99 -, pp. 133 - 135.

(Horrocks et. al., 2003) Horrocks, I.; Patel-Schneider, P. F.; van Harmelen, F. (2003) From SHIQ and RDF to OWL: The Making of a Web Ontology Language, Journal of Web Semantics, 1(1), pp. 7 - 26.

(Horrocks et. al., 2004) Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. (2004) SWRL: A Semantic Web rule language combining OWL and RuleML, W3C Member Submission 21 May 2004. Deborah L. McGuinness and Frank van Harmelen eds. Online: http://www.w3.org/TR/owl-features/

(Hu et. al., 2003) Hu, Z.; Kruse, E.; Draws, L. (2003) Intelligent binding in the engineering of automation systems using ontology and Web services. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, 33(3), pp 403 - 412.

(ISO, 1992) International Organization for Standarization (ISO) (1992) Manual de Referencia del Lenguaje EXPRESS, ISO 10303, Industrial Automation Systems and Integration – Part 11.

(ISO, 2007) ISO Project 24707 – Common Logic: A Framework for a Family of Logic-based Languages. ISO/IEC JTC 1/SC 32International Online: http://cl.tamu.edu/docs/cl/ISO_IEC_FDIS_24707__E_.PDF

(Johanson et. al., 1998) Johanson, M., Gäfvert, M., Åström, K. J., (1998) Interactive tools for education in automatic control IEEE Control Systems Magazine, 18(3), pp. 33 - 40.

(Joos y Otter, 1991) Joos, H.-D.; Otter, M. (1991) Control engineering data structures for concurrent engineering. In: Proceedings of the 5th IFAC/IMACS Symposium on Computer Aided Design in Control Systems, Pergamon Press, Oxford, England, pp. 107 - 112.

(Juristo et. al., 1999) Juristo, N.; Moreno, A. M.; Silva, A. (1999) Proposal of a joint graduate program in SE & KE. In Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering. Kaiserslautern, Germany.

(Juristo y Acuña, 2002) Juristo, N.; Acuña, S. T. (2002) Software engineering and knowledge engineering. Expert Systems with Applications, 23(4), pp. 345 - 347.

(Kalfoglou et. al., 2000) Kalfoglou, Y.; Menzies, T.; Althoff, K. D.; Motta, E. (2000) Meta-Knowledge in systems design: panacea ...or undelivered promise? The Knowledge Engineering Review, 15(4), pp. 381 - 404.

(Kattenstroth, 2007) Kattenstroth, H. (2007) Combining description logic and F-logic reasoning. Diploma Thesis, Univ. Gottingen.

(Kifer et. al., 1995) Kifer, M.; Lausen, G.; Wu z., J. (1995) Logical foundations of object oriented and frame based languages, Journal of the ACM, 42(4), pp. 741-843.

(Kitamura, 2006) Kitamura, Y. (2006) Roles of ontologies of engineering artifacts for design knowledge modeling. In Proceedings of the 5th International Seminar and Workshop Engineering Design in Integrated Product Development (EDIProD 2006), 21-23 September 2006, Gronow, Poland, pp. 59 - 69.

(Kitamura y Mizoguchi, 2003) Kitamura, Y.; Mizoguchi, R. (2003) Ontology-based description of functional design knowledge and its use in a functional way server. Expert Systems with Applications, 24(2), pp. 153 - 166.

(Klyne y Carroll eds., 2004) Klyne, G.; Carroll, J. J. (eds.) (2004) Resource Description Framework (RDF): concepts and abstract syntax, W3C Recommendation, 10 February 2004, Online: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ Latest version available at: http://www.w3.org/TR/rdf-concepts/

(Knublauch, 2002) Knublauch, H. (2002) Extreme programming of knowledge-based systems. In Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002), Alghero, Sardinia, Italy.

(Kogut et. al., 2002) Kogut, P.; Cranefield, S.; Hart, L.; Dutra, M.; Baclawski, K.; Kokar, M.; Smith, J. (2002) UML for ontology development, The Knowledge Engineering Review, 17 (1), pp. 61 - 64.

(Kumar y Krogh, 2006) Kumar, R.; Krogh, B.H. (2006) Heterogeneous verification of embedded control systems. In Proceedings of the American Control Conference, Minneapolis, Minnesota, USA, pp. 4597 - 4602.

(Lassila y McGuinness, 2001) Lassila, O.; McGuinness, D.L. (2001) The role of frame-based representation on the Semantic Web. Knowledge Systems Laboratory, Report KSL-01-02. Online: ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-01-02.doc

(Levesque y Brachman, 1985) Levesque, H.; Brachman, R. (1985). A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachmann, R. & Levesque, H., (eds.), Readings in Knowledge Representation, Morgan Kaufmann, pp. 41 - 70.

(Lewen et. al., 2006) Lewen, H.; Supekar, K.; Noy, N. F.; Musen, M. A. (2006) Topic-specific trust and open rating systems: an approach for ontology evaluation. In proceedings of the 4th International EON (Evaluation of Ontologies for the Web) workshop. Online: http://km.aifb.uni-karlsruhe.de/ws/eon2006/eon2006lewenetal.pdf

(Lin y Harding, 2007) Lin, H.K.; Harding, J.A. (2007) A manufacturing system engineering web ontology model on the Semantic Web for inter-enterprise collaboration', Computers in Industry, 58, 2007, pp 428 - 437.

(López et. al., 1999) Lopez, M. F.; Gomez-Perez, A.; Sierra, J. P.; Sierra, A. P. (1999) Building a chemical ontology using Methontology and the Ontology Design Environment, Intelligent Systems and their Applications, 14(1), pp. 37 - 46.

(Lubell et. al., 2004) Lubell J.; Peak R. S.; Srinivasan V.; Waterbury S. (2004) STEP, XML, and UML: complementary technologies. Paper DETC2004-57743, American Society of Mechanical Engineers ASME 2004, Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Salt Lake City.

(Lukibanov, 2005) Lukibanov, O. (2005) Use of ontologies to support design activities at DaimlerChrysler, In Proceedings of the 8th International Protégé Symposium, Madrid, 2005.

(Maciejowski, 2006) Maciejowski, J. (2006) The changing face and role of CACSD. Plenary lecture, IEEE Symposium on Computer-Aided Control Systems Design, 2006.

(Mattson y Elmqvist, 1997) Mattsson, S. E., Elmqvist, H. (1997) Modelica - an international effort to design the next generation modeling language. Proceedings of the 7th IFAC Symposium on Computer Aided Control Systems Design, CACSD'97, Gent, Belgium, April 28-30, 1997.

(Mattson et. al., 1993) Mattson, S. E., Anderson, M. and Åström, K. J. (1993) Object oriented modeling and simulation. In D. A. Linkens, editor. CAD for Control Systems. Marcel Dekker, Inc.

(McCarthy, 1980) McCarthy, J. (1980) Circumscription - a form of non-monotonic reasoning, Stanford Artificial Intelligence Laboratory, Memo AIM-334, Computer Science Department, Report No. STAN-CS-80-788. Online: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/80/788/CS-TR-80-788.pdf

(McDermott, 1982) McDermott, J. P. (1982) R1: A Rule-Based Configurer of Computer Systems. Artificial Intelligence, 19(1), pp. 39 - 88.

(McGregor, 1988) MacGregor, R. F. (1988) A deductive pattern-matcher. In Proceedings of the Association for the Advancement of Artificial Intelligence, AAAI-88, pp. 403 - 408.

(McGuinness, 2001) McGuinness, D. L. (2001) Description logics emerge from ivory towers. Stanford Knowledge Systems Laboratory Technical Report KSL-01-08 2001. In the Proceedings of the International Workshop on Description

Logics. Stanford, CA, August 2001. Online: http://www.ksl.stanford.edu/people/dlm/papers/dls-emerge-final.ps

(McGuinness, 2003) McGuinness, D. L. (2003) Ontologies come of age. In Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press. Online: http://www.ksl.stanford.edu/people/dlm/papers/ontologies-come-of-age-mit-press-(with-citation).htm

(McGuinness et. al., 2002) McGuinness, D. L., Fikes, R.; Hendler, J.; Stein, L. A. (2002) DAML+OIL: an ontology language for the Semantic Web, IEEE Intelligent Systems, 17(5), pp. 72 - 80.

(Michel y Gauthier Associates, 1996) Michel and Gauthier Associates (1996) Advanced Continuous Simulation Language (ACSL), Concord, Massachusetts.

(Niles y Pease, 2001) Niles, I.; Pease, A. (2001) Towards a standard upper ontology. In Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Chris Welty and Barry Smith, eds, Ogunquit, Maine, October 17 - 19.

(Minsky, 1975) Minsky, M. (1975) A framework for representing knowledge. In P. Wisnton, ed., The Psychology of Computer Vision, McGraw-Hill, New York, pp. 211 - 280, reprinted in (1985) Readings in Knowledge Representation, R. J. Brachman & H. J. Levesque, eds., Morgan Kaufmann, pp. 246 - 262.

(Mizoguchi, 2004) Mizoguchi, R. (2004) Ontology engineering environments. In S. Staab, R. Studer (eds.) Handbook on ontologies, Springer, pp. 275 - 295.

(Mizoguchi e Ikeda, 1996) Mizoguchi, R.; Ikeda, M. (1996) Towards ontology engineering. Technical Report AI-TR-96-1, I.S.I.R., Osaka University. Online: http://www.ei.sanken.osaka-u.ac.jp/pub/miz/miz-onteng.pdf

(Morbach et. al., 2007) Morbach, J.; Yang, A.; Marquardt, W. (2007). OntoCAPE – a large-scale ontology for chemical process engineering. Engineering Applications of Artificial Intelligence 20(2), pp. 147 - 161.

(Morgan et. al., 2005) Morgan, A. P.; Cafeo, J. A.; Godden, K.; Lesperance, R. M.; Simon, A. M.; McGuinness, D. L.; Benedict, J. L. (2005) The General Motors' variation-reduction adviser. AI Magazine, 26(2), pp. 269 – 276.

(Motta, 1998) Motta, E. (1998) An overview of the OCML modelling language. In Prooceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML '98).

(Mueller et. al., 2006) Mueller, W.; Rosti, A.; Bocchio, S.; Riccobene, E.; Scandurra, P.; Dehaene, W.; Vanderperren, Y (2006) UML for ESL design - basic principles, tools, and applications, In Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'06, pp. 73 - 80.

(Muha, 1991) Muha, P. A. (1991) Expert system for SROOT. lEE Proceedings. - D, 138(4), pp. 381 - 387.

(Munro, 1990) Munro, N. (1990) ECSTASY – A Control System CAD Environment. In Proceedings of the 11th IFAC World Congress on Automatic Control, Tallinn, Estonia.

(Murray et. al., 2003) Murray, R. M.; Åström, K. J.; Boyd, S. P.; Brockett, R. W.; Stein, G. (2003) Future directions in control in an information-rich world, IEEE Control Systems Magazine, 23(2), pp 20 - 33.

(Musen, 1998) Musen, M. A. (1998). Modern architectures for intelligent systems: reusable ontologies and problem solving methods. American Medical Informatics Association (AMIA) Fall Symposium, Orlando, FL.

(Musen, 1999) Musen, M. A. (1999) Stanford Medical Informatics: uncommon research, common goals, M.D. computing : computers in medical practice, 16(1), pp. 47 - 49.

(Musen, 2004) Musen, M.A. (2004) Ontology-oriented design and programming. In: Cuena, J., Demazeau, Y., Garcia, A., and Treur, J., eds. Knowledge Engineering and Agent Technology. Amsterdam: IOS Press.

(NASA, 2006) NASA (2006) Semantic web for earth and environmental terminology (SWEET). Jet Propulsión Laboratory, California Institute of Technology, 2006. Online: http://sweet.jpl.nasa.gov/

(National Instruments, 2007a) National Instruments (2007), MATRIXx. http://www.ni.com/matrixx/

(National Instruments, 2007b) National Instruments (2007), LabView. http://www.ni.com/labview/

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

(Newell et. al., 1959) Newell, A.; Shaw, J. C.; Simon, H. A. (1959). Report on a general problem-solving program. Proceedings of the International Conference on Information Processing. pp. 256 - 264.

(Noy, 2004) Noy, N. (2004) Semantic integration: a survey of ontology-based approaches. SIGMOD Record, Special Issue on Semantic Integration.

(Noy et. al., 2000) Noy, N.; Fergerson, R.; Musen, M. (2000) The knowledge model of Protege-2000: Combining interoperability and flexibility. In Proceedings of EKAW 2000.

(Noy y McGuinness, 2001) Noy, N.; McGuinness, D. L. (2001) Ontology development 101: a guide to creating your first ontology'. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

(OMG, 1997) Object Management Group (OMG), (1997). Unified Modeling Language: Semantics 1.1 Final Adopted Specification ptc/97-08-04. Online: www.omg.org

(OMG, 2006) OMG Systems Engineering Domain Special Interest Group (2006) SysML Specification v1.0. Online: http://www.omg.org/docs/ad/06-03-01.pdf

(OMG, 2007) Object Management Group: Ontology Programming Special Interest Group (2007) Ontology definition metamodel. Online: www.omg.org/docs/ad/05-08-01.pdf

(Palma et. al., 2000) Palma, J.T.; Paniagua, E.; Martin, F.; Marin, R.; (2000) Ingeniería del conocimiento, de la extracción al modelado de conocimimiento. Revista Iberoamericana de Inteligencia Artificial, 11, pp. 46 - 72.

(Palumbo y Butz, 1991) Palumbo, N. F.; Butz, B. P. (1991) Considerations in the development of a knowledge-based control systems design associate. In Proceedings of the 1988 IEEE International Symposium on Intelligent Control, pp. 263 - 268.

(Pang, 1992) Pang, G. (1992) A Matrix and Expert system Development Aid Language, In Proceedings of the IEEE Symposium on CACSD, Napa, pp. 148 - 155.

(Patel-Schneider, 1984) Patel-Schneider, P. F. (1984) Small can be beautiful .in knowledge representation. In Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, Denver, pp. 11 - 16.

(Patil et. al., 1992) Patil, R.; Fikes, R. F.; Patel-Schneider, P. F.; McKay, D.; Finin, T.; Gruber, T.; Neches, R. (1992) The {DARPA} Knowledge Sharing Effort: Progress Report. In Proceedings of the Third International Conference {KR}'92. Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, San Mateo, California, Bernhard Nebel and Charles Rich and William Swartout", pp. 777 - 788. Online: http://citeseer.ist.psu.edu/rd/75021831%2C568463%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/27156/http:zSzzSzumbc.eduzSz%7EfininzSzpaperszSzkr92-status-report.pdf/patil98darpa.pdf

(Pinto y Martins, 2004) Pinto, H. S.; Martins, J. P. (2004) Ontologies: how can they be built? Knowledge and Information Systems, Springer Verlag, 6(4), pp. 441 - 464.

(Pop y Fritzson, 2004) Pop, A.; Fritzson, P. (2004) The Modelica standard library as an ontology for modeling and simulation of physical systems, Internal Report, August, 2004.

(Price et. al., 2006) Price, D.; West, M.; Christiansen, T.; Kendall, J. (2006) ISO 15926 as OWL - an ontological approach to data warehousing using OWL. In Proceedings of the PDE 2006, 8th NASA-ESA Workshop on Product Data Exchange (PDE).

(Psyché et. al., 2003) Psyché, V., Mizoguchi, R.; Bourdeau, J. (2003) Ontology development at the conceptual level for theory-aware ITS Authoring Systems, In Proceedings of the 2003 AIED Conference, Sydney (2003).

(Pulido et. al., 2006) Pulido, J. R. G.; Ruiz, M. A. G.; Herrera, R. Cabello, E., Legrand, S.; Elliman, D. (2006) Ontology languages for the Semantic Web: a never completely updated review, Knowledge-Based Systems, 19(7), pp.: 489 - 497.

(Quillian, 1967) Quillian, M. R. (1967) Word Concepts: A Theory and Simulation of some Basic Semantic Capabilities, Behavioral Science, 12, pp. 410 - 430.

(Randall et. al., 1993) Randall, D.; Schrobe, H.; Szolovits, P. (1993) What is a knowledge representation? AI Magazine, 14(1), pp. 17 - 33.

(Rector, 2004) Rector, A. (2004) Why use a classifier? When will it help? And when will it not? In 7th International Protégé Conference, 6th - 9th, July 2004, Bethesda, Maryland. Online: http://protege.stanford.edu/conference/2004/abstracts/Rector2.pdf, http://protege.stanford.edu/conference/2004/slides/6.3_rector_Why_classify_Protege_workshop_2004.pdf

(Rector, 2005) Rector, A. (2005) Why and when to use a classifier? In 8th International Protégé Conference, 18th - 21th, July 2005, Madrid, Spain. Online: [http://protege.stanford.edu/conference/2005/submissions/abstracts/accepted-abstract-rector.pdf,http://protege.stanford.edu/conference/2005/slides/5.2_Rector_Why_classify_Protege_workshop_2005-v2.pdf

(Rector ed., 2005) Rector, A. (ed.) (2005) Representing specified values in OWL: "value partitions" and "value sets". W3C Working Group Note. Online: http://www.w3.org/TR/swbp-specified-values/

(Reguera et. al., 2004) Reguera, P.; Fuertes, J. J.; Domínguez, M. (2004) Operating systems resources for web-based training in engineering education. In Proceedings of the IFAC Internet based control education IBCE-04, Grenoble. Laboratoire de automatique, Francia.

(Reiss et. al., 1999) Reiss, M.; Moal, M.; Barnard, Y.; Ramu, J.-Ph.; Froger, A. (2006). Using ontologies to conceptualize the aeronautical domain. In F; Reuzeau, Korker, K. & Boy, G. (eds.). Proceedings of the International Conference on Human-Computer Interaction in Aeronautics, Cépaduès-Editions, Toulouse, France, pp. 56 - 63.

(Reiter, 1977) Reiter, R. (1977) On closed world data bases. In H. Gallaire & J. Minkers, eds., Logic and Data Bases, Plenum Press, New York, pp. 55 - 77.

(Rimvall y Jobling, 1996) Rimvall, C. M.; Jobling, C. P. (1996) Computer-Aided Control Systems Design. In William S. Levine (ed.) The Control Handbook, CRC Press, pp. 429 – 442.

(Rosati, 2006) Rosati, R. (2006) The limits and possibilities of combining description logics and datalog. In Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2006), IEEE Computer Society Press, 2006.

(Rosse y Mejino, 2003) Rosse, C.; Mejino, J. L. V. (2003) A reference ontology for bioinformatics: the Foundational Model of Anatomy. Journal of Biomedical Informatics, 36(6), pp. 478 - 500.

(Sabou et. al., 2006) Sabou, M.; Lopez, V.; Motta, E.; Uren, V. (2006) Ontology selection: ontology evaluation on the real Semantic Web. In proceedings of the 4th International EON (Evaluation of Ontologies for the Web) workshop. Online: http://km.aifb.uni-karlsruhe.de/ws/eon2006/eon2006sabouetal.pdf

(Sanz y Årzen, 2003) Sanz, R.; Årzen, K. E. (2003) Trends in software and control, IEEE Control Systems Magazine, 23(1), pp. 12-15.

(Schevers y Drogemuller, 2005) Schevers, H.; Drogemuller, R. (2005) Converting the Industry Foundation Classes to the Web Ontology Language. In Proceedings of the IEEE 1st International Conference on Semantics, Knowledge, and Grid (SKG 2005), pp. 73 - 75.

(Schlenoff y Uschold, 2004) Schlenoff, C.; Uschold, M. (2004) Knwoledge engineering and ontologies for autonomous systems, 2004 AAAI Symposium. Robotics and Autonomous Systems 49(1-2), pp. 1 - 5.

(Shadbolt et. al., 2004) Shadbolt, N.; O'Hara, K.; Cottam, H. (2004) The use of ontologies for knowledge acquisition. In J. Cuena et. al. (eds.) Knowledge engineering and agent technology, IOS Press, pp. 19 - 42.

(Shank, 1975) Schank, R. C. (1975) Conceptual information processing. Elsevier. Amsterdam

(Shank y Abelson, 1977) Schank, R. C.; R. P. Abelson (1977) Scripts, plans, goals and understanding. New York: Lawrence Erlbaum Associates.

(Sirin et. al., 2007) Sirin, E.; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A.; Katz, Y. (2007) Pellet: a practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), pp. 51-53

(Smith, 2004) Smith, R. (2004) Aristotle's logic. Stanford Encyclopedia of Philosophy. Online: http://plato.stanford.edu/entries/aristotle-logic/

(Sowa, 1991) Sowa, J. F. (1991) Principles of semantic networks: explorations in the representation of knowledge, Morgan Kaufmann Publishers.

(Sowa, 2000) Sowa, J. F. (2000) Knowledge representation: logical, philosophical, and computational foundations", Ed. Brooks/Cole.

(Speel, 1995) Speel P.-H. (1995) Selecting knowledge representation systems. PhD thesis, Universidad de Twente, Enschede, the Netherlands.

(Sure et. al., 2004) Sure, Y.; Gómez-Pérez, A.; Daelemans, W.; Reinberger, M-L.; Guarino, N.; Fridman Noy, N.(2004) Why evaluate ontology technologies? Because it works!, IEEE Intelligent Systems, 19(4), pp. 74 - 81.

(Taylor y Frederick, 1984) Taylor, J. H.; Frederick, D. K., (1984). An expert system architecture for computer-aided control engineering. In Proceedings of the IEEE, 72, December 1984.

(The Bussiness Rules Group, 2000) The Business Rules Group (2000) Defining business rules ~ what are they really? (4th ed. July 2000). Online: http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf

(The Mathworks, 2007) The Mathworks (2007) MATLAB®, the language of technical computing. http://www.mathworks.com/products/matlab/

(Tobies, 2001) Tobies, S. (2001) Complexity results and practical algorithms for logics in knowledge representation, PhD. Thesis. Online: http://citeseer.ist.psu.edu/rd/0%2C441829%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/22105/ftp:zSzzSzwww-lti.informatik.rwth-aachen.dezSzpubzSzpaperszSz2001zSzTobies-PhD-2001.pdf/tobies01complexity.pdf

(Top y Akkermans, 1994) Top, J.; Akkermans, H. (1994) Tasks and ontologies in engineering modelling. International Journal of Human-Computer Studies, 41(4), pp. 585 - 617.

(Trapp, 1993) Trapp, G. (1993) The emerging STEP standard for product-model data exchange, Computer, 26(2), pp. 85 - 87.

(Tu, 2001) Tu, S. (2001) A Tutorial on PAL (Protégé Axiom Language), Stanford Medical Informatics, Stanford University, 2001 Fifth International Protégé User Group Meeting, Newcastle upon Tyne. Online: http://protege.stanford.edu/plugins/paltabs/pal-quickguide/PALTutorial.pdf

(Uschold, 2003) Uschold, M. (2003) Where are the Semantics in the Semantic Web? AI-Magazine. 24(3), pp. 25 - 36.

(Uschold, 2006) Uschold, M. (2006) Ontologies ontologies everywhere, but who knows what to think? 9th International Protégé Conference, 23-26 de Julio, Stanford, California, US.

(Uschold y Grüninger, 1996) Uschold, M.; Grüninger, M. (1996) Ontologies: principles, methods, and applications, Knowledge Engineering Review, 11(2), pp. 93 - 155.

(Uschold y Welty, 2005) Uschold, M.; Welty, C. - W3C Working Group - (2005) Representing Classes As Property Values on the Semantic Web. Online: http://www.w3.org/TR/swbp-classes-as-values/

(Valente et. al., 1999) Valente, A.; Russ, T.; MacGregor, R.; Swartout, W. (1999) Building and (re)using an ontology of air campaign planning. IEEE Intelligent Systems and Their Applications, [ver también IEEE Intelligent Systems], 14(1), pp. 27 - 36.

(van der Vet et. al., 1994) Van der Vet P.E., Speel P.-H., Mars N. J. I. (1994) The Plinius ontology of ceramic materials. Proceedings of ECAI (European Conference on Artificial Intelligence) 94's Workshop on Comparison of Implemented Ontologies, Amsterdam.

(van der Vet et. al., 1995) van der Vet P.E.; Mars N. J. I. (1995) Bottom-up construction of ontologies: the case of an ontology of pure substances. Memoranda Informatica 95-35, Univ. of Twente, Enschede, the Netherlands. Online:
http://citeseer.ist.psu.edu/rd/75021831%2C88343%2C1%2C0.25%2CDownload/http://coblitz.codeen.org:3125/citeseer.ist.psu.edu/cache/papers/cs/3214/http:zSzzSzwww.cm.cf.ac.ukzSzUserzSzJ-C.PazzagliazSzReferenceszSztr:vandervet95.pdf/vandervet95bottomup.pdf

(van der Vet y Mars, 1998) van der Vet, P. E.; Mars, N. J. I. (1998) Bottom-up construction of ontologies. IEEE Transactions on Knowledge and Data Engineering, 10(4). pp. 513 - 526.

(Vanderperren y Dehaene, 2006) Vanderperren, Y.; Dehaene. W. (2006) From UML/SysML to Matlab/Simulink: current state and future perspectives. In Proceedings of Design, Automation and Test in Europe, DATE'06, Munich.

(Varsamidis et. al., 1994) Varsamidis, T.; Hope, S.; Jobling, C.P. (1994) Information management for control system designers. In Proceedings of the IEE International Conference on Control (Control '94), 1, pp 13 – 17.

(Varsamidis et. al., 1996) Varsamidis, T.; Hope, S.; Jobling, C.P. (1996) Integration using a unified model for CACSD. IEE Colloquium on Advances in Computer-Aided Control System Design (Digest No: 1996/061), pp. 2/1 - 2/4.

(Varsamidis et. al., 1999) Varsamidis, T.; Hope, S.; Jobling, C. P. (1999) Use of a prototype CACE integration framework based on the unified information model. In Proceedings of the International Symposium on Computer-Aided Control System Design, Hawaii (USA), pp. 392 - 397.

(von Luck et. al., 1987) von Luck, K.; Nebel, B.; Peltason, C.; Schmiedel, A. (1987) The anatomy of the BACK system. KIT-Report 41, Technise Universitat Berlin.

(Vrandecic, 2005) Vrandecic, D. (2005) Explicit knowledge engineering patterns with macros. In Chris Welty and Aldo Gangemi, Proceedings of the Ontology Patterns for the Semantic Web Workshop at the ISWC (International Semantic Web Conference) Galway, Ireland, November 2005.

(W3C, 2004) W3C consortium (2004) OWL Web Ontology Language overview, W3C Recommendation 10 February 2004. Deborah L. McGuinness and Frank van Harmelen eds. Online: http://www.w3.org/TR/owl-features/

(Wagner, 2002) Wagner, G. (2002) How to design a general rule markup language, Proceedings of. the Workshop XML Technologies for the Semantic Web (XSW 2002), HU Berlin.

(Wang et. al., 2006) Wang, H.; Rector, A.; Drummond, N.; Horridge, M.; Seidenberg, J.; Noy, N.; Musen, M.; Redmond, T.; Rubin, D.; Tu, S.; Tudorache, T. (2006) Frames and OWL Side by Side, In 9th International Protégé Conference, Stanford, CA, USA Online: http://protege.stanford.edu/conference/2006/submissions/slides/7.2wang_protege2006.pdf, http://protege.stanford.edu/conference/2006/submissions/abstracts/7.2_Wang_Hai_Protege_conf.pdf. Versión extendida disponible en: http://smi-protege.stanford.edu/svn/*checkout*/frames-vs-owl/FrameOWLSidebySide_Stanford_v3.pdf?rev=2583

(Waterbury, 2007) Waterbury, S. (2007) The Pan Galactic status report (an update on the Pan Galactic Engineering Framework [PGEF]).in Proceedings of the 9th NASA-ESA Workshop on Product Data Exchange (PDE), Santa Barbara, CA, USA.

(West, 1992) West, J. C. (1992) Review of: "Control theory - a guided tour", IEE Review, Septiembre de 1992, pp. 318

(Wielinga y Schreiber, 1993) Wielinga, B.J.; Schreiber, A. T. (1993) Reusable and sharable knowledge bases: a european perspective. In Proceedings of First International Conference on Building and Sharing of Very Large-Scaled Knowledge Bases. Tokyo, Japón.

(Willard, 2007) Willard, B. (2007) UML for systems engineering, Computer Standards & Interfaces, Vol. 29(1), pp. 69 - 81.

(Winograd, 1972) Winograd, T. (1972) Understanding Natural Language, Academic Press.

(Woods, 1975) Woods, W. A. (1975) What's in a Link?: Foundations for Semantic Networks, in D.G. Bobrow & A. Collins (eds.), Representation and Understanding, Academic Press; reprinted in, Collins & Smith eds., Readings in Cognitive Science, section 2.2, reprinted in Readings in Knowledge Representation, R. J. Brachman & H. J. Levesque, eds., Morgan Kaufmann, pp. 218 - 241.

Las frases que nunca escribiré, los paisajes que no podré nunca describir, con qué claridad los dicto a mi inercia y los describo en mi meditación cuando, recostado, no pertenezco sino de lejos a la vida. Esculpo frases enteras, perfectas palabra por palabra, tramas de dramas se me narran construidas en el espíritu, siento el movimiento métrico y verbal de grandes poemas con todas sus palabras, y un gran entusiasmo, como un esclavo al que no veo, me sigue en la penumbra. Pero si diera un paso desde la silla donde sepulto estas sensaciones casi perfectas hasta la mesa donde me gustaría escribirlas, las palabras huyen, los dramas mueren, del nexo vital que unió el murmullo rítmico no queda más que una saudade ajena, unos restos de sol sobre montes remotos, un viento que levanta las hojas junto al umbral desierto…

Fernando Pessoa - Libro del desasosiego