Departamento de Automática, Ingeniería Electrónica e Informática Industrial

E.T.S. Ingenieros Industriales

UNIVERSIDAD POLITECNICA DE MADRID

OASys: Ontology for Autonomous Systems

PhD Dissertation

Ing., MSc., Julita Bermejo Alonso

Thesis Advisor: Dr. Ing. Ricardo Sanz Bravo

Thesis Advisor: Dr. Ing. Manuel Rodríguez Hernández

2010

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día 23 de Julio de 2010.

Presidente: Dr. Dña. Anisoara Calinescu
Vocal: Dr. D. Rodolfo Haber Guerra
Vocal: Dra. Dña. Miren Idoia Alarcón Rodríguez
Vocal: Dr. D. Manuel González Bedia
Secretario: Dr. D. Ignacio López Paniagua
Suplente: Dr. D. Juan Ramón Velasco Pérez
Suplente: Dr. D. Miguel Hernando Gutiérrez

Realizado el acto de defensa y lectura de la Tesis el día 17 de Septiembre de 2010.

En la E.T.S. de Ingenieros Industriales.

CALIFICACIÓN: Sobresaliente Cum Laude.

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

Contents

List of Figures

List of Tables

To the best autonomous systems I will ever design: my sons Alberto and Diego

Acknowledgements

My journey

This dissertation you are holding in your hands it is the result of a long quest that started some 26 years ago. I will explain it. When I was fourteen, I had an ephiphany. I wanted to be an engineer, a researcher, and a lecturer. An engineer to acquire the knowledge about how things work, behave and are in nature and artificial systems. A researcher to continue the search for knowledge in new fields. Finally, a lecturer to share this knowledge with others, challenge them and create new engineers.

The first one, I became after my college studies. Being an engineer has influenced, ever since, every decision, feeling, and thought I have had. Whether I like it or not.

For the other two, I started long ago. Law and society require to fulfill a PhD research and dissertation, at least to be the third one. The second one, the researcher, is intertwined with it.

However, dreams are one thing and real life quite another. What should have been a rather short journey, became the most exciting experience in my life, which has taken me from country to country, and university to university, through ups and downs.

Some five years ago, I happened to meet someone that changed the course of my personal history. Someone crazy enough, or may I say bright enough, to envision intelligent and autonomous systems to aid in our everyday life (or at least, to ease our control systems domain). He just opened a door for me.

Once that door was opened, I became part of that dream. My journey has driven me through cognitive science, human mind, knowledge, autonomous systems, and finally, ontologies (the main topic of my current research). My own mind expanded and grew with the new knowledge, whilst opening opportunities for new own.

I do not expect the journey to finish after this. Once you open a door in your mind, it can never be closed.

Whom I am indebted with

I do not want this section to be a never ending list of people in my life. Colleagues and friends throughout this years have contributed, each one in his/her own way, to what I am, I think, and I have reflected in this research. All of them know who they are.

However, I would like to express my gratitude to the people related to the cornerstones of my life. As I describe myself, I am a full–time mother and spouse, with a strong engineering background. But I am also a part–time researcher.

Combining both roles had not always been easy. Children and research do not always get along, especially when your children are ill and you have not got a wink of sleep throughout the night!!. However, both my husband and my children reminded me the importance of being who I am and how I am. Not everything in life is working. One should be something else.

I have contributed to make life more rational for my husband: Nacho. He has always supported what I wanted to make with my life, throughout all the journey we have travelled together. He has also made it more complicated, but perhaps I will not be who I am without his challenges. I hope he is willing to continue a lifetime journey with me.

Moreover, I want to believe that, along with this dissertation, I have achieved two personal masterpieces: Alberto and Diego, my sons. I am not sure if they fully grasp what their mother has been doing all these years, when I told them I was working on my dissertation. I expect to have opened the doors on knowledge and science to them for their minds to be challenged. It is up to them to follow this league. I do wish so.

Regarding the researcher role, I will forever be grateful to a colleague, and also friend. My supervisor, Ricardo. He was the one who opened the door, and who believes enough in me to keep it open despite my blues and my wandering lifestyle. His role is not constricted to a mere supervisor. He stood up for me through good and bad (heaven and hell?). He believed in me more than I will ever do. We have shared talks, discussions, and brainstorms together. We know our dream for systems to be fully autonomous is not totally finished, but we have the rest of our lives to fulfill it. His companionship has been crucial for you to hold this dissertation.

Written on a sunny day in Houston, 13th February 2007

Con el tiempo, el apoyo de dos personas se convirtió en fundamental. Manuel Rodríguez, inicialmente mi codirector en la sombra. Digo en la sombra porque al principio estaba presente, pero no oficialmente. El papel que ha jugado Manuel es el de siempre estar ahí, algo importanto cuando se esta a millas o kilómetros del grupo de investigación. Siempre ha estado al otro lado de la línea cuando he enviado mis avances, textos o artículos. Su revisión detallada de cuanto iba haciendo me ha ayudado sobremanera a seguir en la brecha, reconociendo al mismo tiempo mis carencias y fallos. No sé muy bien que hubiera sido de toda esta investigación durante el último período si él no hubiera estado ahí, apoyándome cuando más hacía falta. Gracias Manolo.

Otra persona importante en esta historia es Carlos, mi aliado técnico. Sin él no hubiera sido posible solventar los mil y un problemas técnicos que han surgido en estos ultimos años. El acceso a los documentos, los servidores y demás tecnologías ha sido su aportación a esta tesis. Consiguió que estuviera presente en el grupo de investigación, cuando físicamente esto no era posible. Su ayuda en la revisión también aclaró puntos e ideas que yo no había considerado previamente. Gracias por tu ayuda. Espero devolvértela cuando tú hagas tu propia tesis (algo no muy lejano).

Escrito en un día gris y frío en Paris, 8 de Enero de 2010

Versión Española

RESUMEN

El marco de esta investigación es un proyecto a largo plazo sobre sistemas autónomos, el proyecto ASys, que aborda el desarrollo de una tecnología universal para toda clase de sistemas autónomos, independientemente de su dominio de aplicación en particular. Por sistemas autónomo se entiende un sistema capaz de operar en un entorno real sin ningún tipo de control externo durante largos periodos de tiempo. La estrategia consiste en explotar los bucles de control cognitivo, utilizando el conocimiento capturado como modelos diferentes, basados en la ontología para sistemas autónomos (OASys) desarrollada en esta investigación.

Una ontología es una especificación formal, explícita de una conceptualización compartida, es decir, un modelo abstracto legible por máquina donde los conceptos y las relaciones se identifican y definen explícitamente por consenso en un grupo. Las ontologías se han utilizado ampliamente para la ingeniería de software. También se han usado como mecanismos de representación basados en un lenguaje computacional, para aclarar y compartir el conocimiento del dominio.

OASys captura y utiliza los conceptos como soporte software para la descripción y el proceso de ingeniería de cualquier sistema autónomo, como una ontología de dominio, estructurada en dos niveles de abstracción, interiormente organizados en Subontologías y Paquetes. METHONTOLOGY es la metodología utilizada como método de diseño, con una implementación final en UML.

La descripción del sistema autónomo ha sido formalizada en la Ontología de ASys, dividida en la Subontología de Sistemas y la Subontología de Sistemas Autónomos.

La primera subontología contiene los elementos para definir cualquier sistema: el Paquete General de Sistemas para caracterizar la estructura y el comportamiento de cualquier sistema que se basa en la teoría general de sistemas, el Paquete de Mereología con las taxonomías de la parte y el todo, y el Paquete de Topología para conexiones topológicas.

La segunda especializa los anteriores conceptos para sistemas autónomos: el Paquete de Percepción para describir el proceso de percepción; el Paquete de Conocimiento para caracterizar los diferentes tipos de conocimiento utilizados por el sistema autónomo; el Paquete de Pensamiento con conceptos de procesos orientados al objetivo y las tareas; el Paquete de Acción acerca de las operaciones y los actores; y el Paquete de Equipos para

detallar los dispositivos.

El proceso de ingeniería de sistemas autónomos ha sido formalizado en la Ontología de Ingeniería de ASys, estructurada en la Subontología de Ingeniería de Sistemas y la Subontología de Ingeniería de ASys.

La primera subontología reúne los conceptos relacionados con un proceso de ingeniería, basado en especificaciones y metamodelos diferentes: el Paquete de Requisitos para especificar los requisitos del sistema; el Paquete de Perspectivas con los diferentes puntos de vista en el desarrollo de un sistema; el Paquete de Proceso de Ingeniería para describir un proceso de desarrollo de ingeniería; y el Paquete de Ingeniería Basada en Modelos.

La segunda subontología contiene la especialización y los elementos adicionales para describir un proceso genérico de ingeniería de un sistema autónomo: el Paquete de Requisitos de ASys para caracterizar los requisitos de calidad de proceso y del sistema, el Paquete de Perspectivas de ASys para diseñar un sistema autónomo desde diferentes aspectos; el Paquete de Proceso de Ingeniería de ASys para describir un proceso genérico de ingeniería de un sistema autónomo.

OASys ha sido complementada con el desarrollo de una Metodología basada en OASys para ejemplificar su uso en un proceso de ingeniería genérico de un sistema autónomo.

OASys se ha aplicado en el sistema de control de robot (RCT en inglés) y el sistema de control de procesos (en inglés PCT). RCT es un sistema robot móvil, con una amplia gama de implementaciones y capacidades. PCT consiste en el desarrollo de una arquitectura de control robusto para un sistema de reacción química, dotando al sistema con capacidades cognitivas para llevar a cabo tareas complejas, como diagnóstico de fallos, gestión de alarmas y control de la reconfiguración del sistema desde un punto de vista teórico único.

OASys ofrece mejoras en el intercambio de conocimientos y la reutilización entre los desarrolladores. La separación entre la descripción y el proceso de ingeniería define de forma independiente la caracterización y el proceso de ingeniería de los sistemas de prueba. Permite la conceptualización común de diferentes sistemas, identificando elementos comunes de una manera independiente del dominio. Los diferentes niveles de abstracción describen los sistemas ejemplo desde un punto de vista de focalización de dominio, aunque el proceso de refinamiento y el uso de los elementos ontológicos tiene que desarrollarse aún más.

Capítulo 1

SISTEMAS

1.1 Teoría General de Sistemas

Esta tesis se ha basado en parte en la investigación llevada a cabo por George J. Klir, quien considera la ciencia de sistemas como un disciplina interdisciplinar por encima de la existencia de diferentes dominios. De acuerdo con Klir, la ciencia de sistemas se compone de: un dominio de investigación, el conocimiento sobre dicho dominio, y una metodología para la obtención de nuevo conocimiento. Los primeros intentos de describir un entorno general para la descripción y clasificación de sistemas fueron descritos en [Kli69]. Klir propone que " cualquier entidad debe satisfacer dos requisitos para ser un sistema: estar formada por un conjunto de cosas de cierta clase, y que estas cosas estén relacionadas de forma reconocible" [Kli85].

En el campo de la ingeniería, un sistema es una entidad cuya interacción con el entorno, así como su operación interna, puede conocerse analizando ciertas propiedades de sus elements, cómo estos elementos se comportan y cómo el sistema interactúa con el entorno [UIT06d]. Señalar que en esta definición hay dos elementos a considerar: que la frontera entre un sistema y su entorno no siempre se puede especificar claramente; y que un sistema puede ser considerado como tal o como un elemento de otro dependiendo del contexto.

Un sistema se define seleccionando un conjunto de propiedades o atributos. El entorno de un sistema es otro sistema, cuyos elementos no son parte del sistema en estudio. A cada propiedad se le asigna una cantidad o variable. Para distinguir entre diferentes medidas de la misma propiedad, se utiliza el tiempo, el espacio, la población o una combinación de ellos como referencia. La frecuencia y la precisión de las observaciones se determina mediante el nivel de resolución en la medida. La variación de los valores de todas las cantidades o variables en un período de tiempo se denomina actividad del sistema. Estas medidas se traducen en un relación invariante en el tiempo.

El objetivo es explicar la actividad del sistema, de forma que las cantidades del sistema satisfagan relaciones invariantes en el tiempo, unas en función de otras. El conjunto de todas estas relaciones es lo que se define como comportamiento del sistema. Si el sistema tiene un cierto comportamiento, entonces posee ciertos atributos o propiedades que lo justifican. Estas propiedades se denominan organización del sistema. Para reflejar que si el comportamiento cambia, la organización también, la organización se divide en un parte constante llamada estructura y un parte variable llamada programa.

En cuanto a la estructura física, los sistemas se componen de subsistemas que a su vez pueden descomponerse en elementos. El comportamiento del sistema se ve influído por el de sus subsistemas o elementos. El comportamiento de dos elementos se combina siempre y cuando haya ciertas cantidades o variables comunes entre los dos. El conjunto de todas las cantidades externas comunes se denomina acoplamiento. El conjunto de todos los elementos y sus acoplamientos recibe el nombre de estructura del universo de discurso y los acoplamientos (abreviada como estructura UC).

Para completar la descripción, se analiza la evolución de los valores de las cantidades del sistema. El estado del sistema se define como el conjunto de los valores de la cantidades (externas e internas) de un sistema. En el instante siguiente, el sistema habrá evolucionado a un nuevo estado. Esta evolución se llama transición, es decir, los cambios válidos de los valores de las cantidades determinados por la combinación de elementos y sus acoplamientos, influído por el entorno. Dado un estado, no toda evolución es posible, sino dentro de un conjunto de posibles transiciones. El conjunto de estados y de transiciones se denomina la estructura de estados–transiciones (abreviada como estructura ST).

Un sistema se define mediante algunas de las características antes definidas, pero sólo aquéllas constantes y totalmente conocidas pueden definir el sistema de forma unívoca. Dichas características se denominan primarias, y son:

1. El conjunto de cantidades externas junto con la resolución espacio–tiempo.
2. Una actividad dada.
3. El comportamiento permanente.
4. La estructura UC real.
5. La estructura ST real.

1.2 Sistemas Autónomos

1.2.1 Autonomía y Sistemas Artificiales

El concepto de autonomía se refiere a la capacidad de un sistema natural de autogobernarse. Para sistemas artificiales, autonomía se refiere a la capacidad de operar en un entorno real sin control o supervisión externa durante largos períodos de tiempo [Bek05]. El enfásis se hace en aspectos tales como: ausencia total o parcial de operadores humanos, dependencia mínima con el entorno, y cohesión en el sistema [UIT06d]. La autonomía se obtiene como combinación de diferentes capacidades tales como la inteligencia, el conocimiento, o la capacidad de respuesta. Los sistemas artificiales se enfocan hacia una otra capacidad [UIT06c], aunque en general se busca no una autonomía completa (difícil desde el punto de vista técnico y económico) sino una autonomía limitada en función de los objetivos impuestos en el diseño [SR08].

Hay diferentes razones que justifican el desarrollo de sistemas autónomos. En primer lugar, los motivos económicos ya que los sistemas autónomos son económicamente más rentables por su alto rendimiento, la reducción de personal humano, mejores condiciones de operación, etc. En segundo lugar, los motivos técnicos como una mayor producción, mejores condiciones de trabajo, etc [SR08].

No obstante, los sistemas reales tienen que hacer frente a condiciones operativas que limitan su autonomía [UIT06d], [DTC08]:

- Perturbaciones externas, que pueden hacer que el sistema diverja del estado deseado. Normalmente no se consideran en el diseño del sistema, dado su origen desconocido y aleatorio. No obstante, se suelen modelar, por ejemplo estadísticamente, para incluir en el sistema los mecanismos necesarios para tratarlas.
- Mayores niveles de abstracción de las tareas que el sistema ha de realizar.
- Ciertas dinámicas no consideradas que llevan al sistema a un estado no deseado, y no predecido.

Por tanto, los sistemas artificiales se basan en su capacidad de adaptación para conseguir dicha autonomía, de forma que puedan adaptar su estructura y operación para compensar las perturbaciones e incertidumbres del entorno, al mismo tiempo que se asegura la convergencia a los objetivos inicialmente definidos. Esta capacidad de adaptación se obtiene dotando al sistema de ciertas capacidades [SLBH06]:

1. Capacidades cognitivas basadas en el conocimiento, el aprendizaje y la inteligencia.
2. Modularidad mediante el diseño basado en módulos que pueden ser intercambiados o reconfigurados según se necesite.
3. Técnicas de tolerancia a fallos que dotan al sistema de la capacidad de reacción cuando un fallo tiene lugar.
4. Técnicas de inteligencia artificial que permiten al sistema operar con elementos imprecisos o abstractos.

1.2.2 Definición de un Sistema Autónomo

Hay diferentes ideas sobre en qué consiste un sistema autónomo. Inicialmente, este término se asoció a la capacidad de movimiento en un entorno, y por tanto la investigación se centró en el diseño y definición de robots autónomos [MA02], [BCKS04], [CTSB04], [WVH04], [Bek05], [Hua04]. Otro enfoque se centró en el concepto de agente autónomo inteligente, definido como “un sistema situado dentro de un entorno que lo percibe y actúa sobre él, en función de sus objetivos para influir lo que suceda en el futuro.” [FG96]. Recientemente, el concepto de sistema autónomo se ha ampliado para considerar sistemas más complejos, cuyo comportamiento es combinación de múltiples subsistemas que trabajan en coordinación. Nuevos desarrollos basados en computación se centran en este nuevo enfoque [KC03], [JL04], [Cal09]. Este tipo de sistemas se caracterizan por alguna de las siguientes habilidades [IBM01], [DTC08], [HM07]:

- autoreoganización, cambiando su estructura según se necesite,
- autoconfiguración de acuerdo a sus objetivos,
- reasignación de tareas en función de nuevos recursos,
- autooptimizacion que mejora su rendimiento,
- autoadaptación que le permite reaccionar de forma más rápida y eficiente,
- autoreparación que le permite detectar y corregir fallos,
- autoprotección para anticipar y prevenir fallos,
- capacidad de decisión para seleccionar entre un conjunto de alternativas,
- aprendizaje de nuevos entornos o situaciones, y
- percepción para utilizar y optimizar el uso de sensores y actuadores.

Capítulo 2

ONTOLOGIAS

Este capítulo resume diferentes aspectos relacionados con el tema de las ontologías. Se presentan diferentes definiciones, se resumen los tipos existentes, las aplicaciones, y las técnicas y métodos existentes para el desarrollo de ontologías.

2.1 Definición de Ontología

Hay múltiples definiciones de ontología en textos científicos, en función del dominio en el que se habla. En Ingeniería del Conocimiento, generalmente se entiende como aquello que puede representarse. Una de las definiciones más conocidas es la de [Gru93a], [Gru93b]:

Una ontología es una especificación explícita de una conceptualización.

Una ontología contiene clases, relaciones, atributos, instancias, funciones y axiomas. Las clases representan las entidades en el dominio. Las relaciones establecen nexos entre dichas entidades. Las instancias son los objetos en el dominio. Las funciones y axiomas dirigen y limitan el uso de los elementos anteriores.

Según [Gua98], una ontología es "una teoría lógica respecto a la intención del significado de un vocabulario formal, es decir, el compromiso ontológico a una conceptualización particular del mundo".

Una ontología también se ha definido por Uschold [UG96] como "una descripción explícita de un conocimiento compartido en un area concreta". Una definición muy precisa ha sido dada por [SBF98]: "Una ontología es una especificación formal y explícita de una conceptualización compartida" en la que conceptualización se refiere a un modelo abstracto de cierto fenómeno en el entorno; explícita indica que los elementos ontológicos se han definido explícitamente; formal que debe ser leída por una máquina, y compartida que la ontología captura el conocimiento común de un grupo.

Otras definiciones inciden en que una ontología es un elemento para ingeniería que consta de un vocabulario específico y un significado concreto para dicho vocabulario [JU99], [Gua98]. No obstante, una ontología no es sólo un vocabulario en un dominio, sino sobre todo el contenido sobre los objetos, propiedas y relaciones entre ellos [CJB99].

2.2 Clases de Ontologías

Existen diferentes clasificaciones sobre los tipos posibles de ontologías, en función del aspecto que se tiene en consideración. Una primera clasificación hace referencia a la clase de lenguaje utilizado [UG96]: altamente informal (en lenguaje natural), semiinformal (en lenguaje natural estructurado), semiformal (en lenguaje artificial estructurado), y formal (con semántica formal, teoremas y pruebas).

Si la clasificación se centra en la clase de conceptualización utilizada, las ontologías se pueden clasificar en [Gua97], [Gua98]: ontología de alto nivel (describe conceptos generales como tiempo, espacio, etc), ontología de dominio (descripción de un dominio genérico), ontología de tarea (descripción de una tarea), ontología de aplicación (conceptos de un dominio para una tarea concreta).

2.3 Criterios de Diseño para Ontologías

Existen unos criterios definidos por [Gru93b], que han de considerarse cuando se desarrolla una ontología:

Claridad: Las ontologías deben ser lo más objetivas posibles para permitir la comunicación y la compartición de los conceptos.

Coherencia: Es preferible la utilización de axiomas formales frente al uso de definiciones en lenguaje natural.

Posibilidad de ampliaciones: El diseño de la ontología debe prever ampliaciones y cambios de forma que las definiciones ya existentes no se vean modificadas.

Minimizar el efecto del lenguaje utilizado: La conceptualización debe hacerse sin tener en cuenta el lenguaje de implementación, sino el conocimiento a conceptualizar.

Minimizar los compromisos ontológicos: Las ontologías se crean para ser compartidas, por tanto los compromisos en significado deben minimizarse para permitir la instanciación al nivel que se necesite.

No todos los criterios pueden respetarse cuando se diseña una ontología. Se necesitan ciertos acuerdos entre cuales se eligen [Gru93b] y un compromiso entre el diseño y la aplicación de la ontología [BAT97].

2.4 Técnicas y Metodologías

Existen diferentes lenguajes, técnicas y metodologías que se han desarrollado para el diseño y la implementación de ontologías. Los enfoques se centran en aquellas relativas a técnicas de inteligencia artificial, frente a las que se basan en técnicas similares a las utilizadas en el desarrollo de software. Una excelente revisión de las técnicas basadas en inteligencia artificial se encuentra en [GPFLC04b].

2.5 El Uso de Ontologías

Las ontologías se diseñan con el objetivo de compartir y facilitar el conocimiento. Papeles tradicionales son [VB96], [JU99], [CJB99], [NM01], [WVV$^+$01], [VS02], [Kit06]:

1. Ontología como especificación: una ontología se desarrolla en un dominio como soporte al software.
2. Compartir el conocimiento: la ontología se utiliza para compartir un vocabulario entre usuarios (humanos o máquinas).
3. Búsqueda basada en ontologías: la ontología se utiliza para buscar información en una librería.
4. Adquisición de conocimiento: la ontología se utiliza para comprender un dominio, y su terminología.
5. Clarificar la estructura del conocimiento: un análisis ontólogico de un dominio permite definir con claridad el vocabulario y la conceptualización de dicho dominio.
6. Reutilización del conocimiento: la ontología facilita el reutilizar el mismo conocimiento en diferentes aplicaciones.

Existen diferentes dominios en los que se han desarrollado ontologías tale como la medicina, el comercio electrónico, la red semántica, el modelado de empresas, la gestión del conocimiento, etc. Excelentes revisiones de diferentes aplicaciones se pueden encontrar en [KMAM00], [GPFLC04b], [SSSS01].

2.6 Ontologías para Ingeniería Software

Las ontologías se utilizan ampliamente en el desarrollo de aplicaciones software. Por una parte, las ontologías se han utiliado como ontologías de dominio con la descripción del proceso o tecnologías empleados. Por otra parte, las ontologías también se han utilizado como elementos software de soporte.

Una taxonomía de ingenierías en el dominio de la ingeniería software ha sido propuesta por Ruiz en [RH06]. Se consideran dos categorías:

1. *Ontologías de Dominio* para representar el conocimiento de un dominio concreto dentro de la ingeniería software.
   - Ingeniería Software
     - Genérica
     - Específica: requisitos, diseño, mantenimiento, configuración, calidad, métodos y herramientas, procesos y gestión de ingeniería.
   - Tecnología Software
     - Software: técnicas de programación, lenguajes, sistemas operativos.
     - Datos: estructura, almacenamiento, codificación, etc.
     - Sistemas de información: modelos, bases de datos, almacenamiento y recuperación de información, representación.
2. *Ontologías como Elementos del Software*: las ontologías se utilizan como elemento de apoyo al proceso software.
   - Durante el desarrollo de aplicaciones
     - Para procesos de ingeniería: las ontologías se utilizan en el desarrollo y mantenimiento.
     - Para otros procesos: las ontologías se utilizan en el soporte y gestión.
   - Durante la ejecución de aplicaciones
     - Como parte de la arquitectura.
     - Como recurso y soporte a la información.

La Tabla 2.1 y la Tabla 2.2 resumen, respectivamente, la amplia bibliografía analizada para las dos categorías consideradas: ontologías de dominio y las ontologías como elementos del software, a excepción de las ontologías para Tecnología Software, ya que se centra en la descripción de técnicas y lenguajes, y por tanto se ha considerado que no tiene especial relevancia para la investigación de esta tesis.

2.7 Ontologías para Sistemas Autónomos

Las ontologías se han utilizado para sistemas autónomos como representación del conocimiento basado en una conceptualización, definiendo los conceptos con un determinado lenguaje para describir las diferentes entidades que forman parte del diseño y operación de un sistema autónomo: el entorno, los objetos con los que interacciona, acciones, recursos, etc. La Tabla 2.3 recoge los diferentes ejemplos seleccionados sobre el uso y las ventajas obtenidas mediante la aplicación de ontologías a los distintos tipos de sistemas autónomos considerados (robots móviles, sistemas multiagente, e informática autonómica).

Categoría	Subcategoría	Ejemplos
Genérica		Ontologías basadas en la guía SWEBOK [MA05b], [MA05a], [SCR05]
Específicas	Requisitios	Generic Requirement Analysis Method based on Ontologies (GRAMO) [GdFB04] and ONTODM [GF03]
	Design	Diseño basado en ontologías [Hru05] Ontologías para diseño de software [Kal00]
	Mantenimiento	Ontologías para mantenimiento software [DAdO03], [AdOD06], [RVPG04], [VAO$^+$05]
	Calidad del Software	Ontologías para calidad del software [FGD02] Ontología para métrica del software [BVG06] Ontología para métrica y calidad del software [MO04]
	Descripción del proceso de ingeniería software	Entorno de desarrollo basado en ontologías (ODE) [FGD02], [FNM$^+$03], [FRPM04], [Fal04], [FRM05] Entorno Onto–ActRE [LG05] Definición de requisitos basada en ontologías (ORE) [Sae04] Ontologías de proceso software (SPO) [LQL05] Ontología para metodologías de desarrollo y tareas [GPHS06]

Table 2.1: Ejemplos de Ontologías de Dominio

Categoría	Subcategoría	Ejemplos
Durante el desarrollo de aplicaciones	Para procesos de ingeniería	DOSDE [dOZR+04] y EOSDE [OVRT06] Proceso de mantenimiento: MANTIS [RGPP02]
	Otros procesos	Ontología para la ingeniería software distribuída [WCD05], [WCC05]
Durante la ejecución de aplicaciones	Como parte de la arquitectura	Arquitectura basada en ontologías (ODA) [For01], [Knu04] Componentes software en un servidor de aplicaciones [OESV04]
	Como soporte a la información	Infraestructura basada en ontologías para sistemas inteligentes [Ebe03] Entorno de información web basado en ontología (WISE) [TLQ+06]

Table 2.2: Ejemplos de Ontologías como Elementos del Software

Papel de la ontología	Ventaja obtenida	Robots Móviles	Sistemas multiagente	Informática Autonómica
Definición de la estructura del conocimiento	Estructura del conocimiento del dominio	[SBU$^+$03], [SB04], [SBM05]	[CD00], [MPO05]	[TT03], [SAL06]
Especificación y gestión del conocimiento	Ajuste del conocimiento	[UPS$^+$03], [WVH04], [PUS]	[CPN00], [CPNH05]	[MSSS$^+$07]
Reutilización y compartición del conocimiento del dominio	Reutilización del conocimiento	[SM05], [HB06]	[MPO04], [WVV$^+$01], [BAT97]	[SSM$^+$04]
Búsqueda basada en ontología	Incremento de la robustez de la aplicación	[BAM$^+$04], [Woo04], [Eps04], [JBK$^+$04]	[LSRH97], [CPL05]	[SASS04]
Integración de fuentes heterogéneas	Interoperabilidad	[CTSB04] , [IBGM08]	[CD00], [MPO05], [DMQ05], [Tam01], [NF05]	[LSAF06]

Table 2.3: Ontologías para sistemas autónomos: su papel y ventaja

Capítulo 3

INGENIERIA ONTOLOGICA DE OASys

3.1 Introducción

La ingeniería ontológica se refiere a las diferentes actividades en el proceso de desarrollo, las metodologías utilizadas, los lenguajes y las herramientas utilizadas para el desarrollo de una ontología. El proceso de ingeniería ontológica en relación con la Ontología de Sistemas Autónomos (OASys), objetivo de esta investigación, ha sido descrito en varios capítulos de esta tesis. Estos capítulos están dedicados a describir las características de diseño de OASys, su estructura, su contenido, su implementación y su uso.

Este capítulo presenta los requisitos generales para el desarrollo de la ontología, especificando cómo ésta debe ser. En función de ellos, las características de la ontología finalmente desarrollada se describen en las secciones siguientes. Por último, una descripción detallada de la estructura y el contenido de OASys, con una definición detallada de los conceptos clave, las relaciones y otros elementos considerados en la ontología, haciendo hincapié en las ideas subyacentes para su inclusión en OASys.

3.2 Requisitos de OASys

Para la ingeniería ontológica, un aspecto clave es definir el propósito de la ontología, que dirigirá su desarrollo y contenido adicional. Sabiendo para qué se elabora la ontología, permite centrarse en los elementos ontológicos esenciales que se incluirán, dejando de lado los no necesarios. Complementando el propósito de la ontología, el tipo de la ontología a considerar, bien sea una ontología de nivel superior a una ontología de tareas, pasando por una ontología de dominio. El tipo elegido determina el nivel de abstracción, la generalidad y la reutilización de los términos ontológicos que se incluyeron en la ontología. Diferentes criterios de diseño sirven como directrices para apoyar este desarrollo. La metodología y el lenguaje elegido para la implementación de la ontología también se especifican y se justifican en esta sección.

Objetivo OASys definirá los conceptos y las elementos que deben considerarse en la definición de la estructura, la función y el comportamiento de los sistemas autónomos. La ontología también captura y hace uso de los conceptos necesarios para apoyar el proceso de ingeniería de cualquier sistema autónomo. La ontología proporcionará los elementos de construcción para construir un sistema autónomo, a través de la caracterización de los objetivos, el sistema y las características del sistema de control.

Tipo de la ontología OASys será una ontología de dominio para describir el dominio de los sistemas autónomos. Una ontología de dominio proporciona los conceptos y sus relaciones dentro de un dominio concreto, sobre las actividades en ese ámbito y sobre las teorías y los principios rectores de ese dominio [GPFLC04b]. El ser una ontología de dominio permite un alto nivel de facilidad de uso, ya que captura el conocimiento del dominio en una solución independiente del problema, estando limitada su reutilización a los aspectos relacionados con sistemas autónomos.

Estructura general OASys tendrá que abordar dos aspectos diferentes en su estructura. Por un lado, la necesidad de cubrir tanto sistemas generales como sistemas autónomos. Por otra parte, su propósito de proporcionar elementos ontológicos, tanto para el análisis del sistema autónomo como para su proceso de ingeniería.

Criterios de diseño Para asegurar su coherencia y su calidad, OASys se desarrollará teniendo en cuenta los criterios de diseño resumidos en la Sección 4.4 en la parte en inglés, como pautas objetivas para el desarrollo de la ontología. Entre los diferentes criterios posibles, OASys prestará especial atención a la claridad, la coherencia, la posibilidad de ampliaciones, el minimizar el efecto del lenguaje utilizado y el minimizar los compromisos ontológicos.

Adquisición del conocimiento OASys abarcará el amplio dominio de los sistemas autónomos. Por lo tanto, la ontología debe considerar elementos de una gran variedad de teorías y dominios, para considerar los diferentes tipos de sistemas autónomos. En un nivel más abstracto, se analizarán las teorías relacionadas con los sistemas generales, la mereología, y la topología. Diferentes teorías de ingeniería de sistemas serán consideradas como un metamodelo para un refinamiento posterior para la ingeniería de sistemas autónomos.

En cuanto al dominio de los sistemas autónomos, se estudiará una amplia gama de temas, tales como: Sistemas Autónomos Inteligentes, Robótica, Tecnología de Agentes, Ingeniería de Control, procesos continuos, y Ciencias Cognitivas. Se prestará especial atención a las ideas subyacentes y las teorías desarrolladas en el Proyecto ASys [SR08], como columna vertebral de la ontología. La investigación no sólo proporcionará los conceptos para la caracterización de los sistemas autónomos, sino también propondrá un marco para desarrollar estos sistemas.

Metodología El objetivo de esta investigación no es desarrollar una metodología nueva para el desarrollo de la ontología, sino utilizar una existente y bien establecida entre las diversas metodologías disponibles. La metodología elegida debe ser lo suficientemente adecuada para el desarrollo de OASys, pudiendo ser necesario ajustarla.

Formalización OASys se formalizará mediante lenguajes para ingeniería software generales y de propósito específico, tales como UML [Obj05] y SysML [Gro08a]. Las razones para elegir estos idiomas son de dos tipos: las técnicas de ingeniería de software para desarrollar ontologías se centran en su mayoría en UML, tal como fue descrito en la Sección 5.3 y el Capítulo 6 de la parte en inglés; en segundo lugar, la ontología desarrollada servirá de soporte software al proceso de ingenieria del proyecto global ASys [SHH+09], [SHG+09].

Reutilización Un aspecto principal de la ontología es la de su reutilización, es decir, la posibilidad de ser utilizada fuera del proyecto ASys. La ontología nace con el objetivo de proporcionar la base para el desarrollo de cualquier sistema autónomo, cuando tales sistemas abarcan una amplia gama de subdominios y aplicaciones diferentes.

3.3 Características de OASys

Esta sección describe como los requisitos antes mencionados han sido finalmente implementados en el proceso de ingeniería ontológica de OASys.

3.3.1 Estructura

OASys se ha organizado de acuerdo con dos dimensiones ortogonales: una compuesta por diferentes niveles de abstracción para separar el conocimiento general del particular, y la otra la separación entre el análisis del sistema autónomo y su proceso de ingeniería.

Para hacer frente a los diferentes niveles de abstracción, OASys ha adoptado una estructura estratificada teniendo en cuenta las ideas subyacentes de la focalización progresiva del dominio como se describe en [SAS+99], donde un problema tan complejo como el diseño de controladores inteligentes autónomos se aborda por la descomposición del análisis en etapas, desde el dominio hasta la aplicación. Este hecho coincide con la necesidad de considerar diferentes niveles de abstracción, desde los sistemas generales a las aplicaciones específicas. Por lo tanto, la ontología ha examinado el dominio de un sistema general, así como los más específicos de los sistemas autónomos, donde los conceptos son más especializados.

El nivel más abstracto es la capa de dominio superior. Esta capa contiene los conceptos relacionados con la Teoría General de Sistemas y los conceptos adicionales de mereotopología para expresar las relaciones mereológicas y topológicas en cualquier sistema. Para esta capa, se han analizado y considerado ontologías existentes ya definidas, para su integración dentro de OASys. Una segunda capa, la capa de dominio ASys centraliza el proceso de ingeniería ontológica de esta investigación, proporcionando los conceptos y elementos relacionados con la descripción de la estructura, la función y el comportamiento del sistema autónomo. El enfoque son los conceptos sobre sistemas autónomos en el marco del proyecto de investigación ASys, siendo no obstante suficientemente generales como para ser reutilizados en el desarrollo de un sistema autónomo cualquiera. En general, la capa superior define los conceptos que podrían ser instanciados en la inferior. La capa inferior especializa y detalla los conceptos que se utilizan para una aplicación real dentro del dominio de los sistemas autónomos.

Para hacer frente a la separación entre la caracterización de los sistemas autónomos y de la ingeniería, OASys se ha estructurado en dos ontologías, que se muestran en la Figura 7.1. La **Ontología de ASys** reúne los conceptos, relaciones, atributos y axiomas para caracterizar un sistema autónomo. La **Ontología de Ingeniería de ASys** recoge los elementos ontológicos para describir y apoyar el proceso de construcción de un sistema autónomo. Cada una de las dos ontologías se expande a través de las diferentes capas para describir los conceptos de mayor nivel de abstracción al inferior.

3.3.2 Criterios de Diseño

Los criterios de diseño se han seguido durante todo el desarrollo de OASys. Las siguientes secciones explican cómo el requisito original se ha abordado, y finalmente se ha llevado a cabo durante la definición y el desarrollo de la ontología.

3.3.2.1 Claridad

Para la exigencia de claridad, diferentes ontologías y glosarios existentes se han revisados para incluir los términos necesarios para OASys. El objetivo era beneficiarse de los conceptos que han existido y ya se han utilizado en la comunidad científica. Este proceso fue especialmente importante para la capa de dominio de sistemas, donde se utilizaron conceptos pertenecientes la teoría de sistemas generales, la mereología, la topología y la ingeniería de sistemas.

Los conceptos de la capa de dominio ASys proceden principalmente de los esfuerzos del equipo de investigación ASLab. Los diferentes documentos disponibles han sido detenidamente analizados para extraer los elementos ontológicos, identificando la similaridad o disparidad entre ellos. Tales conceptos han sido analizados posteriormente con los miembros del grupo de investigación para acordar el sentido y el significado deseado para nuestra investigación. Por último, todas las construcciones ontológicas (conceptos, relaciones, atributos, axiomas) se han definido en lenguaje natural. Estas definiciones han sido recogidas en el Anexo.

3.3.2.2 Subontologías y Paquetes

Para atender a las extensiones de OASys en el futuro, los conceptos dentro de una capa han sido clasificados mediante dos elementos distintos. El primer elemento son las *subontologías*, lo que permitirá modificar aspectos concretos de la ontología, sin la necesidad de cambiar todo. Diferentes subontologías se han diseñado para clasificar los conceptos estrechamente relacionados con un punto de vista concreto.

El segundo es el uso de *paquetes*, para clasificar los conceptos dentro de una subontología particular, de acuerdo a un aspecto concreto. La posibilidad de organizar elementos relacionados en paquetes prevé el aumento en el número de elementos ontológicos a medida que la ontología evolucione con el tiempo.

El beneficio de estos dos elementos de organización permite la ampliación o modificación de la ontología, sin grandes cambios en su estructura y composición. Dentro de una capa, nuevas subontologías se pueden añadir para ampliar las ya existentes con el propósito de abordar un nuevo punto de vista cuando la ontología se amplía. Dentro de un subontología, los paquetes se pueden agregar con conceptos nuevos de un aspecto particular. Los paquetes existentes pueden ser modificados o actualizados añadiendo o suprimiendo conceptos.

Por lo tanto, diferentes subontologías se han considerado en OASys, para contener los conceptos relacionados. Dentro de cada subontología, se han definido paquetes para recoger los conceptos relacionados con un aspecto concreto. Como se mencionó, las subontologías y los paquetes facilitarán el proceso de modificación, ampliación o actualización de la ontología con subdominios nuevos o aplicaciones no consideradas.

- **Ontología de ASys**

  Para la Ontología de ASys, se han desarrollado dos subontologías en relación a los diferentes niveles de abstracción en la descripción del sistema autónomo. La *Subontología de Sistemas* contiene los elementos ontológicos necesarios para definir la estructura del sistema, el comportamiento y la función. La *Subontología de ASys* especializa y perfecciona los conceptos anteriores, agregando conceptos específicos para sistemas autónomos.

  En el nivel superior de abstracción, la *Subontología de Sistemas* contiene diferentes paquetes recopilando los conceptos para la definición de cualquier sistema: Sistemas Generales, Mereología y Topología.

  En un nivel inferior de abstracción, la *Subontología ASys* consta de varios paquetes para la descripción del sistema autónomo. Los elementos ontológicos dentro de los paquetes incluidos en este subontología detallan y especializan los conceptos y las relaciones definidas en la Subontología de Sistemas: percepción, conocimiento, pensamiento, acción y equipos.

- **Ontología de Ingeniería de ASys**

  Para la Ontología de Ingeniería de ASys, se han considerado dos subontologías diferentes. La *Subontología de Ingeniería de Sistemas* reúne los conceptos relacionados con cualquier proceso de ingeniería de sistemas lo más general posible teniendo como base la ingeniería de sistemas y las metodologías de ingeniería de software. La *Subontología de Ingeniería de ASys* contiene la especialización y otros elementos necesarios para describir el proceso de ingeniería de un sistema autónomo.

  La *Subontología de Ingeniería de Sistemas* proporciona los conceptos del proceso de ingeniería de cualquier sistema, basados en las teorías ya desarrolladas para la ingeniería de software, organizada en paquetes para abordar diferentes aspectos de la ingeniería del sistema: requisitos, perspectivas, procesos de ingeniería, e ingeniería basada en modelos.

  La *Subontología de Ingeniería de ASys* proporciona los conceptos de un proceso de ingeniería para sistemas autónomos, con diferentes paquetes para recoger elementos relacionados, como especialización de los conceptos del nivel superior: requisitos, perspectivas, y el proceso de ingeniería de un sistema autónomo. Mencionar que esta subontología aún no incluye un paquete de ingeniería basada en modelos para sistemas autónomos (como una especialización de la capa superior) ya que la ingeniería basada en modelos dentro del proyecto de investigación general se especificará en próximas etapas.

3.3.2.3 Uso de Representaciones Intermedias

Con el intento de minimizar el efecto del lenguaje utilizado, es decir, la conceptualización errónea de conceptos basados en la sintáxis del lenguaje final utilizado, diferentes representaciones tabulares y gráficos intermedios se han utilizado para definir los elementos ontológicos.

Como guía, se han utilizado inicialmente las tablas sugeridas en METHONTOLOGY para las tareas de conceptualización (glosario de términos, relaciones binarias, atributos, constantes, axiomas, reglas), implementado como libros de Excel. Aunque no es la mejor herramienta posible, su facilidad de uso y su disponibilidad facilitó su uso como herramienta de partida para conceptualizar la ontología. La metodología permite la flexibilidad suficiente para adaptar o modificar estas representaciones tabulares para la ontología en desarrollo. Por lo tanto, ciertos campos adicionales como la fuente (para mostrar el documento, los expertos externos o las ontologías existentes como entrada), capa y subontología para especificar dónde se definió el elemento ontológico, el estado (tanto notas personales o fase de aprobación) se añadieron a las tablas originales.

3.3.2.4 Especialización de los Conceptos Fundamentales

En cada capa sólo los conceptos fundamentales se han descritos. La capa inferior añade más detalles sobre estos conceptos, para proporcionar un mayor nivel de detalle. Además, sólo los conceptos de interés para el ámbito de los sistemas autónomos se han considerado inicialmente. Nuevos conceptos se añadirán a medida que nuevas aplicaciones o desarrollos se describan mediante el uso de OASys.

3.3.2.5 Convención en la Nomenclatura

La elección de una convención para la nomenclatura, y adherirse a ella hace que la ontología sea más fácil de entender y ayuda a evitar algunos errores comunes de modelado. Por lo tanto, el siguiente conjunto de convenciones para la ontología se ha adoptado:

1. Mayúsculas
   - Cada concepto empieza con una letra mayúscula. Si el nombre contiene más de una palabra, cada una empezará por mayúscula, por ejemplo, ModeloDelSistema.
   - Cada atributo con minúsculas. Si el nombre del atributo contiene más de una palabra, la primera palabra es toda en minúsculas y cada una de las restantes se capitalizan, por ejemplo, estadoDelObjetivo.
   - Cada relación sigue la misma convención que los atributos.
   - Axiomas y normas se expresan en lenguaje natural, con relación a los conceptos definidos.
2. Uso del singular

   Para el nombre de los elementos de la ontología (conceptos, atributos, relaciones, etc), la forma singular es la preferida, por ejemplo, sistema en vez de sistemas.
3. Convenios adicionales
   - Los nombres de las ontologías, subontologías y paquetes comenzarán por mayúsculas.
   - Las ontologías y subontologías agregarán la palabra Ontología y Subontología a su nombre. El término Paquete será utilizado de manera similar.
   - Palabras como concepto, atributos y demás, no debe utilizarse como parte del nombre de un elemento.

3.3.3 Fuentes

La adquisición de conocimientos para el desarrollo de la ontología se puede hacer considerando diferentes fuentes: documentos, ontologías existentes y expertos. Documentos como artículos, informes técnicos, o los libros sirven como una fuente de entrada para los elementos ontológicos. Las ontologías existentes también deben ser revisadas, ya que el dominio podría haber sido conceptualizado, sin embargo con un punto de vista o un próposito diferente. Estas ontologías existentes deben ser seleccionadas, evaluadas y, finalmente, total o parcialmente reutilizadas, prestando atención al nivel de granularidad (si la ontología actual cubre el mismo nivel de detalle que en la ontología en desarrollo). Los expertos del dominio también actúan como una posible fuente para la conceptualización ya que brindan las palabras y los términos con los que están familiarizados al hablar sobre el dominio.

A los efectos del desarrollo de la ontología, principalmente los documentos y ontologías existentes han sido consideradas. Los documentos han sido analizados en lo referente a su terminología y las definiciones de los diferentes dominios, subdominios, las aplicaciones y los aspectos considerados en la estructura de la ontología. Glosarios y ontologías ya existentes se han estudiado para determinar su integración en la ontología. Expertos disponibles en algunos de los ámbitos de actuación de la ontología se han cuestionado para la adquisición de algunos conceptos, sin ser la fuente principal.

3.3.4 Metodología: Proceso de desarrollo basado en METHONTOLOGY

De las metodologías disponibles y los métodos para la ingeniería de la ontología, METHONTOLOGY [FLGPJ97] ha sido elegida como punto de partida ya que:

1. Las etapas para el proceso de desarrollo están bien y claramente definidas en un ciclo de vida de la ontología.
2. Comprende los elementos y las tareas que hay que considerar, tales como el mantenimiento de la ontología.
3. La actividad de conceptualización se descompone en diferentes tareas detalladas.
4. Uso de representaciones intermedias, tales como tablas y gráficos.
5. Flexibilidad en el proceso (las tareas pueden revisarse si se agregan nuevos conceptos), en la representación (las tablas pueden ser modificadas según sea necesario) y la evolución de prototipos (cada vez se obtiene un nuevo prototipo en el que agregar, modificar o eliminar términos).

METHONTOLOGY propone tanto un proceso de desarrollo de la ontología como un ciclo de vida de la ontología, estrechamente entrelazados. El proceso de desarrollo se refiere a las actividades que se realizan durante la construcción de la ontología. El ciclo de vida de la ontología identifica cuándo deben llevarse a cabo estas actividades, a través de un conjunto de etapas que define las actividades a realizar en cada etapa y cómo se relacionan. Tanto el proceso de desarrollo y las actividades del ciclo de vida han sido inicialmente seguidos para el desarrollo de OASys. Se han considerado también pautas adicionales descritas en [NM01], [Miz04] y [BA07]. Se han ajustado y adaptado algunas de las actividades propuestas en la metodología durante el desarrollo de OASys:

- Actividades de gestión: consisten en las actividades de programación y el control de calidad de la ontología. Estas actividades se han realizado mediante el establecimiento de manera general de un calendario, tiempo y esfuerzo necesarios para desarrollar OASys durante la investigación. Las actividades de control se han realizado tanto por el desarrollador de la ontología, así como el equipo de apoyo.

- Actividades de desarrollo: estas actividades se agrupan dentro de METHONTOLOGY en actividades de predesarrollo, desarrollo y posdesarrollo. Estas actividades se han llevado a cabo durante el desarrollo del OASys. Señalar que, debido a la estructura basada en capas y subontologías, las actividades de desarrollo se han aplicado a cada capa y dentro de ella, a cada subontología y a cada paquete.

  1. Especificación: el objetivo es determinar qué se considera como dominio, el uso dado a la ontología y quién usará la ontología.

  2. Conceptualización: el objetivo es organizar y estructurar los conocimientos adquiridos mediante representaciones externas independientes de la representación del conocimiento y los paradigmas de aplicación de la ontología en la que se formalizará después. Se incluyen diferentes tareas, como la construcción de un glosario, las taxonomías, las relaciones, el detalle de los elementos anteriores y los axiomas.

     Para cada una de las ontologías, subontologías y paquetes, se han seguido y adaptado las tareas de conceptualización. Ejemplos de los resultados se proporcionan en la sección que explica la subontología y paquetes en detalle.

  3. Formalización: el objetivo de esta actividad es formalizar el modelo conceptual. Para satisfacer el enfoque de ingeniería de software, OASys ha sido descrita utilizando UML [Obj05].

  4. Implementación: esta actividad se basa en modelos computables utilizando lenguajes ontológicos de aplicación.

  Durante el periodo de posdesarrollo, las actividades de mantenimiento de la ontología corrigen y actualizan la ontología para ser usada por otras ontologías y aplicaciones.

- Actividades adicionales: tales como la adquisición de conocimiento, la integración, la evaluación y la documentación. Para el desarrollo de OASys, principalmente la adquisición de conocimiento, la evaluación y la documentación se han llevado a cabo.

3.4 Descripción de OASys

Esta sección describe las ontologías, las subontologías y los paquetes considerados en OASys, prestando especial atención a la descripción del proceso de ingeniería ontológica de cada uno de ellos. Cada paquete ha sido desarrollado teniendo en cuenta las actividades de desarrollo propuestas de la metodología METHONTOLOGY, desde su conceptualización hasta su implementación final UML.

3.4.1 Ontología de ASys

La Ontología de ASys reúne, en diferentes niveles de abstracción, las construcciones ontológicas necesarias para describir y caracterizar un sistema autónomo.

La ontología se compone de dos subontologías: la *Subontología de Sistemas* para describir el nivel más alto de abstracción de conceptos de sistemas y la *Subontología de ASys* que contiene los elementos ontológicos del dominio de los sistemas autónomos. Las siguientes secciones describen en detalle los paquetes en cada una de las subontologías.

3.4.1.1 Subontología de Sistemas

El propósito de la Subontología de Sistemas es el de definir los conceptos y las relaciones al nivel más alto de abstracción en OASys, utilizados para describir cualquier sistema. Estos conceptos proporcionan una conceptualización común que puede ser usada para describir cualquier tipo de sistemas autónomos.

Para organizar los diferentes aspectos que generalmente coexisten en la caracterización de un sistema, los conceptos se han agrupado en diferentes paquetes. El paquete principal es el Paquete de Sistemas Generales, que contiene los conceptos cuya fuente principal es la teoría general de sistemas. Este paquete se complementa con otros dos. El primero, llamado Mereología, contiene los conceptos para expresar las relaciones mereológicas todo–parte. Un segundo paquete, llamado Topología, se utiliza para reunir los conceptos relacionados con las conexiones topológicas entre las partes.

Paquete de Teoría General de Sistemas

- Propósito y campo de aplicación: El Paquete de Teoría General de Sistemas ha sido desarrollado para reunir los conceptos relacionados con la teoría general de sistemas, como una teoría que permite describir cualquier tipo de sistemas. Esta generalidad es un elemento clave para considerar dicha teoría como el nivel más alto de abstracción para definir cualquier sistema autónomo.

- Especificación: Los elementos considerados en este paquete se centran principalmente las ideas desarrolladas en [Kli69], y que se detallan en [Kli85], [Kli91] y [KE03].

- Conceptualización: De la amplia gama de conceptos desarrollados en la teoría general de sistemas, sólo un conjunto de conceptos han sido considerados en el paquete. El criterio para incluir estos conceptos se refiere a los distintos rasgos característicos de un posible sistema, tanto desde un punto de vista estructural y de comportamiento, que incluyen las definiciones básicas (cantidades externas con el nivel de resolución, una determinada actividad, el comportamiento permanente, la estructura UC real y la estructura ST real) consideradas en [Kli69]. Por ello, el Paquete de Teoría General de Sistemas se ha organizado a su vez en tres paquetes diferentes: Nivel de resolución, estructura y comportamiento. El primero proporciona los conceptos relacionados con las cantidades, el nivel de resolución y la actividad del sistema. El segundo proporciona los elementos ontológicos para describir las diferentes estructuras consideradas en la teoría. El último se refiere al comportamiento de los sistemas y los elementos.

Las tareas de conceptualización han obtenido un glosario de términos descritos en el anexo. Se han analizado taxonomías de conceptos para encontrar relaciones de generalización o clasificación entre los conceptos. Por otra parte, se han representado de forma tabular las relaciones entre los conceptos para expresar los conceptos relacionados, el nombre de relación, la multiplicidad y otras propiedades.

- Formalización: Los conceptos, los atributos y las relaciones se han formalizado con elementos de modelado UML. Los conceptos han sido modelados como clases, con sus atributos. Las relaciones han sido modeladas usando, de la gama disponible de las asociaciones previstas por UML, principalmente asociación dirigida binaria (para las relaciones binarias) y la asociación de generalización (para taxonomías). Los diferentes paquetes considerados en el Paquete de la Teoría General del Sistema de se muestra en la Figura 7.9, en la Figura 7.10, y la Figura 7.11, con sus definiciones en la Sección 11.1.1.1 del Anexo .

Paquete de Mereología

- Propósito y campo de aplicación: El Paquete de Mereología se ha desarrollado para reunir los conceptos y las relaciones relativas a los aspectos mereológicos y meronímicos, es decir, las relaciones del todo–parte, tanto a un nivel de metamodelo como al nivel del sistema.

- Especificación: los conceptos considerados en el paquete (a nivel meta) son una selección de los ya desarrollados por [Bor97], [Gui05], [KA07] y [MWM07]. El objetivo de la investigación no era el de elaborar una teoría mereológica nueva, sino reutilizar conceptos de las diferentes teorías existentes (ver Figura 7.12). Los conceptos considerados en el paquete a nivel del sistema han sido elegidos entre los que se definen en [MBW$^+$08].

- Conceptualización: Los conceptos propuestos en las diferentes fuentes seleccionadas han sido alineados y combinados. Esto se ha hecho para garantizar que los conceptos y las relaciones tenían la misma granularidad. El análisis también permitió detectar posibles solapamientos o definiciones similares. Al final, un conjunto de conceptos, relaciones y axiomas han sido elegidos como los más adecuados para ser utilizados para la caracterización de cualquier sistema. El glosario de términos seleccionados, siempre con la fuente de material, se puede ver en la sección 11.1.1.2

del anexo.

Diferentes conceptos se han utilizado para describir la parte y el todo en cualquier sistema (Figura 7.13), para distinguir entre los elementos agregados y compuestos. Por otra parte, las relaciones tanto mereológicas como meronímicas han sido consideradas. Los conceptos y las relaciones mereológicos permitirán describir estructuras (de agregación o composición) y las relaciones espaciales entre el todo y las diferentes partes. Los nombres al nivel meta se han elegido principalmente de [MWM07], con conceptos adicionales de [Bor97] y [KA07]. Las relaciones meronímicas se han incluido para permitir la descripción de relaciones parte–todo que no comparten las mismas propiedades mereológicas, es decir, la transitividad entre los elementos relacionados, como se propone en [Gui05] y [KA07]. Esta última se ha utilizado como fuente para los nombres utilizados en este subontología. Una taxonomía uniendo las relaciones tanto mereológicas como meronímicas se da en la Figura 7.14 .

- Formalización: Algunos de los conceptos definidos y las relaciones, se han definido explícitamente para futuras ampliaciones de los conceptos mereológicos para otros lenguajes de ontologías que no proporcionan primitivas de construcción para las relaciones parte–todo [MWM07]. Sin embargo, el lenguaje de modelado UML posee elementos para el modelado de agregación y composición utilizando, respectivamente, las asociaciones de agregación y de composición. La diferencia entre la agregación y la composición en UML reside en la existencia de dependencia entre el todo y las partes, con la composición siendo una forma más fuerte de agregación, cuando la existencia de la parte depende de la existencia del conjunto. Un ejemplo de la aplicación del nivel meta y del nivel de sistemas se muestra, respectivamente, en la Figura 7.15 y la Figura 7.16.

  En cuanto a otras relaciones mereológicas y meronímicas definidas en la taxonomía, UML no proporciona un elemento de modelado directo. Guizzardi en [Gui05] propone una modificación de la agregación y la composición de UML, usando nuevos iconos para el resto de las relaciones. Sin embargo, Keet [KA07] descarta esta extensión por la dificultad en los diagramas (así como que no existe esta facilidad en las herramientas de desarrollo UML). Como planteamiento inicial, se va a utilizar asociaciones dirigidas con nombres de rol que coincidirán con los nombres ontológicos de las relaciones (restricciones expresadas en OCL podrían utilizarse para expresar los axiomas y las limitaciones relacionadas con una relación particular). Si el uso adicional de la subontología muestra una alta necesidad de estas relaciones, podría resultar útil dotar de un estereotipo a las asociaciones de agregación y composición con los nuevos nombres y roles.

Paquete de Topología

- Propósito y campo de aplicación: El Paquete de Topología se ha desarrollado con el propósito de recoger los conceptos topológicos y las relaciones existentes en un sistema. Cuando se realice la caracterización de un sistema, especialmente desde el punto de vista estructural, será necesario especificar qué elementos y subsistemas están relacionados con uno en particular. Esto permitirá obtener una descripción topológica del sistema considerado.

- Conceptualización: Los conceptos, relaciones y axiomas considerados dentro de este paquete han sido seleccionados de los definidos en [MWM07] a nivel metamodelo y [MBW$^+$08] a nivel de dominio del sistema (donde los rangos de las relaciones topológicas se limitan al dominio de los sistemas). No se han definido nuevas relaciones topológicas, ya que las existentes son suficientes para el objetivo de la investigación. Un resumen de los conceptos elegidos y las relaciones se dan en la sección 11.1.1.3 del Anexo.

3.4.1.2 Subontología de ASys

El propósito de la Subontología de ASys es el de proporcionar un modelo de conocimiento consensuado del dominio de los sistemas autónomos, al abordar la estructura de los sistemas autónomos, su comportamiento y su función. La subontología se organiza en paquetes diferentes (Fig. 7.17) que cubren los distintos aspectos y funcionalidades de un sistema autónomo. Dichas funcionalidades incluyen la percepción, el pensamiento y las acciones basadas en conocimiento que un sistema autónomo realiza.

Paquete de Percepción

- Propósito y campo de aplicación: El Paquete de Percepción reúne los conceptos y las relaciones sobre con los procesos de percepción que tienen lugar en un sistema autónomo.

- Especificación: Los elementos considerados en este paquete son algunos de los conceptos y las ideas desarrolladas en [UIT06e] y [UIT06a]. Una descripción detallada de los procesos de percepción y detección, con énfasis en sus características, su contexto y los aspectos cognitivos especialmente de estos procesos se da en [Lóp07].

- Conceptualización: En primer lugar, se ha obtenido un glosario de términos, mediante una representación intermedia. Las taxonomías de conceptos también se han definido. A continuación, las relaciones entre los conceptos se han caracterizado en tablas y representaciones gráficas para expresar los conceptos relacionados, el nombre de relación, la multiplicidad y otras propiedades, usando cuando sea necesario las relaciones mereological. Con los elementos anteriores, se ha hecho un Diccionario de Conceptos (Tabla 7.7). Por último, se han idenficado los distintos atributos.

- Formalización: los conceptos, los atributos, las relaciones y los elementos obtenidos en la tarea anterior, se han formalizado con elementos de modelado UML. Los conceptos adoptaron la forma de clases. Las relaciones de taxonomía han sido modeladas mediante las asociaciones de generalización de UML. Las relaciones mereológicas han sido formalizadas mediante las asociaciones de agregación y de generalización. UML no distingue entre clases y atributos de instancia [GPFLC04b], por tanto, los atributos se han especificado en la sección correspondiente en cada clase de concepto. La formalización general se da en la Figura 7.18 como un diagrama UML, con las definiciones ontológicas de los elementos en la Sección 11.1.2.2 del Anexo.

Paquete de Conocimiento

- Propósito y campo de aplicación: El Paquete de Conocimiento se ha considerado para reunir los conceptos y las relaciones en relación con los elementos de conocimiento utilizados por un sistema autónomo en su pensamiento y las operaciones de evaluación.

- Especificación: Diferentes publicaciones de los miembros de ASLab ([SR08], [Lóp07], [HSL08], [SHR07], [AT06b]) se han utilizado como fuentes principales, complementados con fuentes externas para aclarar conceptos.

- Conceptualización: La tarea de conceptualización del paquete sigue la metodología adoptada, obteniendo un glosario de términos como representación intermedia, como se muestra en la Tabla 7.8, que se detalla en la Sección 11.1.2.3 en el Anexo.

  Diferentes taxonomías de conceptos se han definido como se muestra en la tabla 7.9, donde el campo Superconcepto muestra el concepto padre del concepto relacionado. Se han caracterizado también relaciones entre algunos conceptos. Ciertos atributos han sido también detallados. Un diccionario de conceptos especifica cada concepto, detallando sus atributos y relaciones. Representaciones gráficas intermedias se han utilizado para caracterizar las taxonomías de conceptos, como se muestra en la Figura 7.19.

  El conocimiento ha sido definido como estructura de información internalizada que es isomorfo desde una perspectiva útil a una parte determinada de la realidad. Inicialmente, las estructuras de información, que utiliza un sistema autónomo para pensar y evaluar las operaciones, se han adquirido, por definición y por aprendizaje (por ejemplo, objetivos, algoritmos, etc) o por transformación de las fuentes de otros procesos (por ejemplo, las cantidades conceptuales de magnitudes físicas) en un nivel conceptual. Más tarde, los elementos conceptuales se basarán, según sea necesario durante las operaciones de otros, en verdaderos elementos físicos en el sistema autónomo. De ahí que el Paquete de Conocimiento se haya organizado en diferentes paquetes para hacer frente a los diferentes tipos de estructuras de datos (Figura 7.20).

- Formalización: los elementos de modelado UML se han utilizado para formalizar los distintos paquetes conceptualizado en la fase anterior, cada uno como un diagrama de clases UML (Fig. 7.21a, Fig. 7.21b , Fig. 7.21c, Fig. 7.21d , Fig. 7.21e y Fig. 7.21f).

Paquete de Pensamiento

- Propósito y campo de aplicación: Los conceptos incluidos en este paquete se refieren a las ideas principales que se describen en [UIT06d] y [UIT06b].

  El Paquete de Pensamiento se ha centrado en conceptualizar el pensamiento como un proceso abstracto basado en conocimiento, en el que conocimiento hace referencia a los conceptos definidos en el Paquete de Conocimiento (Figura 7.21).

- Especificación: las diferentes publicaciones de los miembros de ASLab se utilizaron como fuentes principales.

- Conceptualización: Teniendo en cuenta la finalidad descrita anteriormente, el paquete de Pensamiento ha prestado especial atención a los conceptos relacionados con los Objetivos y Algoritmos. Términos como GeneraciónDeObjetivo, CicloDeVidaDelObjetivo, y ReconFiguraciónDelObjetivo han sido considerados. Para los relacionados con Algoritmos, el concepto de DescomposiciónFuncional que implica procesos llevados a cabo cuando los Algoritmos se han conceptualizado.

- Formalización: los elementos de modelado UML se han utilizado para formalizar los elementos del paquete conceptualizado en la etapa anterior en el diagrama UML de la Figura 7.22, cuyos elementos ontológicos se especifican en la Sección 11.1.2.4 del Anexo.

Paquete de Acción

- Propósito y campo de aplicación: el Paquete de Acción define los elementos ontológicos relacionados con las distintas actividades y entidades que participan en las operaciones de un sistema autónomo.

- Especificación: De las diferentes publicaciones de ASLab, tanto [Lóp07] como [HSL08] han desempeñado un papel especial para la conceptualización de los conceptos.

- Conceptualización: un Glosario de Términos se ha obtenido como resultado principal de la conceptualización, que se muestra parcialmente en la Tabla 7.11 y más detallado en la Sección 11.1.2.5 del Anexo. El glosario se ha complementado con las taxonomías de los conceptos y la representación gráfica de las relaciones entre los conceptos en el paquete.

El punto clave en este paquete es la conceptualización de los aspectos de qué, quién y cómo actúa el sistema autónomo. En primer lugar, el concepto de Operación para reflejar las posibles tareas o actividades que pueden ser llevadas a cabo por un sistema autónomo. Estas operaciones consisten en los procesos sobre las Cantidades sistema. Un primer grupo considera que las operaciones se centraron en sólo CantidadesConceptuales, conceptualizado como OperaciónConceptual. Otro tipo importante de operación es el de Actuación, como la conceptualización de las operaciones que implica una especie de transformación conceptual de las cantidades en cantidades físicas, tales como la Percepción y el Grounding. Un tercer concepto se refiere a las posibles operaciones con las cantidades ya asimiladas, definida como OperaciónAsimilada. Por último, hay algunas operaciones que implican algún tipo de Interacción con el entorno del sistema, como la Detección y Acción. Todos estos conceptos han sido relacionados en la taxonomía de la Figura 7.23.

En segundo lugar, la conceptualización de los elementos que llevan a cabo esas operaciones, que se define como Actor. Este concepto puede ser más detallado como se requiere en los conceptos de Agente (con un enfoque de software) y de Actuador (con una orientación de hardware). Por último, dentro del Paquete de Acción, los conceptos relacionados con los aspectos de Control del sistema autónomo también se han considerado.

- Formalización: Distintos elementos de modelado UML, se utilizaron para formalizar los conceptos y las relaciones conceptualizados en la etapa anterior, como un diagrama de clases UML dado en la Figura 7.24.

Paquete de Equipos

- Propósito y campo de aplicación: Este paquete proporciona los conceptos para describir los distintos equipos y sus propiedades.

- Especificación: las diferentes publicaciones de los miembros de ASLab [RJB04], [HH08], [dlM09b] se han utilizado como fuentes principales. Se centran en la descripción de las características físicas de las estructuras de los diferentes sistemas y subsistemas, así como las funcionalidades requeridas.

- Conceptualización: Un glosario de términos ha sido obtenido como el resultado principal de la conceptualización, complementada con las taxonomías de los conceptos y la representación gráfica de las relaciones entre los conceptos de la subontología.

  Los conceptos recogidos en esta capa se centran en los términos y las relaciones que actualmente se utilizan para referirse a las aplicaciones en desarrollo. Conceptos generales, tales como sistema o subsistema pueden, especializarse en conceptos tales como el Nodo, Equipo, Componente, Artefacto, que están más cerca de la definición e implementación de software del hardware. Los diferentes tipos de conexiones, los puntos de conexión y los puertos pueden ser especializarse para modelr equipos interconectados o redes de información.

Es preciso señalar que, aunque la fase de conceptualización debe ser independiente de la formalización final, se ha hecho una excepción. El concepto de Equipo que plantea cierta controversia. Por un lado, es un concepto utilizado por los ingenieros para referirse a cualquier elemento físico en el desarrollo de un sistema. Por otra parte, el concepto en lenguaje UML implica una unidad física computacional. Como el proyecto ASys va a utilizar los dos tipos, se ordenaron los conceptos en dos paquetes: Paquete de EquipoComputacional (para considerar los dispositivos de computación tales como computadoras, servidores, etc) y el Paquete de EquipoFísico (para otros tipos de dispositivos tales como cámaras fotográficas, las muñecas, GPS, etc), como parte del Paquete de Equipos (Fig. 7.25).

- Formalización: distintos elementos de UML se han utilizado para formalizar los conceptos y las relaciones conceptualizados en la etapa anterior. El Paquete de EquipoComputacional puede verse en la Figura 7.26 y el Paquete de EquipoFísico en la Figura 7.27. Los diferentes elementos ontológicos se describen en la sección 11.1.2.1 del Anexo.

3.4.2 Ontología de Ingeniería de ASys

Esta ontología es la segunda parte de OASys, y está dedicada a los elementos ontológicos relacionados con la ingeniería de sistemas. Se compone de dos subontologías para hacer frente a un nivel diferente de abstracción de los elementos ontológicos empleados en describir el proceso de ingeniería del sistema autónomo (Figura 7.28): la Subontología de Ingeniería de Sistemas que recoge los conceptos relacionados con la ingeniería de sistemas generales, y la Subontología de Ingeniería de ASys que recoge de una manera similar los elementos de ingeniería de sistemas autónomos, como especialización de los anteriores.

3.4.2.1 Subontología de Ingeniería de Sistemas

Esta subontología reune los conceptos y las relaciones relacionadas con el proceso de ingeniería de sistemas y software. Está basada en metamodelos diferentes, especificaciones y glosarios que se han utilizado durante mucho tiempo para el software. La subontología contiene paquetes diferentes para abordar diferentes aspectos a considerar en el desarrollo de un sistema: los requisitos, el proceso de ingeniería, las perspectivas y la ingeniería basada en modelos (Fig. 7.29).

Paquete de Requisitos

- Propósito y campo de aplicación: El Paquete de Requisitos reúne los conceptos sobre los requisitos del sistema, que generalmente se establecen durante el proceso de ingeniería. Los elementos del paquete han sido escogidos de fuentes ya existentes sobre los requisitos de definición y clasificación, como [IEE90], [Obj09b] y [Gro08a].

- Conceptualización: en esta fase se definen los conceptos, taxonomías, relaciones y atributos relacionados con los requisitos definidos en las fuentes antes mencionadas. Un requisito es definido como una capacidad o condición que debe (o debería) ser satisfecha por un sistema. En general se identifican por un nombre, un texto de descripción, su fuente, un nivel de riesgo y algún tipo de método para verificar que se cumplen. Los requisitos se clasifican como funcionales, de interfaz, de prestaciones, físicos o de diseño.

  Los requisitos son generalmente capturados utilizando casos de uso, como un medio para describir los requisitos del sistema o del subsistema de considerar y los actores del caso de uso, como entidades externas al sistema que interactúan con él.

- Formalización: Los distintos conceptos y las relaciones se han formalizado en un diagrama de clases UML (Figura 7.31) y sus definiciones se pueden encontrar en la Sección 11.2.1.1 del Anexo.

Paquete de Proceso de Ingeniería

- Propósito y campo de aplicación: Este paquete reúne conceptos generales para describir las etapas y pasos del proceso de ingeniería de un sistema tomando como base el metamodelo de procesos de ingeniería software (SPEM 2.0) [Gro08b].

- Conceptualización: Los conceptos y relaciones en este paquete definen términos generales para ser utilizado en cualquier proceso de ingeniería de sistema, sin restringir a un proceso de ingeniería en particular.

- Formalización: Los diferentes elementos ontológicos se han formalizado en un diagrama de clases UML que muestra los conceptos como clases, las relaciones como asociaciones UML (Figura 7.32). Las definiciones del paquete se dan en la Sección 11.2.1.2 del Anexo.

Paquete de Perspectivas

El Paquete de Perspectivas reúne los elementos de la ontología necesarios considerar los diferentes puntos de vista o perspectivas en el desarrollo de la arquitectura de cualquier sistema. Los conceptos y las relaciones han sido elegidos de entre los descritos en [Ins00], que proporciona un marco conceptual con los términos más comunes para este propósito. El Paquete de Perspectivas se ha formalizado como un diagrama UML (Figura 7.33), cuyos elementos se resumen en la sección 11.2.1.3 del Anexo.

Paquete de Ingeniería Basada en Modelos

Este Paquete proporciona los conceptos fundamentales y las relaciones relacionados con la ingeniería basada en modelos, que se refiere a una gama de enfoques de desarrollo que se basan en el uso de modelos como una forma primaria de expresión . Los elementos ontológicos en este paquete se han elegido entre los de [SV06]y [Obj03].

La formalización de los conceptos y las relaciones se ha hecho como un diagrama de clases UML (Figura 7.34) y los elementos ontológicos se resumen en la Sección 11.2.1.4 del Anexo.

3.4.2.2 Subontología de Ingeniería de ASys

La Subontología de Ingeniería de ASys se centra en los procesos de ingeniería de los sistemas autónomos, reuniendo los conceptos y relaciones para describir un proceso de ingeniería genérica o ciclo de vida para un sistema autónomo. Se trata de un perfeccionamiento o especialización de los conceptos ya definidos en la Subontología de Ingeniería de Sistemas, organizada en diferentes paquetes (Figura 7.35).

Paquete de Requisitos de ASys

- Propósito y campo de aplicación: el Paquete de Requisitios de ASys reúne los conceptos necesarios para definir los requisitos de un sistema autónomo, ya sea como nuevos conceptos o la especialización del conjunto de requisitos del sistema que definen los requisitos a un mayor nivel de abstracción.

- Conceptualización: Para llevar a cabo el proceso de ingeniería de un sistema autónomo, se lleva a cabo la caracterización de diferentes aspectos [SR08]: Caracterización del Producto, Caracterización del Proceso y Caracterización del Sistema. Los dos últimos comprenden varios conceptos que han sido clasificados además en dos paquetes diferentes.

- Formalización: los conceptos anteriores han sido formalizados como diagramas de clases UML, para los elementos de Caracterización del Proceso (Fig. 7.37) y de los conceptos de Caracterización del Sistema (Figura 7.38). La definición de los elementos ontológicos figura en Sección 11.2.2.1 del Anexo.

Paquete de Perspectivas de ASys

- Propósito y campo de aplicación: Este paquete proporciona los conceptos y otros elementos para definir las distintas perspectivas que generalmente se consideran en la caracterización del sistema autónomo. Los distintos conceptos y relaciones en este paquete se han extraído de [RJB04], [MWM07], [vL08], [FMS08].
- Conceptualización: los sistemas autónomos suelen ser demasiado complejos para ser abordados en su totalidad, por lo que es común dividir el sistema teniendo en

cuenta diferentes puntos de vista. Para los sistemas autónomos, se han considerado los puntos de vista relativos a: los requisitos, la estructura, la función, el comportamiento y el rendimiento.

- Formalización: los distintos conceptos que se han formalizado en un diagrama UML (Figura 7.39), donde el concepto de Punto de Vista es importado del Paquete de Perspectivas. Los diferentes Enfoques no han sido incluidos en el diagrama en aras de la simplicidad. Las definiciones de todos los conceptos se proporcionan en la Sección 11.2.2.2 del Anexo.

Paquete de Proceso de Ingeniería de ASys

- Propósito y campo de aplicación: Este paquete contiene los elementos ontológicos especializarse de los términos de proceso de ingeniería. Varias fuentes se han utilizado para definir y organizar este Paquete, teniendo en cuenta los requisitos de autonomía, integración, tiempo real e ingeniería basada en UML y SysML considerados en el proyecto de investigación ASys [SMP99], [Dou04], [Seg05], [SR08], [Est08].

- Conceptualización: Los distintos conceptos definidos en el Paquete de Proceso de Ingeniería pueden especializarse para describir el proceso de ingeniería de un sistema autónomo. En este sentido, la fase de diseño pueden ser conceptualizados como: RequisitosDelASys, AnálisisDelASys, DiseñoDelASys, ImplementaciónDelASys y EvaluaciónDelASys.

  Hay que mencionar que aunque el objetivo de este Paquete es abordar todo el proceso de ingeniería, las tareas de conceptualización se han centrado en los conceptos pertenecientes a las Fases de Requisitos y Análisis.

  La Fase de RequisitosDelASys se centra en la definición de los requisitos del sistema autónomo por parte de los usuarios y los diseñadores. La Fase de AnálisisDelASys obtiene una descripción de alto nivel del sistema desde diferentes PuntosdeVista definidos en el Paquete de Perspectivas. La FaseDeDiseñoDelASys se refiere a una fase en la que se obtiene una descripción más detallada. La FaseDeImplementaciónDelASys obtiene la descripción de un sistema de implementación. La FaseDeEvaluaciónDelASys se refiere a la evaluación del sistema implementado.

  Las fases anteriores en esta capa heredan todas las características y las relaciones de los conceptos definidos en la capa de dominio del sistema. Por lo tanto, cada fase puede consistir en las diferentes tareas que están implicados en su consecución. La fase de RequisitosDelASys consta de la tarea de CasosDeUsoDelSistema para abordar la definición y el detalle de los casos de uso del sistema y subsistemas, a través de las subtareas de ModeladoDeCasosDeUso y DetalladoDeCasosDeUso. Como Productos, modelos tales como el ModeloDeCasosDeUso y la EspecificaciónDeCasosDeUso. La tarea de CaracterizaciónDeRequisitos se refiere a la especificación de requisitos funcionales y no funcionales, en la forma de CaracterizaciónDelProcso y CaracterizaciónDelSistema.

La fase de AnálisisDelASys comprende las tareas de AnálisisEstructural, AnálisisDeComportamiento, y AnálisisFuncional para diferenciar diferentes tipos de análisis en función del punto de vista seleccionado. El AnálisisEstructural se refiere al análisis del sistema en términos de su estructura, su topología y el conocimiento que posee. Como resultado, un ModeloEstructural para definir la estructura y topología del sistema y un ModeloDeConocimiento para representar los diferentes tipos de conocimiento en el sistema autónomo.

El AnálisisDelComportamiento es la tarea relacionada con el análisis del sistema en términos de su comportamiento y el proceso cognitivo. De manera similar, la tarea de AnálisisFuncional se centra en los procesos de descomposición y representación funcional, con la obtención de un ModeloFuncional que comprende la representación de Actor, Responsabilidad y Operación.

- Formalización: los diferentes conceptos y las relaciones se han formalizado en diagramas UML (Fig. 7.41 y Fig. 7.42) y sus definiciones correspondientes se dan en la Sección 11.2.2.3 del Anexo.

Capítulo 4

METODOLOGIA DE INGENIERIA BASADA EN OASys

4.1 Introducción

Este capítulo describe el próposito de la Metodología de Ingeniería basada en OASys a utilizar para el análisis y desarrollo de sistemas autónomos. Las diferentes secciones describen el propósito, el campo de aplicación, la descripción y otros elementos de la metodología.

4.2 Propósito y campo de aplicación

El propósito de la metodología es el de proporcionar algunas indicaciones y mostrar cómo se pueden utilizar los elementos ontológicos de OASys como parte de un proceso de ingeniería genérico. Los elementos proporcionados por la metodología son:

- Describir un proceso de ingeniería genérico de sistemas autónomos, conteniendo fases y tareas para el desarrollo de dichos sistemas.
- Sugerir técnicas y guías para ayudar en las tareas anteriores.
- Describir los productos obtenidos, como una clase de modelos (que puede constar de diferentes diagramas y especificaciones).
- Detallar los elementos de OASys a utilizar para obtener cada producto.

4.3 Metamodelado: el Uso de los Conceptos de OASys

La instanciación de conceptos (creando o identificando una entidad que se ajuste a la definición[GPHS06]), es el mecanismo habitual en metamodelado. No obstante, esta visión no tiene en cuenta los aspectos ontológicos implicados en modelado. Para resolverlo, es necesario considerar dos formas distintas de metamodelado [AK03]:

Metamodelado Lingüistíco: relaciona un concepto con su tipo *a través* de los niveles de metamodelado. Los conceptos se instancian utilizando relaciones *instancia de*, que expresan que un concepto se ha creado a partir de otro [AZW06]. Esta clase de instanciación se utiliza en OASys para describir un sistema autónomo concreto o objetos en la aplicación (Fig. 8.1).

Metamodelado Ontológico: relaciona un concepto con su tipo *en* un nivel. Los conceptos se instancian utilizando relaciones *es un, es una*, que expresan que un concepto es similar a un elemento ontológico en un nivel de la ontología [AZW06]. Esta forma de instanciación se ha utilizado con los conceptos del paquete de Perspectiva de ASys como especialización de los conceptos del paquete de Perspectiva a un nivel mayor de abstracción (Fig. 8.2).

La existencia de dos formas de metamodelado ha dado origen a determinados problemas [HSGP08], cuando se ha intentado satisfacer tanto la instanciación lingüistíca como la ontológica en las capas del modelo MOF de la OMG [Obj06a]. La capa superior M3 proporciona el metametamodelo. La capa M2 proporciona metamodelos que permiten describir la siguiente capa. La capa M1 contiene los modelos, por ejemplo definidos en UML. La capa M0 se utiliza para describir objetos reales. Las relaciones del tipo *es un, es una* pueden tener lugar dentro de una capa, mientras que las relaciones de instanciación del tipo *es instancia de* sólo se permiten entre las capas del MOF. Puede suceder entonces que un presunto metamodelo en el nivel M1 sea más un modelo ontológico (con conceptos del dominio) que un metamodelo real de nivel M2.

Para solucionarlo, para el metamodelado en la Metodología de Ingeniería basada en OASys, la ontología OASys jugará el papel de un modelo ontológico definido con UML en el nivel M1, heredando por tanto sus características del propio metamodelo UML en el nivel M2. Dentro del nivel M1, los diferentes elementos de los modelos de ingeniería se obtendrán como especialización o instanciación del tipo *es un, es una* dentro de un nivel, o del tipo *es instancia de* entre niveles (Fig. 8.3).

Las relaciones en los modelos tendrán el mismo nombre que en la relación original del paquete de OASys, lo que se conoce como sobrecarga del elemento [Gui05]. No obstante, la relación especializada lo es con un rango y dominio diferente, y por tanto con semántica también diferente.

4.4 Descripción de la Metodología

The Metodología de Ingeniería basada en OASys es una propuesta del desarrollo genérico de sistemas autónomos basado en los elementos ontológicos de OASys, centrándose en las fases de requisitos y análisis. La metodología indica las fases, tareas, y productos obtenidos tal y como se definieron en las Subontologías de Ingeniería de Sistemas e Ingeniería de ASys. Al basarse en OASys, la metodología también indica los paquetes de OASys a utilizar para la obtención de cada producto. Consta de dos fases, la de fase de Requisitos y la de Análisis. Puede considerarse también una fase inicial para definir las Vistas en el sistema autónomo.

Vistas del ASys: esta fase identifica las vistas a considerar en el desarrollo del sistema autónomos, a partir de los elementos del Paquete de Perspectivas, especializados en el Paquete de Perspectivas de ASys. Un Modelo de Vistas se puede obtener para cada una de las perspectivas consideradas.

Requisitos del ASys: esta fase identifica los requisitos considerando el PuntodeVistadeRequisitos, mediante técnicas de ingeniería tradicionales. Las diferentes tareas, subtareas y productos se han definido en el Paquete de Ingeniería ASys, y se detallan en la Figura 8.6.

CasosdeUsoDelSistema obtiene los modelos de casos de uso del sistema y subsistema, mediante la ejecución de dos subtareas: Modelado de Casos de Uso y Detallado de Casos de Uso.

- La Subtarea de Modelado de Casos de Uso obtiene un Modelo de Casos de Uso, para los casos de uso del sistema y subsistemas. La subtarea consiste en la definición del Sujeto, el Caso De Uso y los Actores del Caso de Uso de acuerdo con los conceptos y relaciones definidos en el Paquete de Requisitos de la Subontología de Ingeniería de Sistemas. Los modelos obtenidos pueden posteriormente definirse en una diagrama de casos de uso de UML o SysML.

- La Subtarea de Detallado de Casos de Uso produce una Especificación del Caso de Uso, como una tabla detallando los datos principales del mismo.

Caracterización de Requisitos define los requisitos del sistema autónomo, basándose en los conceptos de los Paquetes de Requisitos y de Requisitos ASys. Esta caracterización puede luego transformarse en tablas y diagramas de requisitos en SysML.

Análisis de ASys: el propósito de esta fase es describir el sistema autónomo considerando su estructura, comportamiento y función, como se detalla en la Figura 8.10:

- *Análisis Estructural* analiza el sistema autónomo desde un Punto De Vista Estructural, mediante diferentes subtareas. Los principales Productos obtenidos son el Modelo Estructural y el Modelo de Conocimiento, que pueden ser transformados en diagramas de clase de UML o SysML.

  La Subtarea de Modelado del Sistema se centra en modelar la estructura del sistema como subsistemas y elementos, obteniendo un Modelo Estructural. Dicho modelo consta de un Modelo de Estructura que define las relaciones mereológicas entre los subsistemas y elementos, utilizando y refinando los conceptos de los Paquetes de Teoría General de Sistemas y Mereología. Los modelos obtenidos pueden transformarse en diagramas de clases de UML o diagramas de definición de bloques en SysML.

  Una vez caracterizados los Subsistemas y Elementos, se describen las relaciones topológicas entre ellos mediante la utilización de los conceptos del Paquete de Topología. Se obtiene un Modelo de Topología que describe los enlaces físicos o de comunicación entre los componentes. Estas relaciones topológicas pueden posteriormente transformarse en un diagrama de bloques internos en SysML.

  Los distintos elementos identificados pueden detallarse durante la fase de diseño mediante la utilización de los elementos ontológicos del Paquete de Equipos, detallando si es un equipo computacional o físico.

  La Subtarea de Modelado del Conocimiento obtiene una serie de Modelos de Conocimiento, considerando los diferentes conceptos definidos en el Paquete de Conocimiento. Destaca el análisis de los Objetivos del sistema autónomo, identificando la Estructural de Objetivos mediante Objetivos Raíz, Objetivos Intermedios y Objetivos Locales en un Modelo de Estructura de Objetivos. Otros modelos se refieren a las Cantidades o Variables en el sistem, las Ontologías, y los Algoritmos utilizados. Los diferentes modelos pueden transformarse en diagramas de clases de UML o SysML.

- *Análisis de Comportamiento* es la Tarea que analiza el sistema autónomo desde el Punto de Vista del Comportamiento, mediante una Subtarea de Modelado del Comportamiento. La Tarea obtiene un Modelo de Comportamiento, que puede ser formalizado mediante diagramas de actividad, secuencia y máquinas de estado de UML o SysML. El Modelado de Comportamiento refina los conceptos del Paquete de Teoría General de Sistemas relativos al comportamiento.

- *Análisis Funcional* analiza el sistema desde el Punto de Vista Funcional, obteniendo un Modelo Funcional que consta de un Modelo de Agentes, un Modelo de Operaciones, y un Modelo de Responsabilidad (que pueden expresarse posteriormente mediante diagrams de actividades, secuencia y colaboración de UML o SysML). El Modelo Funcional especializa los conceptos de los Paquetes de Percepción, Pensamiento y Acción.

El Modelado de Funciones identifica los Actores y las Operaciones como especialización de los Paquetes de Percepción, Acción y Pensamiento. Un Modelo de Actores se obtiene identificando los Actores en el sistema autónomo, utilizando la parte correspondiente de conceptos del Paquete de Acción.

El Modelo de Operaciones captura las diferentes Operaciones que tienen lugar en el sistema autónomo, como especialización de la taxonomía de Operaciones recogida en el Paquete de Acción. Dentro de las Operaciones se incluyen las de Percepción (definida en el Paquete de Percepción), las Conceptuales que incluyen las Fases de Ciclo de Vida de Objetivos y de Reconfiguración de Objetivos (definidos en el Paquete de Pensamiento), y la operaciones de Grounding relativas a las Fases de Descomposición Funcional (definidas en el Paquete de Pensamiento). Por último, el Modelo de Responsabilidad captura la distribución de responsabilidades entre los Actores identificados.

Capítulo 5

APLICACIONES

5.1 Introducción

Este capítulo describe cómo la ontogía OASys se ha utilizado para caracterizar dos bancos de pruebas considerados en el programa de investigación ASys global: el sistema de control robótico y el sistema de control de procesos. Se han elegido para mostrar como OASys puede utilizarse para sistemas diferentes, que sin embargo comparten la característica de la autonomía.

5.2 El Sistema de Control Robótico

El Sistema de Control Robótico RCT en inglés es una aplicación robótica. El RCT consta de un sistema robótico formado por una plataforma y varios subsistemas interconectados que proporcionan un amplio rango de capacidades. El objetivo es dotarle de las capacidades cognitivas necesarias y de un sistema de control tal que permita la realización de tareas complejas. Su descripción se proporciona en [CH08], [HH08]y[San07].

5.2.1 Descripción Estructural

El Sistema de Control Robótico consta de tres subsistemas principales: la plataforma base, los sistemas de a bordoy los sistemas de soporte

1. Plataforma Base

   Dicha plataforma es un robot móvil Pioneer 2–AT8 (Fig. 9.1), que incluye todos los elementos necesarios para su control y navegación, especialmente en el exterior. Se denomina Higgs como nombre coloquial en el grupo de trabajo. Como elementos hardware, consta de un panel, un cuerpo, un panel de control, sensores, actuadores, parachoques y sistema de alimentación. Como elementos software, contiene un sistema operativo, una interface de programacióny varios módulos software desarrollados por ASLab (comunicación [Par04], emociones [Con05], integración SOAR

[Sán05], voz [Pan05], servidor de control RT–CORBA [Her05], operaciones remotas basadas en CORBA [Con07]y reconocimiento de superficies [God07]).

2. Sistemas de a bordo

   En la plataforma base se han añadido diferentes equipos para ampliar el rango de funciones del robot, tales como un ordenador, un laser, una muñeca para orientar la cámara, una cámara, una radio, un ordenador portátily un sistema GPS.

3. Sistemas de Apoyo Además de los anteriores, el sistema utiliza diferentes sistemas de apoyo, tales como la red inalámbrica, un control remoto, un servidor y una terminal de usuario.

5.2.2 Descripción Funcional

La descripción funcional del Sistema de Control Robótico se ha hecho mediante la especificación de requisitos en casos de usos. Existen requisitos funcionales, tal y como se ha definido en OASys, que especifican las operaciones que el sistema debe ejecutar. En el Sistema se han considerado requisitos principales (navegación y supervivencia)y requisitos secundarios (exploración, esquivar obstáculos, movimiento de reacción, e identificación de objetos).

5.3 Metodología Aplicada al RCT

La Metodología se ha utilizado para el análisis conceptual del Sistema de Control Robótico. Se describen a continuación la realización de las tareasy la descripción de los modelos obtenidos en los que los términos en cursiva son los definidos en la ontología OASys.

5.3.1 Fase de Vistas del RCT

El propósito de esta *Fase* es el de identificar las diferentes vistas de interes para el RCT, considerando los elementos en el Paquete de Perspectivas, especializados con los del Paquete de Perspectivas de ASys.

Por ejemplo, el *Punto de Vista de Requisitos* especializa el concepto del *Punto de Vista* para enfocar el análisis del RCT mediante un *Enfoque* de *Requisito ASys*. Considerando este punto de vista, un Modelo de Punto de Vista de Requisitos para el RCT se instancia (Fig. 9.3). Los distintos elementos se eligen de los Paquetes de Perspectiva y de Perspectiva ASys. Por ejemplo, el concepto Requisito de RCT es un *Requisito ASys*. De forma similar, el Punto de Vista de Rendimiento del RCT es un *Punto de Vista*, tal y como se definió en el Paquete de Perspectiva.

El *Punto de Vista Estructural* se centra en la estructura del sistema autónomo, considerando un enfoque de *Estructura del ASys.* Un Modelo de Punto de Vista Estructural del RCT se obtiene (Fig. 9.3), instanciando los diferentes elementos de los Paquetes de Perspectiva y de Perspectiva ASys. En este modelo, la Vista Estructural del RCT es una *Vista* que consta de un Modelo Estructural del RCT, como *Modelo de Ingeniería* obtenido como resultado de la Tarea de Análisis Estructural.

El *Punto de Vista de Comportamiento* describe el comportamiento del sistema. El Modelo de Punto de Vista de Comportamiento del RCT obtenido (Fig. 9.5) instancia los conceptos y relaciones de los Paquetes de Perspectiva y Perspectiva ASys.

El *Punto de Vista Funcional* se centra en el análisis de las funciones del sistema autónomo, como comportamientos esperados o deseados. El Modelo de Punto de Vista Estructural del RCT (Fig. 9.6) instancia los elementos de los Paquetes de Perspectiva y Perspectiva ASys.

Durante el desarrollo del RCT, se necesita considerar el *Punto de Vista de Rendimiento*, para evaluar el rendimiento del RCT una vez implementado. El Punto de Vista de Rendimiento utiliza los elementos de los Paquetes correspondientes para centrarse en los aspectos de rendimiento del RCT, en forma de un Modelo de Punto de Vista de Rendimiento para el RCT (Fig. 9.7).

5.3.2 Fase de Requisitos del RCT

Esta fase identifica los requisitos de los usuarios para el RCT, considerando el Punto de Vista de Requisitos.

- Tarea de Casos de Uso del Sistema

  La primera *Tarea* es la de analizar los *Casos de Uso del Sistema* para el RCT, instanciado los elementos del Paquete de Requisitos de la Subontología de Ingeniería de Sistemas para el Sistema de Control Robótico (Fig. 9.8). Esta *Tarea* consta de las *Subtareas* de *Modelo de Casos de Uso* y *Detallado de Casos de Uso* para el RCT.

  Un *Caso de Uso* en OASys se ha definido como un medio para capturar los requisitos del sistema, considerando el Paquete de Requisitos. Para definir el Caso de Uso, se identifican el*Sujeto* como el sistema bajo análisisy los diferentes *Actores de Caso de Uso* como objetos que interactúan con el sistema. Como resultado, se obtiene un *Modelo de Casos de Uso del Sistema* detallando los conceptos identificados. Las clases UML en el diagrama son instanciación de los conceptos originales en OASys, especificándose la instanciación mediante los nombres de los roles en las relaciones del diagrama.

  Cuando el *Sujeto* en estudio es el Sistema de Control Robótico, un *Modelo de Casos de Uso* muestra los requisitos del RCT por medio de casos de uso. Los requisitos de un sistema puede ser de diferentes tipos (físico, funcional, de rendimiento, interfaz, diseño) como se define en el Paquete de Requisitos. Un análisis de las necesidades iniciales, identificó los *Requisitos Funcionales* del RCT, que ha sido definido como un requisito que especifica una acción o conducta que debe realizar un sistema. *Requisitos Funcionales* principales para el RCT es la Navegación y la Supervivencia.

El requisito de Navegación se captura por medio del *Caso de Uso* de Navegación, que incluye el *Requisitio Funcional* de ser capaz de explorar el entorno, identificar los elementos en el entornoy para evitar obstáculos. Estos requisitos se recogen en los *Casos de Uso* de Exploración del Entorno, Identificación, Evitar Obstáculos (Figura 9.9). A su vez, el *Requisito Funcional* de supervivencia es capturado en *Caso de Uso* de Supervivencia, que incluye los de Fallo de Subsistema y el de Recarga (Fig. 9.10).

Podría ser interesante detallar un *Caso de Uso* particular para los Subsistemas. Centrándose, por ejemplo, en el requisito de navegación, se pueden obtener los *Casos de Uso* para los subsistemas, en la forma de un *Modelo de Casos de Uso* de Subsistemas (Figura 9.11).

La *Subtarea de Modelado de Casos de Uso* también puede obtener un *Modelo de Casos de Uso del Subsistema*, que reúne a los *Requisitos* para un determinado *Subsistema.* En el caso del RCT, este Caso de Uso se obtuvo para la plataforma base (figura 9.12, que muestra la creación de instancias de los elementos originales ontológicos para detallar el *Requisito Funcional* del movimiento de la plataforma; el *Requisito Funcional* de gestión de comunicaciones (Figura 9.13) y la gestión de señales sensoriales de entrada para el sistema de apoyo (Figura 9.14).

La siguiente *Subtarea* en la *Fase de Requisito ASys* es la de detallar cada *Modelo de Casos de Uso* obtenido en la *Subtarea de Detallado de Casos de Uso.* Por ejemplo, se detalló el requisito de Navegación tal y como aparece en la Fig. 9.15 y el de Supervivencia en la Fig. 9.16.

- Caracterización de Requisitos

  Una *Tarea* adicional de esta *Fase* es la de caracterizar los requisitos del sistema autónomo, para obtener la *Caracterización de Requisitos*, definidos en el Paquete de Ingeniería de Procesos ASys de la Subontología de Ingeniería ASys.

5.3.3 Fase de Análisis del RCT

Como parte de la Fase de Análisis se consideran diferentes *Tareas*, como el *Análisis Estructural*, el *Análisis de Comportamiento*y el *Análisis Funcional.*

Análisis Estructural

La *Tarea de Análisis* del RCT (Fig. 9.17) se centra en el *Punto de Vista Estructural*, es decir, analizando la estructura del sistema. Como un resultado, un *Modelo Estructural* para el RCT, que consta de un *Modelo de Estructura* y un *Modelo de Topología.*

- Modelado del Sistema

  La *Subtarea de Modelado de Sistema* para el RCT obtuvo el *Modelo Estructural*, que consta de un *Modelo de Estructura* y el *Modelo de Topología.* Para el*Modelo de Estructura* del RCT, se identificaron tres Subsistemas diferentes: la plataforma, los sistemas añadidos a la plataformay los sistemas de apoyo (Fig. 9.18).

  El *Modelo de Topología* representa las conexiones topológicas entre las partes del sistema, refinando los conceptos del Paquete de Topología. Para el Sistema de Control Robótico, se consideró intersante distinguir entre la topología física y de comunicación WiFi, Ethernet, etc. Por tanto, se obtuvieron dos Modelos de Topología para el RCT: el Modelo de Topología de Comunicaciones (Fig. 9.19)y el Modelo de Topología Física (Fig. 9.20).

  Cada uno de los subsistemas y elementos identificados en el*Modelo de Estructura*, puede detallarse durante el diseño utilizando los elementos del Paquete de Equipos en forma de un *Modelo de Equipos.*

5.4 Sistema de Control de Procesos

El Sistema de Control de Procesos (PCT en inglés) es el sistema químico con reactor diseñado para probar las ideas de sistemas autónomos para sistemas de procesos. Su componente principal es un tanque reactor continuamente agitado CSTR en inglés, así como un sistema de control y un sistema de instrumentación para medidas. El objetivo es el de dotar al sistema de capacidades cognitivas que le permitan realizar tareas complejas tales como diagnosis de fallos, gestión de alarmasy reconfiguración del sistema de control. El banco de pruebas se ha descrito en [dlM09b], [dlM09a], and [Rod07].

5.4.1 Descripción Estructural

El CSTR se compone de un reactor, con una camisa para refrigerar o calentar. Consta también de una unidad de agitación, movida con un motor eléctrico.

Subsistema hidraúlico: se compone de diferentes tubos para alimentar los reactivos, extraer los productos y circular el vapor, así como bombas y válvulas.

Instrumentación: consta de varios sensores (uno de pH, otro de temperatura y otro de presión). Además hay un sensor ORP, un analizador electroquímico y dos caudalímetros electromagnéticos.

Sistema de control cuyo elemento principal es un PC Dell, con varios módulos entrada–salida de corriente y tensión analógicos.

5.4.2 Descripción Funcional

El sistema consta del reactor CSTR, donde dos corrientes de reactivos se alimentan mediante bombasy otra corriente de salida. Las reacciones químicas tienen lugar en el reactor, que debe ser enfríado o calentado con vapor, en función de la reacción. La unidad de agitación asegura una composición homogénea en el reactor. Existe también una válvula de alivio. Existen diferentes estrategias de control en el sistema: control de temperatura, de nivel, de presión, de composición y de caudal de entrada.

5.5 Aplicación de la Metodología Basada en OASys al PCT

Se ha aplicado la Metodología durante las fases de Análisis del PCT, en el que los diferentes aspectos del sistema se han represetado utilizando los elementos ontológicos de OASys. Se han obtenido diferentes Productos, principalmente modelos en las explicaciones de los mismos en secciones siguientes, los términos de OASys aparecen en cursiva.

5.5.1 Fase de Vistas del PCT

El objetivo de esta *Fase* es identificar los diferentes puntos de vista, o vistas, de interés en el PCT considerando los elementos del Paquete de Perspectiva, especializados a través del Paquete de Perspectiva de ASys.

Por ejemplo, el *Punto de Vista de Requisitos* especializa el concepto *Punto de Vista* para ejecutar el análisis del PCT considerando el *Enfoque* de *Requisito ASys.* Considerando este punto de vista, un PCT Modelo de Punto de Vista de Requisitos puede obtenerse (Fig. 9.25). En este modelo, el Punto de Vista de Requisito del PCT es una *Vista* que constará de un Modelo de Casos de Uso del PCT, como un *Modelo de Ingeneiría* que se obtiene como Producto de la *SubtareaModelado de Casos de Uso* .

El *Punto de Vista Estructural* presta atención a la estructura del sistema autónomo, considerando el enfoque de *Estructura del ASys.* Un Modelo de Punto de Vista Estructural para el PCT puede obtenerse (Fig. 9.26) como instanción de los conceptos de los Paquetes de Perspectiva y Perspectiva ASys.

El *Punto de Vista Funcional* se centra en el análisis de las funciones del PCT, que se representan en el Modelo de Punto de Vista Funcional del PCT (Fig. 9.27) que instancia los elementos de los Paquetes de Perspectiva y de Perspectiva ASys.

5.5.2 Fase de Requisitos del PCT

El objetivo de esta fase es el de identificar los requisitos para el PCT, mediante técnicas habituales de ingeniería. Dentro de esta *Fase*, la *Tarea de Casos de Uso del Sistema* se dedica al desarrollo de los *Casos de Uso*s, durante las *Subtareas* de *Modelado de Casos de Uso* y *Detallado de Casos de Uso* (Fig. 9.28).

- Modelado de Casos de Uso

  El *Modelo de Casos de Uso* del PCT (Fig. 9.29) se obtiene como Producto en la was obtained as a work product in the PCT *Subtarea de Modelado de Casos de Uso*, considerando los elementos ontológicos del Paquete de Requisitos de la Subontología de Ingenería de Sistemas.

  Algunos casos de uso se han analizado en detalle, obteniendo como resultado el de Arranque de Planta fig. 9.30), Parada de Planta (Fig. 9.31)y Operación de Planta (Fig. 9.32).

- Detallado de Casos de Uso

  Cada *Modelo de Casos de Uso* ha sido detallado durante la *Subtarea de Detallado de Casos de Uso*, utilizando un patrón de Casos de Uso desarrollado por ASLab.

5.5.3 Fase de Análisis del PCT

Análisis Estructural

El *Análisis Estructural* del PCT tiene como objetivo el análisis del sistema considerando los aspectos estructurales, bajo el *Punto de Vista Estructural* (Fig. 9.35). Diferentes *Modelos de Ingeniería* pueden obtenerse como resultado de las dos *Subtareas* principales: *Modelado de Sistema* y*Modelado de Conocimiento.*

- Modelado de Sistema

  El Modelo Estructural del PCT es una clase de modelo que describe el PCT desde un punto de vista estructural. En este caso, el *Modelo Estructural* consta de diferentes*Modelos de Estructura* para describir los elementos del banco de pruebas, así como el *Modelo de Topología* para describir las relaciones topológicos entre los componentes. El Modelo de Estructura especializa los elementos ontológicos de los Paquetes de Teoría General de Sistemas y Mereología, para describir las características estructurales del banco de pruebas. El Modelo de Topología refina los conceptos del Paquete de Topología.

  El Modelo de Estructura del PCT muestra que el *Sistema* consta de *Subsistemas*, que pueden descomponerse hasta llegar a sus *Elementos.* El Sistema de Control de Procesos consta del Reactor CSTR, el Subsistema Hidraúlico, el Sistema de Control y la Instrumentación (Fig. 9.36).

Se obtienen *Modelos de Estructura* cuando un *Subsistema* como el Reactor CSTR (Fig. 9.37), el Sistema Hidraúlico (Fig. 9.38), o el de Instrumentación (Fig. 9.39) se analiza.

Capítulo 6

CONCLUSIONES Y DESARROLLOS FUTUROS

La motivación de la investigación, el alcance y objetivos se describieron en el Capítulo Introducción. La explicación sobre cómo los objetivos se han logrado finalmente por el desarrollo de la Ontología de Sistemas Autónomos (OASys), y su Metodología correspondiente se da en este capítulo. Las limitaciones de la investigación desarrollada para satisfacer las necesidades iniciales también son analizadas. Como resultado, se sugiere una propuesta de líneas de investigación identificadas para seguir desarrollando la investigaci'on.

6.1 Revisión de la Investigacion

El objetivo general de la investigación es el de capturar y explotar los conceptos para la descripción y el proceso de ingeniería de cualquier sistema autónomo, como parte del programa de investigación ASys que aborda el desarrollo de una tecnología para la ingeniería de sistemas autónomos, independientemente de su aplicación particular. Con este propósito, dos elementos han sido considerados y desarrollados [BASRH10b]:

1. *Una ontología para el dominio de sistemas autónomos OASys*

   OASys fue inicialmente concebida con una función similar a los encontrados en la literatura, es decir, una ontología de dominio como mecanismo de representaciónen durante el análisis y diseño. Por lo tanto, OASys ha sido utilizada para llevar a cabo **una conceptualización ontológica del dominio de los sistemas autónomos**.

   OASys se ha desarrollado de acuerdo a un amplio conjunto de requisitos que se han cumplido con las siguientes características:

   Estructura OASys ha sido organizada de acuerdo con dos dimensiones ortogonales: una compuesta por los diferentes niveles de abstracción, la otra consiste en la separación entre el análisis del sistema y sus procesos de ingeniería.

Para hacer frente a los diferentes niveles de abstracción, OASys ha adoptado una estructura de capas para separar el conocimiento general de todo uno, por lo tanto teniendo en cuenta los diferentes niveles de abstracción para conceptualizar el dominio de los sistemas autónomos. El nivel más abstracto, es la capa de dominio del sistema, que contiene los elementos ontológica acerca de una caracterización general del sistema y de ingeniería. Una segunda capa, la capa de dominio Asys recoge la ontológica construye para describir y charaterise la estructura del sistema autónomo ", la función y el comportamiento. El enfoque está en los sistemas autónomos como conceptualizado en el marco del proyecto ASys, no obstante ser suficientemente generales como para ser re-utilizado en el desarrollo de cualquier sistema autónomo de otros.

Para separar entre el análisis del sistema y su proceso de ingeniería, OASys ha sido dividida en dos ontologías. La Ontología de Asys reúne los elementos para el análisis ontológico de un sistema autónomo. Asimismo, la Ontología de Ingeniería de ASys recoge los conceptos y relaciones para describir el proceso de ingeniería de un sistema autónomo.

Criterios de diseño A fin de garantizar su coherencia y calidad, OASys ha sido desarrollada teniendo en cuenta los siguientes criterios de diseño

- Posibilidad de ampliación: Una cuestión importante que debía abordarse en OASys es la posibilidad de su ampliación y modificación. Para apoyar esta función, los conceptos dentro de una capa se clasificaron mediante el uso de dos elementos diferentes: subontologías y paquetes. Las subontologías se han considerado, para contener conceptos relacionados con la caracterización y la ingeniería de sistemas autónomos. Los paquetes dentro de cada subontología fueron definidos para recoger los conceptos relacionados con un aspecto concreto.
- Claridad: Ontologías y glosarios se han revisados para incluir sus términos en OASys con el objetivo de la utilización de conceptos que han existido y se ha utilizado durante mucho tiempo en la comunidad científica. Los documentos disponibles fueron cuidadosamente analizados para extraer los elementos ontológicos, la comprobación de los desajustes o características comunes. La integración de las terminologías, diccionarios y ontologías existentes ha prestado especial atención a la granularidad de su contenido. Tales conceptos se han analizado más tarde con los miembros del grupo, para asegurar el significado deseado para nuestra investigación. Por último, todas las construcciones ontológicas (conceptos, relaciones, atributos, axiomas) se han definido en el lenguaje natural.
- Mínimo sesgo por el lenguaje de implementación: Para evitar la conceptualización errónea de conceptos basados en la sintaxis del lenguaje de impletación final, representaciones tabulares y gráficos intermedios se han utilizado para definir las construcciones ontológicas.

- Mínimo compromiso ontológico: En cada capa sólo los conceptos fundamentales han sido descritos. Las capas inferiores aportan más detalles sobre estos conceptos, añadiendo un mayor nivel de detalle. Nuevos conceptos se añadirán a medida que nuevas aplicaciones se desarrollen utilizando OASys.
- Normalización: la convención para la nomenclatura de los elementos de la ontología se ha elegido para hacer más fácil de entender y evitar algunos errores comunes de modelado.

Desarrollo basado en METHONTOLOGY La investigación no tenía como objetivo desarrollar una metodología nueva, sino a la reutilización de un bien establecida entre las varias metodologías disponibles. METHONTOLOGY ha sido elegida como punto de partida, ya que propone tanto un proceso de desarrollo de la ontología, así como un ciclo de vida de la ontología estrechamente entrelazados [FLGPJ97] . El proceso de desarrollo se refiere a las actividades que se realizan cuando la construcción de la ontología. El ciclo de vida de la ontología identifica cuando estas actividades deben llevarse a cabo, por un conjunto de etapas que definir las actividades a realizar en cada etapa y cómo se relacionan. Tanto el proceso de desarrollo y las actividades del ciclo de vida fueron inicialmente seguido para el desarrollo de OASYS. Otras indicaciones adicionales descritas en [NM01], [Miz04], [BA07] también han sido consideradas.

Formalización La ontología se ha formalizado mediante un general de ingeniería de software y lenguaje de propósito específico, como UML [Obj09a, Obj09b]. UML no es un lenguaje de desarrollo de la ontología, por lo tanto, se ha considerado que algunos de los inconvenientes para la ingeniería de la ontología: ontologías ligeras [GPFLC04b] , semántica incompleta y el patrimonio software [Obj09c] , la falta de mecanismos de inferencia [GDD06d]. Sus deficiencias se han abordado en el metamodelo de ontología Definition ODM [Obj09c] que define metamodelos, mapas y perfiles para permitir la interacción entre UML y los lenguajes tradicionales ontológicos como OWL [SWM04].

2. *Una metodología basada en OASys*

La Metodología basada en OASys es una propuesta de una metodología para un proceso de ingeniería genérico de sistemas autónomos, basándose en los elementos de OASys. La metodología provee algunas **directrices y un ejemplo de la aplicación de los elementos ontológicos de OASys como apoyo semántico en un proceso de ingeniería genérico de sistemas autónomos**. La metodología ha propuesto:

- El proceso de ingeniería genérico de sistemas autónomos, que contiene fases y tareas para llevar a cabo el desarrollo del sistema autónomo.
- Las técnicas y guías para ayudar a las tareas y subtareas.

- Los productos de trabajo que se obtienenn como una clase de modelos.
- Los elementos relacionados con OASys para ser utilizados en los productos de trabajo.

6.2 Principales Contribuciones de la Investigación

OASys ha reunido, conceptualizado y analizado cómo los ingenieros de desarrollo de los sistemas autónomos describen los diferentes elementos que participan en la operación de un sistema autónomo. OASys ha atendido a los principales aspectos de la ingeniería de un sistema autónomo: el proceso de percepción, el conocimiento utilizado, los objetivos del sistema, el proceso de pensamiento como descomposición funcional, y los actores que llevan a cabo las acciones del sistema.

OASys ha examinado el dominio de los sistemas autónomos con una visión global. Anteriores ontologías para este tipo de sistemas se centraron en un enfoque limitado, ya sea que los robots móviles o sistemas basados en agentes. El enfoque aquí ha sido el de definir los diferentes elementos para describir la estructura del sistema, y su función de una manera lo suficientemente general como para ser reutilizado entre diferentes aplicaciones. OASys puede complementarse con ontologías adicionales sobre un subdominio o una aplicación concretos, sin perder sus características de reutilización y generalidad.

La ontología OASys y su metodología han proporcionado varias ventajas:

- OASys ha proporcionado una conceptualización común del dominio de los sistemas autónomos para los desarrolladores de sistemas autónomos.
- OASys permite la gestión de una amplia gama de conceptos, evitando las definiciones imprecisas y los desajustes cuando se refiere a los sistemas autónomos.
- La separación entre la descripción y los aspectos de ingeniería en un sistema, ha permitido abordar de forma independiente el desarrollo de las aplicaciones ejemplo.
- La conceptualización de las diferentes aplicaciones ejemplo ha permitido definirlas de manera independiente, identificando aspectos comunes entre ellas. Las propiedades particulares para cada aplicación han sido identificadas, lo que apoya la idea de utilizar ontologías de subdominio y aplicación ya existentes, como parte de los elementos de soporte en la ingeniería.
- La existencia de diferentes niveles de abstracción dentro de la ontología ha demostrado su validez para describir un sistema autónomo, mediante el uso de las técnicas de focalización.
- OASys mejora la interoperabilidad de las aplicaciones, ya que contienen un núcleo de vocabulario común.
- OASys sirve como modelo ontológico para la elaboración de modelos conceptuales de un sistema autónomo.

- La Metodología ha indicado cómo los sistemas autónomos pueden ser caracterizados y deben ser diseñados con modelos conceptuales, en forma de un proceso de ingeniería genérico para este tipo de sistemas.
- El modelado conceptual del proceso ofrece mejoras en el intercambio de conocimientos y la reutilización entre los desarrolladores de aplicaciones durante la fase de análisis.
- OASys y la metodología han contribuido a aumentar la fiabilidad de los modelos de software obtenidos.

6.3 Futuros Desarrollos

Durante la ingeniería ontológica de OASys y su metodología, algunos elementos han aparecido que indican la necesidad de un ulterior desarrollo.

- OASys y ontologías de dominio

  OASys ha permitido conceptualizar el dominio de los sistemas autónomos de un modo general suficiente, y reutilizable, para abordar cualquier tipo de sistema autónomo. Sin embargo, las propiedades particulares de diferentes aplicaciones de investigación han identificado la necesidad de complementar OASys con ontologías de dominio, mediante la adición de las diferencias de subdominios o aplicaciones. Para hacer frente a un subdominio o una concretas, se deben investigar ontologías de dominio adicionales. Un análisis exhaustivo de estas ontologías de dominio decidirá su posible integración con OASys, o sugerir nuevas ampliaciones de OASys en forma de paquetes adicionales o incluso una capa orientada a aplicación.

- OASys y la definición de puntos de vista de ingeniería

  El programa de investigación en sistemas autónomos prevé la utilización de puntos de vista diferentes para definir un sistema autónomo. Esta necesidad ya ha sido examinado en la Ontología de Ingeniería ASys incluyendo el Paquete de Perspectivas, que está especializado en el Paquete de Perspectivas de ASys, para representar a la caracterización de un sistema autónomo.

  La consideración de los puntos de vista se relaciona estrechamente con los diferentes modelos a desarrollar, y que serán utilizados por un sistema autónomo. Para hacer frente a las diferentes vistas seleccionadas, podría ser necesario dividir la definición y el uso de los distintos conceptos en OASys en relación con estos puntos de vista diferentes (estructurales, de comportamiento, funcionales, etc.).

- OASys, su evaluación y mantenimiento

OASys se ha desarrollado siguiendo una metodología y considerando los criterios de diseño. Sin embargo, la ingeniería ontológica es una tarea no exenta de problemas. La investigación ha prestado especial atención a las actividades de desarrollo de la especificación, y la conceptualización. También se abordaron los aspectos de la adquisición de conocimientos y la integración de las fuentes. Sin embargo, las actividades de apoyo como la evaluación y el mantenimiento de la ontología se deben considerar en el uso futuro y el desarrollo de OASYS.

- La extensión de la metodología de ingeniería de sistemas autónomos

  La metodología descrita en esta investigación es una propuesta sobre cómo seguir un proceso de ingeniería genérico basado en un ontología, para sugerir y orientar sobre la aplicación de los elementos ontológicos de OASys. No obstante, la fase actual del programa de investigación global ASys ha hecho difícil llegar a una conceptualización clara sobre cómo se lleva a cabo el proceso de ingeniería de sistemas autónomos. Dicho proceso de ingeniería aún no se ha descrito, y por lo tanto no ha sido plenamente conceptualizado como parte de la Subontología de Ingeniería de ASys de OASys.

- Metodología para la modelización de sistemas autónomos

  La metodología propuesta utiliza OASys y los compromisos ontológicos subyacentes para crear los diferentes modelos, asegurando un significado común y evitando los desajustes conceptuales durante el desarrollo de un sistema autónomo. La construcción de los modelos conceptuales obstante, ha sido para las aplicaciones concretas mediante la instanciación y especialización de los conceptos.

  Una metodología adecuada basada en la ontología y patrones de software con la ayuda de herramientas de modelización conceptual es un paso más para enriquecer estos modelos conceptuales. Este proceso de refinamiento de los conceptos para hacer frente a mayores niveles de información, así como el uso de elementos ontológicos se ha de definir en la Metodología de Modelización de ASys, como una etapa siguiente en el programa general de investigación de sistemas autónomos. Como parte de esta metodología de modelización, es necesario definir entre otros aspectos: cómo se selecciona un concepto, cómo integrar el concepto en un modelo, cómo establecer e importar sus relaciones con otros conceptos, y como detallarlo añadiendo sus atributos como parte del desarrollo de un modelo concreto.

- Ingeniería basada en modelos y ontologías

  OASys y otros elementos adicionales se van a utilizar como soporte software para obtener los modelos del sistema autónomo. OASys como una ontología de dominio de los sistemas autónomos puede desempeñar un papel importante cuando se utiliza la Arquitectura Basada en Modelos (MDA en inglés).

6.4 Conclusiones

Para que un sistema autónomo se comporte adecuadamente en un entorno incierto, debe tener algún tipo de representación de lo que siente y las experiencias que percibe como entidades, eventos, y situaciones en el mundo. También sería deseable que el sistema tuviera un modelo interno que represente lo que sabe y aprende, así como un mecanismo para calcular los valores y prioridades que le permite decidir qué objetivos y metas cumplir.

Un reto importante en los sistemas autónomos es la capacidad de mantener las representaciones actualizadas, tanto del sistema como el entorno en el que opera. La incapacidad de hacer esto podría retrasar la la planificación y la ejecución de las tareas de forma eficaz.

Además, los conceptos utilizados en el desarrollo de los sistemas autónomos serán de utilidad para los propios sistemas para poder pensar en su funcionamiento interno. Los conceptos son utilizados por un agente para pensar, porque proporcionan la estructura y los métodos de uso de los contenidos mentales. Los conceptos se utilizan para percibir la realidad coincidente con la señal obtenida a través de sensores para estos conceptos. Los conceptos, en forma de ontologías se utilizan para la interacción cognitiva con otros agentes. Los conceptos se utilizan por el ingeniero para pensar en el sistema en desarrollo. Esos conceptos se pueden poner dentro del sistema cuando se trata de un sistema cognitivo.

OASys amplía el papel tradicional de conceptualización común, para convertirse en una ontología que se utilizará en tiempo de ejecución como un componente adicional cooperando con el sistema autónomo. OASys pretende dotar al sistema autónomo con los mismos conceptos de diseño como conceptos en tiempo de funcionamiento, en forma de modelos. El objetivo es que cualquier sistema autónomo pueda utilizar los conocimientos de diseño durante su funcionamiento, capturado en un formato legible por máquina, tal y como una ontología: un conjunto de conceptos para ser utilizados por los ingenieros en la descripción y construcción del sistema y por el propio sistema en su funcionamiento. El sistema autónomo utilizará modelos basados en la ontología desarrollada en esta tesis [SHH$^+$09], como parte de una estrategía de ingeniería basada en modelos [SHG$^+$09]. Otra línea de investigación es cómo los modelos conceptuales para el sistema autónomo será generados a partir de la ontología de dominio OASys, mediante un posible uso de herramientas de modelización conceptual, complementadas con patrones predefinidos.

Un sistema autónomo operará utilizando los modelos de su entorno, de su tarea y de sí mismo, como en el paradigma de control basado en modelos, donde los modelos serán los mismos utilizados por los ingenieros para construir el sistema. En última instancia, proporcionará al sistema de autocapacidades de ingeniería necesarias para una robusta autonomía robusta [SLH07]. Un paso más será para los agentes el poder interpretar los modelos basados en OASys. Las ontologías se han usado ampliamente en la comunidad de agentes [BAS06], pero aún está por investigar la forma en la que la ontología y los modelos basados en ella pueden ser usados para generar el significado de los conceptos para el proceso de toma de decisiones [SBE$^+$03], [SBLG08].

English Version

ABSTRACT

The framework of this research is a long-term research project on autonomous systems (ASys), which addresses a universal technology for the development of all classes of autonomous systems, regardless of its particular application domain. Autonomous systems refer to systems capable of operating in a real-world environment without any form of external control for extended periods of time. The core strategy is to exploit cognitive control loops, using knowledge captured as different models, based on the ontology for autonomous systems (OASys) developed in this research.

An ontology is a formal, explicit specification of a shared conceptualisation, i.e., a machine-readable abstract model where relevant concepts and relationships are identified and explicitly defined by consensus in a group. Ontologies have been widely applied for software engineering. They have also been used as representational mechanisms based on a computational language, to clarify and share the domain knowledge.

OASys captures and exploits the concepts to support the description and the engineering process of any autonomous system, as a domain-ontology structured in two levels of abstraction, innerly organised into Subontologies and Packages. The METHONTOLOGY methodology has been used as design method, with a final implementation in UML.

The autonomous system´s description has been formalised in the ASys Ontology, divided into the System Subontology and the ASys Subontology. The System Subontology contains the elements to define any system, consisting of: the General Systems Package to characterise any system's structure and behaviour based on the General Systems Theory; the Mereology Package with taxonomies of the whole-part concepts and relationships; the Topology Package for topological connections.

The ASys Subontology specialises the former concepts for autonomous systems: the Perception Package to describe the perceptive and sensing processes; the Knowledge Package to characterise the different kinds of knowledge used; the Thought Package with task-oriented processes concepts; the Action Package about the operations and performing actors, and the Device Package to detail the devices.

The autonomous system´s engineering process has been formalised in the ASys Engineering Ontology, structured in the System Engineering Subontology and the ASys Engineering Subontology.

The System Engineering Subontology gathers the concepts related to an engineering process, based on different metamodels and specifications: the Requirement Package to specify system's requirements; the Perspective Package with viewpoints in a system development; the Engineering Process Package to describe an engineering development process; and the Model-Driven Package to conceptualise the model-driven engineering approach.

The ASys Engineering Subontology contains the specialisation and additional ontological elements to describe an autonomous system's generic engineering process: the ASys Requirement Package to characterise process and system quality requirements; the ASys Perspective Package to design an autonomous system from different aspects; the ASys Engineering Process Package to describe a generic autonomous system engineering process.

OASys has been complemented with the development of the OASys-based methodology to exemplify the use of OASys in a generic autonomous system engineering process. OASys has been applied in the Robot Control Testbed (RCT), and the Process Control Testbed (PCT). RCT is a collection of mobile robot systems, with a wide range of implementations and capabilities (from conventional SLAM based mobile robots to virtual ones inspired in rat brain neuroscience). PCT involves the development of a robust control architecture for a chemical reaction system (with multiple steady states), providing the system with cognitive capabilities to carry out complex tasks such as fault diagnosis, alarm management, and control system reconfiguration from a single theoretical standpoint.

OASys offers improvements on the knowledge sharing and reuse between developers. The separation between the description and the engineering process defines independently the characterisation and the engineering process of the testbeds systems. The common conceptualisation of rather different testbeds allows to identifying commonalities in a domain-independent way. Particular properties for each one support the necessity of domain ontologies as an additional component. The different levels of abstraction describe the testbeds from a domain focalisation viewpoint, yet the refinement process and the ontological elements usage has to be further addressed.

Part I

Background

Chapter 1

INTRODUCTION

1.1 Motivation

There are many reasons to provide systems with increased autonomy. They range from cost reduction and efficiency to improved performance and fault tolerance. Of all these reasons one has recently gained in importance: *complexity.*

The need for augmented dependability of complex systems or for improved management of the increasing complexity of these systems has established a new level of requirements for autonomy. Beyond the classical methods of automatic control that address disturbance rejection in automatic systems, modern technology for autonomy shall address open-ended environments where the characterisation of a concrete and reduced set of disturbances is far from possible. Modern autonomous systems technology is in search of new methods for generalised autonomy in an attempt to overcome wider and less constrained operational conditions[1].

Most engineering domains require a certain degree of autonomy in their systems, *i.e.*, it is requested from systems to be capable of acting with bounded and made–to–fit autonomy. The level of intelligence and autonomy required will depend on the task and the uncertainty affecting its performance. The movement towards extended operational requirements and open–ended environments imply that such systems will be required to accomplish certain high-level cognitive processes such as automatic re–design, social reasoning or self–awareness [UIT06d].

The many challenges to describe and to build such autonomous systems notwithstanding, it is firstly necessary to clarify the very issue at hand: *what is autonomy.* The concept of autonomous system is suffering a concept evolution and domain focusing problem.

In an initial analysis, an autonomous system is regarded as either a battery-driven de-

[1]A paradigmatical example is the *autonomic computing* movement. Autonomic computing systems intend simplexity by delegating their configuration, optimisation, and repair to the systems themselves [KC03].

vice, an autonomous mobile robot or some kind of software–based agent. In the domain of biosystems, however, autonomy implies certain kind of self-construction —autopoiesis. Recent research in computerised systems has pointed into a different direction, understanding autonomy as a relative property that requires a combination of different capacities and characteristics in the sense of autonomic computing. Hence, autonomous systems should be engineered considering, among others, features such as:

- The capability of obtaining data and information (either from the environment or the system itself)
- The capability of transforming or refining data and information in a way that can be used by the autonomous/ decision–making and cognitive elements
- The capability to handle, to understand and to generate concepts
- The capability to own cognitive aspects, meaning by such the capacity to reason, to infer and to learn
- The capability of making decisions to achieve the system's objective, based on data provided at a sufficient level of detail
- The ability to disseminate the decision taken to the appropriate execution/action elements

The former aspects imply and require new design and engineering paradigms —of strongly cognitive bias— to cater for the idiosyncrasy of autonomous systems: the perception process as major input for the system's knowledge, a more precise definition of system's objectives to allow some decision–making to be moved from the designer or the engineer to the system itself, the potential reconfiguration of the system as the objectives, capabilities or the environment changes, and finally the means to assess the adequacy of the current configuration against the current objective.

Secondly, the larger domain of general autonomous artificial systems —the focus of this work—*encompasses a great variety of systems* of very different nature, from bio–inspired ones to mobile robots. However, if autonomy is a solid concept, some common features shall be shared among different types of autonomous systems. This implies also that there is a possibility of building reusable technical assets to support the engineering of artificial autonomy.

The wide domain of autonomy and the multiresolutional nature of goals and decision making processes in complex systems require that these assets address problems at very different levels of abstraction. Hence, *the same feature can be defined or considered at rather different levels within different systems.* This lack of agreement can be a major drawback when trying to describe how to better engineer these systems of when building the reusable assets. However, it is a guiding principle of this research that the agreement shall be possible if the concepts are properly addressing the different abstraction levels.

Concepts for autonomy and the way they are organised and captured —the conceptualisation of the domain of autonomy— constitute the very focus of the work of this thesis and of the long–term project ASys[2]. An ontology that will capture all these features and relationships in a reasonable and unambiguous manner would allow a better characterisation of the autonomous system domain and offer a solid base for a systematic autonomy engineering programme.

It is the objective of this work to capture the core concepts for the engineering of autonomous systems. These concepts will the enable the systematisation of architectural knowledge and its use in the engineering of custom–fit autonomous systems. Knowledge about a domain is usually acquired by analysing system implementations and by talking to experts. This knowledge constitutes the very engineering resource when designing a solution to fulfill the users' requirements. Nevertheless, the development process is not exempt of issues [BCT05]:

1. Lack of a global picture of the problem: despite a thorough research to understand the problem space, system developers encounter a myriad of different viewpoints, partitioned domains of expertise, and unreconcilable priorities. **Common understanding** of the problem is, therefore, needed.

2. Several constraints to consider: when developing a hardware–software solution for autonomy, different constraints should be addressed. **Constraints** regarding both organisational activities (resources, and stakeholders) to technical ones (integration with existing applications and technologies) should be considered.

3. Never–ending changes throughout development: It is not uncommon that research projects should be updated, changed or reconsidered while the development takes place. Changing or updating software components and documentation are time– and effort–consuming tasks, which would be eased by developing the entire development project with **reusability** and autonomic properties in mind.

These issues will be of special importance in the domain of focus of this work —autonomous systems—, due to the wide variety of systems and engineering methods that are used in their construction. The ASys project reduces this variety by focusing on the software architecture and components, considering that advanced software is the main vehicle for achieving artificial autonomy. But software is nothing but the functional reification of a collection of abstractions that constitute the very knowledge of the domain. This is the focus of this work: the capture of a *general ontology for autonomous systems*.

1.2 Research Scope

The main context for this research is the long–term research project ASys [SR08], which addresses the development of a technology for the systematic development of autonomous systems. This includes methods for analysing requirements, architectures of autonomy and reusable assets with a cross-domain approach.

[2]ASys —The Autonomous Systems Project— is a long term project of the Autonomous Systems Laboratory (www.aslab.upm.es).

During their operation, systems might need to put up with external perturbations, changes from the original specification, or unexpected dynamics not always predicted. Production and automation engineers desire autonomous systems, capable of working on its own. However, this autonomy is considered to be bounded, i.e. the system to be fully autonomous but constrained by the engineers who have developed it. This is, in a sense, a strong point of difference between natural and artificial autonomy. Natural autonomous systems are considered to be truly autonomous systems in the etymological sense of the word (i.e. following their own behavioural laws). On the other side, artificial autonomous systems shall behave autonomously but only to a certain extent, being bounded by externally imposed restrictions (e.g. concerning safety, economics or environmental impact).

The strategical decision in the ASys project is the consideration of all the domain of autonomy, *i.e.*, to cater for any autonomous system regardless of its particular application. This implies a wide range of (autonomous) systems to be considered in the research, from robot–based applications to continuous processes or pure software systems. While this movement towards generality may imply a consubstantial reduction of powerfulness of the developments (abstraction may convey vacuity) the progressive domain focalisation [SAS+99] method will enable the derivation of progressively domain-focused assets that are, at the same time, compliant with the abstractions and capable of providing functional value.

A conerstone of ASys is the use of design patterns as the core vehicle for reusable architecture exploitation [SZ03]. The strategy followed to increase a system's autonomy will be by exploiting a particular class of patterns: *cognitive control loops*, which are control loops based on knowledge [SHR10]. This knowledge will be realised in the form of different models: of the system, of the environment, and of the task the system must fulfil in a particular environment.

The ASys project considers different elements to materialise the former ideas: an architecture–centric design approach, a methodology to engineer autonomous systems based on models, and an asset base of modular elements to fill in the roles specified in the architectural patterns. Different tasks and products are to be performed and obtained as a result of the long–term research (see Fig. 1.2), where the Ontology for Autonomous Systems (OASys) described in this dissertation, captures the concepts of the ASys research programme to be used as software support in the process of ASys engineering.

The ASys engineering process covers from the initial specifications and knowledge, to the final product *i.e.* the autonomous system. The first stage of this research focuses on ontologies, as a common conceptualisation to describe domain knowledge. Both a survey of existing domain ontologies and the development of an ontology for the domain of autonomous systems (OASys) are addressed. One of the central goals is to produce a methodology to exploit these ontologies to generate models based on the knowledge they contain.

Models constitute the core for our autonomous systems research programme. The type and use of models to be specifically developed for the autonomous system are ini-

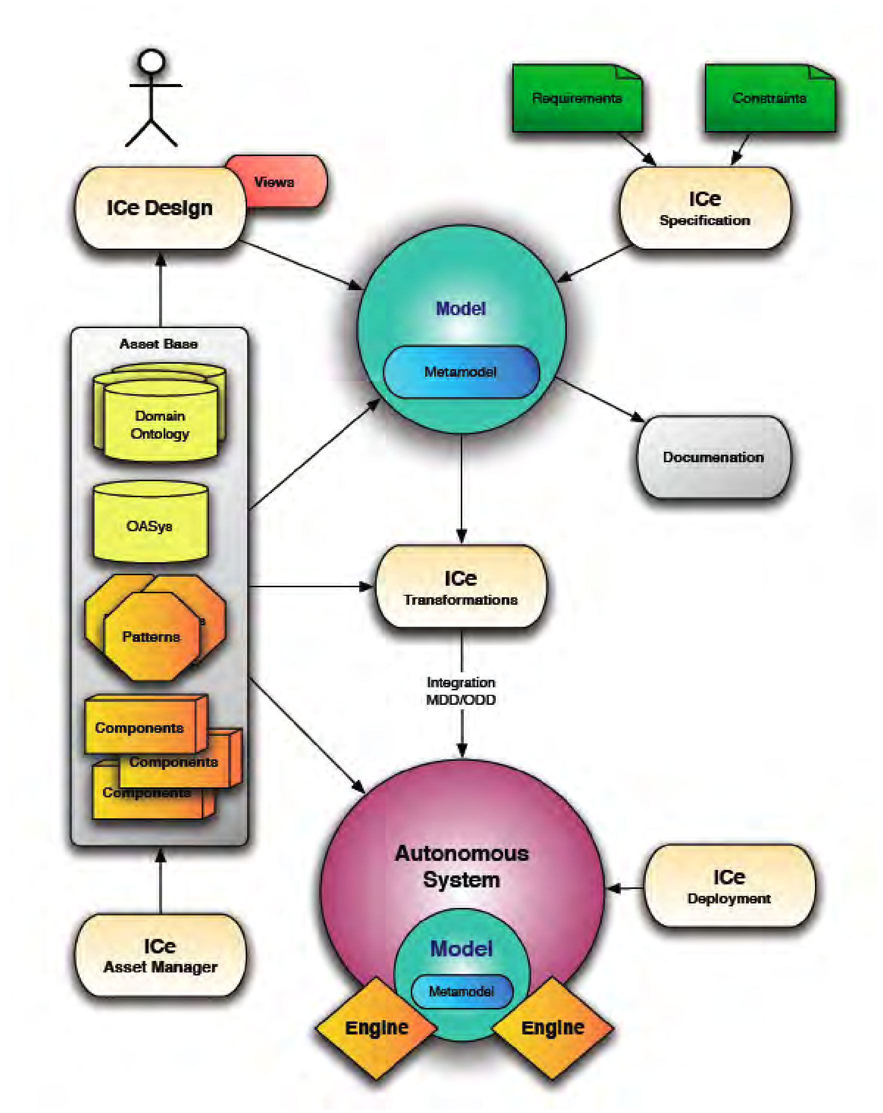


Figure 1.2: Tasks and products in the ASys long–term project (from [SR08])

tially considered. User and designer requirements, and constraints imposed by the system itself will guide the development of the models. The ASys Model Development Methodologies will address this model characterisation and development. The next stage is to extract from the built models a particular view of interest for the autonomous system. Unified functional and structural views are considered critical for our research as they provide knowledge about the intentions and the behaviours of the system.

The obtained models will be exploited by means of commercial application engines or customised model execution modules. Considering the metacognitive needs of autonomous systems, metamodels are addressed in the research. Progressive domain focalisation will be used to address the different levels of abstraction between metamodels and models. As an additional product documentation about the built models for their update, exchange, and query will be obtained.

The ASys approach is conceptual and architecture-centric. The suitability of extant control and cognitive control architectures and how they match the ASys research ideas and developed products (ontologies, models, views, engines) shall be determined and possible adaptations and extensions shall also be considered. The definition and the consolidation of architectural patterns that capture ways of organising components in functional subsystems will be critical. As a generalisation strategy, the different elements considered in the ASys research programme will be assessed in different testbeds.

1.3 Concrete Research Objectives

Given the enormous ambition of this research programme, it is necessary to establish the exact scope and dimension of this thesis work. The overall purpose of the ASys programme is to describe the fundamentals of autonomous systems, from knowledge to implementation, and the construction of reusable assets to aid in their description, characterisation and development.

By autonomous system, it should be understood a system that acts autonomously up to some level in an uncertain and changing environment to fulfill the mission or task it was designed for. The aim is to describe and aid in building systems that sense, analyse, learn, make decisions, and execute on their own actions which are reasonable and intelligent enough. This thesis addresses two core aspects of the engineering of autonomous systems: the fundamental concepts and the engineering processes.

The **OASys Framework** —the main product of this thesis— captures and exploits the concepts to support the description and the engineering process of any autonomous system. This is done by developing two different elements: an *autonomous systems domain ontology* (OASys) as a representation-based mechanism formalised in UML, and an *OASys-based engineering methodology* to describe a generic engineering development process based on OASys:

An autonomous systems domain ontology (OASys): The term autonomous systems covers too broad a domain, being applied to a wide range of systems. Finding common features and elements shared by different autonomous systems will aid for their description and engineering. Hence, a domain ontology describing concepts, relationships and axioms related to the autonomous systems domain will be developed in this research. The aim is to provide generic building elements to cater from generic autonomous systems to specific applications.

Ontologies can be used as representation–based mechanisms based on a computational language, to describe the different entities participating in the operation of the autonomous and autonomic systems: different domains, the environment, the objects the systems interact with, the possible actions to be taken, resources to be considered, etc.

An OASy–based Engineering Methodology: Once the domain ontology is described, it serves as the basis for an ontology–based engineering process for autonomous systems, whose structure aims at being re–usable and scalable.

Software–intensive systems are usually developed in an ad–hoc engineering process, i.e. specifically built to fulfill specific requirements of a project or application, without general guidelines which will allow a further partial or full re–use. Furthermore, a myriad of different users (either human or artificial) interact in the engineering process, generally without sharing a common conceptualization of the problem. The development process is usually prone to misunderstandings, as well as time– and effort–consuming. A common conceptualisation of relevant concepts and development techniques will be useful to aid in the requirements specification and development phases of an autonomous system engineering process.

1.4 Thesis Structure

The research had to cover a wide range of topics, from general concepts to more specialised ones. The research methodology followed different stages, reflected in the different parts of this dissertation summarised in Table 1.1.

The first chapters of the thesis summarise the initial phase of *background* on the underlying topic of the research: systems and autonomous systems. The different chapters in this part gather general ideas and concepts related to both topics, as follows:

- Chapter 2 describes the General Systems Theory underlying ideas and fundamental concepts. Being the purpose of the ASys programme to cater for any kind of (autonomous) system, this theory was considered as the core backdrop of the global project research.
- Chapter 3 summarises the joint research on autonomous systems carried out by the Autonomous Systems Laboratory (ASLab) on the concept of autonomy, as well as providing some insights on the view of autonomous systems through time.

The next chapters comprise the second stage of the research, the *state of the art* of the different theories and developments for ontologies. Also, a review on ontologies for autonomous systems and software engineering is provided.

- Chapter 4 gathers the fundamentals on ontologies. The chaper revises definitions available in the literature, revises the different types of ontologies, as well as their role and application.
- Chapter 5 reviews well established methodologies, languages and tools for ontology development, both from the traditional ontological and from the software engineering viewpoints.
- Chapter 6 reviews the different ontologies developed up to date for autonomous systems engineering. A first part reviews ontologies within the domain of autonomous systems. A last part of the chapter focuses on the concept of ontology–driven engineering, and its application for software engineering.

The last part focus on the *research and its application.* An initial chapter states the ontology general requirements, and provides a description of the detailed development ontology and its contents. How the ontological elements can be used is described in a chapter dedicated to the OASys–based Engineering Methodology. Next, the description on how the ontological elements have been used for the conceptual modelling of selected applications.

- Chapter 7 describes the development process of the ontology based on the METHONTOLOGY methodology [FLGPJ97]. This chapter also provides a detailed description of the key concepts considered in the ontology.
- Chapter 8 proposes the OASys–based engineering methodology for autonomous systems.
- Chapter 9 describes the use, in terms of the application of the developed ontology to the different testbeds considered within the global ASys project. How the concepts, relationships and elements considered in the ontology are specialised for particular systems.

Concluding remarks to the research are provided, as well as future lines of research are suggested in Ch. 10. Finally, a set of annexes collects the additional elements in the research, such as the detailed ontology description, and the bibliography considered in the different chapters.

THESIS PART	RELATED CHAPTERS	PURPOSE
Background	Chapter 1	To state the research scope and objectives
	Chapter 2	To describe General Systems Theory and related concepts
	Chapter 3	To explore the concepts of autonomy and autonomous system
State of the Art	Chapter 4	To review ontology related topics (definition, types, languages, roles, applications)
	Chapter 5	To survey existing ontology tools and methods
	Chapter 6	To analyse the use of ontologies for autonomous systems engineering
Research and Application	Chapter 7	To define the requirements for the ontology to be developed and to describe in detail the ontological elements considered in OASys
	Chapter 8	To propose and describe the OASys–based Engineering Methodology
	Chapter 9	To ilustrate the application of OASys to some exemplary systems
Conclusion and Annex	Chapter 10	To draw conclusions on OASys and to propose new lines of research
	Chapter 11	Glossary of OASys ontological elements

Table 1.1: Thesis Structure

Chapter 2

GENERAL SYSTEMS THEORY

2.1 Introduction

This chapter introduces the main underlying ideas of the General Systems Theory, which allows for the description of a system in general, regardless of the scientific domain considered. Next, special attention is paid to the approach within this Theory described by G. J. Klir in [Kli69], [Kli91], and [KE03]. A summary of his concepts, used to characterise any system, is given in the last sections.

2.2 General Systems Theory

The emergence of interdisciplinary areas in science, and the recognition that certain concepts, ideas, principles and methods were applicable to systems in general led to the notions of general systems and the like [Kli91]. The terms "general system" and "general systems theory" are due to Ludwig von Bertalanffy [vB50], [vB68]. According to him, "general systems theory is a logicomathematical field whose task is the formulation and derivation of those general principles that are applicable to systems in general".

Among early proponents, Kenneth E. Boulding envisioned general systems theory as "a level of theoretical model–building which lies somewhere between the highly generalized constructions of pure mathematics and the specific theories of the specialized disciplines" [Bou56]. He justified the need for such a theory based on the sociological situation in science at the time, where specialization was leading to a break–up in the body of knowledge. Hence, "General Systems Theory is the skeleton of science in the sense that it aims to provide a framework or structure of systems on which to hang the flesh and blood of particular disciplines and particular subject matters in an orderly and coherent corpus of knowledge".

W. Ross Ashby proposed as a general method to be followed: the consideration of "all conceivable systems", regardless of their existence in the real world. He also proposed the so–called Problem of the Black Box, as a means to identify system's properties [Ash58]. Ashby influenced ideas quite similar to those related to general systems research, although with a focus on information processes in systems, such as communication and control, under the name "cybernetics" [Ash56].

This work has mainly focused on the research carried out by George J. Klir, who regards Systems Science as a cross–disciplinary discipline, which overcomes boundaries among different domains, as opposed to the classical science divided in countless specialized domains [Kli91]. According to Klir, such a science comprises three different components: a domain of inquiry, a body of knowledge about the domain, and a methodology for the acquisition of new knowledge as well as for its use to deal with relevant problems in the domain. His first attempts to summarize a general framework for system's characterization and classification were described in [Kli69]. His research evolved to propose the General Systems Problem Solver (GSPS), as a conceptual framework where types of systems problems are defined, together with methodological tools to solve them [KE03].

2.3 System Characterisation

Systems appear as observers make appropriate distinctions, either on real world or abstract entities. If we consider that all possible entities form the universe, a system can be regarded as an isolated and distinguishable part of it which is under investigation. All which is not system is called environment. Different disciplines of science share this general understanding, however paying attention to different criteria to separate the system from the surrounding universe. As stated by Rosen [Ros86] , "the word system is almost never used by itself; it is generally accompanied by an adjective or other modifier: physical system; biological system; social system; economic system;... and even general system. This usage suggests that, when confronted by a system of any kind, certain properties are to be subsumed under the adjective, and other properties are subsumed under the "system", while still others may depend essentially on both".

Klir proposes that "any given entity must satisfy two requirements to qualify as a system: it must consist of a set of things of some sort and these things must be related or connected in some recognizable manner... all additional distinctions are basically of two types: those applicable to the things involved in the system, and those applicable to the relations recognized among the things" [Kli85].

Within the engineering domain, a *system* is conceived as an entity whose interaction with the *environment*, as well as its internal operation can be known by analyzing certain properties about its *elements*, how these elements behave, and how the system interacts with the environment [UIT06d]. We shall point out two important issues regarding this definition. Firstly, the boundaries between a system and its environment might not always be stated clearly, as the system's definition relies on the observer. Secondly, the status of a system as an overall system or as an element is not absolute, as the same system can be regarded as a whole in one context, and as an element or subsystem in another. This way, a system can be conceived as a whole consisting of subsystems, which in turn, consist of interrelated elements, and so forth until finding basic elements which cannot be further decomposed.

2.3.1 Fundamental Concepts

A system is defined by selecting a set of relevant *properties* or *attributes*. The *environment* of the system is another system, whose elements are not part of the observed system. Each property of the observed system is assigned a *quantity* or *variable*. Each property is associated with a set of possible *appearances*. In an observation, the attribute takes on a particular appearance. To determine possible changes, different observations should be made. To distinguish the different observations of the same property or attribute, a *backdrop* is used. Typical backdrops are time, space, population or their combinations, being the most common one the combination time–space or only time.

A quantity or variable can be considered as an abstract or operational representation of an attribute. In the same way, a backdrop is represented by an operational representation called *support*. Quantities are associated with *values* as part of a *state set*. Likewise, supports are associated with a *support instance* which are part of a *support set*. Both state and support set can be ordered or can show discrete or continuous properties. The determination of the state set along with the support set is the *resolution level*, which determines the frequency and the accuracy the observations are made with.

The variation of all the values of the quantities over a period of time, if time is used as backdrop, constitutes the *activity* of the system. The observations are translated into a *support–invariant relation*, as a relation between observed quantities that is satisfied according to the chosen support. For example, when time is used as backdrop, the support–invariant relation becomes the *time–invariant relation* as the relation between observed quantities that is satisfied within a time interval.

The main task of the engineer is to synthesise a particular class of activities. This will be accomplished by identifying patterns in the activity of the system. The quantities or variables of the system satisfy time–invariant relations, by which the values of some variables may be expressed as function of others. The set of all time–invariant relations is the formal notion of *behaviour* of the system. Different kinds of behaviour can be distinguished in a system:

- *Permanent or real behaviour* expressed by the time–invariant relation satisfied over the entire time interval of *every possible activity* containing the variables at the backdrop or resolution level.
- *Relatively permanent behaviour* expressed as the relation which is satisfied anywhere *within a particular activity* containing the variables at the resolution level.
- *Temporary behaviour* expressed as the relation which apply only within some shorter time intervals of a particular activity.

If the system exhibits a behaviour, it must contain certain attributes or properties which produce it. Hence, a system with different properties would exhibit a different behaviour. These properties are called the *organization* of the system. If the behaviour of the system can change, so does the organization. To reflect this possibility of change, the organization can be split into a constant part called *structure* and a variable part called *program.* Three different kinds of programs can be distinguished: *complete program, subprogram and instantaneous program* [Kli69]:

- Complete program – the instantaneous state together with the set of all other states of the system, and the set of all transitions from the instantaneous state to all states of the system in time.
- Subprogram – the instantaneous state together with a nonempty subset of the set of all other states of the system, and a nonempty subset of the set of all transitions from the instantaneous state to all the states under consideration in time.
- Instantaneous program – the instantaneous state together with the transitions from this state.

Systems are generally composed of simpler systems called *subsystems* which can be further decomposed until reaching non–decomposable *elements.* The behaviour of the system is influenced by the behaviour of its subsystems or elements, and by certain compositions of these behaviours. A composition of two element's behaviour takes place only if there are some common observed or external quantities relating the respective two elements. The set of all common external quantities between two elements is called *couplings* (Fig. 2.1). There are *real couplings* which are valid over the entire time interval of any activity of the system. There are also *hypothetic couplings* which are valid anywhere within a particular activity of the system. The set of all elements of system and their couplings is called the *structure of universe of discourse and couplings*, abbreviated as *UC structure.*

To complete the description of a system, it is necessary to analyse the evolution of the values of its quantities. The system can be observed at a certain point of its activity, which will provide its *external quantities.* There are also *internal quantities*, which are not observed, but that play a mediatory part. The set of instantaneous values of all the quantities of the system (both external and internal) is called the *system state.* At the next instant of observation, the system will have evolved to a different state. This evolution is called a *transition*, i.e., the admissible changes of the values of the quantities determined by the combination of elements and couplings, and affected by the environment. Given the system at a certain state, not any transition is possible, only a set of other states is reachable from the original one. Therefore, each system state is associated to a set of possible transitions. The complete set of system states, together with the complete set of transitions between the states is called the *state–transition structure*, abbreviated as *ST structure.*

A system could be characterised using any of the characteristic traits already described: external quantities, resolution level, activity, behaviour, a set of states, a set of transitions between states, the instantaneous state, ST structure, program, couplings, UC structure, and structure. Nevertheless, among these traits only those constant, completely known, uniquely determined, and non redundant should be considered for a sys-

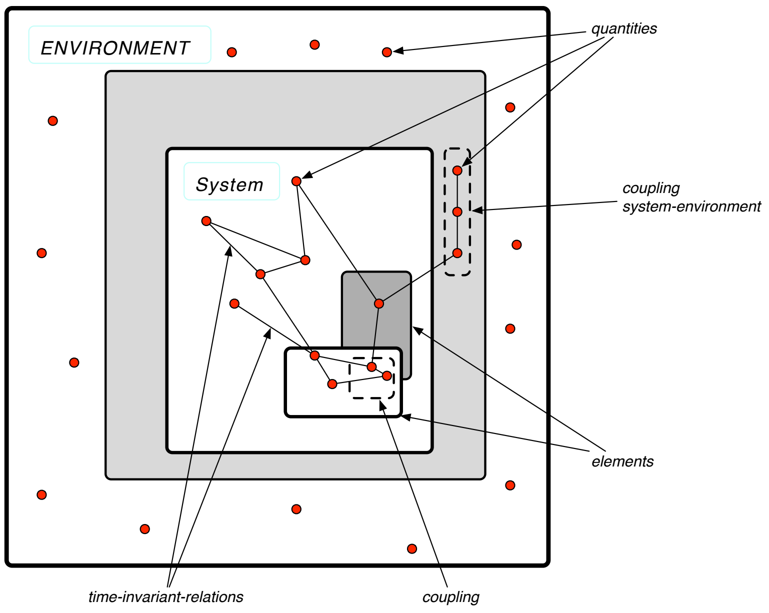


Figure 2.1: General Systems Theory concepts (adapted from [HSL08])

tem's definition. These traits are called *primary traits*. Any other trait of the system that are not involved in the definition are called *secondary traits*. Hence, it is sufficient to consider only the following constant traits:

1. The set of external quantities together with the resolution level.
2. A given activity.
3. Permanent behaviour.
4. Real UC–structure.
5. Real ST–structure.

Each one of these traits satisfies all the requirements imposed on the basic definitions (based only on constant traits, completely known, not underdetermining, not containing redundant traits). Hence, each one can be used for a basic definition of a system. Any intersection among two or more of them also serves as a definition of the system, representing a smaller class of systems. The intersection of all of them defines a single system.

2.3.2 Goal–oriented Systems

Goal–oriented systems can be described using any of the general concepts defined in former sections. However, additional concepts which are specific to characterise the feature of goal orientation are required, as described in [Kli91] and [KE03]. As example, for goal–oriented systems, there are two concepts of fundamental importance such as goal and performance.

The concept of *goal* can be defined in many different ways, without neglecting that is "in the eyes of a cognitive agent (observer, investigator, user, designer)." This means that a goal associated with the system is a concrete restriction of the primary or secondary traits, which the user determines as required under given circumstances. Examples of a system's goal could be such as maintaining an output variable within a range or acquiring in an autonomous way a particular functional relationship.

To evaluate how a system satisfies a defined goal, the concept of *performance* of a system with respect to a goal is used, which measures the similarity between the actual and desired values of those traits considered when defining such goal. To express it, a *performance function* can be used, which allows to quantify the performance of the system. The combination of goal and performance function permits to evaluate and to compare the compatibility and the adequacy of a system to a defined goal.

Goal–oriented systems show what is called *goal–seeking traits*, that is to say some variables, additional elements, and additional couplings, which are directly involved in achieving their goal and thus, increasing their performance. Such *goal–seeking variables* are distinguished from other *goal–implementing variables* which are used to define the goal. The former are generated by what is called *goal–seeking element*, and are usually used as additional inputs to *goal–implementing elements*.

The concept of a goal-oriented behavior system, as described in former paragraphs, is open to different interpretations. Goal–orientation has been chosen as a neutral term that encompasses different types such as regulation, control, adaptation, self–correction, self–organising, autopoiesis, learning, decision–making, etc. One of the most usual one, is that of regulation or control systems, whose goal is to keep output variables in a particular state or a particular subset of states in spite of disturbances, which are represented by the input variables. The goal-seeking variables assume the role of regulating variables. Some additional interpretations are those of decision–making systems, error–correcting systems, self–organising systems, autopoietic systems, etc.

The design of a goal–oriented system resides in defining the goal and the performance function in an explicit way, the definition or selection of a set of goal–seeking variables, the determination of how such variables are generated, and the design of the goal–seeking elements. The underlying assumption is that neither the behaviour function of the goal–implementing element nor the goal will change. However, it may be the case of systems to be designed to cope with changes in their goals. Therefore, to enable a goal-oriented system to adapt to these changes, its goal-seeking element must be designed for a set of alternative goals and supplemented with a special input variable, *goal–designating variable* whose states represent the goals. This kind of goal–oriented system will be called a *multigoal-oriented system.*

Changes in the goal may be determined by a goal-generating system that is included as an element in the multigoal-oriented system itself. Sometimes, an overall goal could be defined as a sequence or hierarchy of subgoals, represented by states of the goal–designating variable. These sequences are determined by a behavior function of the goal–generating elements in terms of input, output, and goal–seeking variables. This type of multigoal-oriented systems can be called *autonomous multigoal-oriented systems.*

Chapter 3

AUTONOMOUS SYSTEMS

3.1 Introduction

The concept of autonomy has long been discussed in the literature, that however, has not ended up with a unique definition. On the contrary, autonomy leads to a myriad of interpretations, due to its nature of abstract quality which makes it difficult to find a formal description, as well as the existence of different ways for a system to be autonomous [ZHSL06]. This chapter aims at summarising some underlying ideas, from the concept of autonomy to the idea of autonomous system in engineering domains.

3.2 On the Concept of Autonomy

On its etymological origin, autonomy comes from Greek. According to [Pre], [MW], its meaning relates to self–governing or giving itself its own laws. For natural systems, the concept of self–governance is sustained as underlying idea:

> Autonomy understood as *self–governance* and does not imply that a system can be separated from its medium....There is no autonomy in this sense because every living system exists in a medium. What influences a system, however, is determined by its internal dynamics, which shapes these influences in quite particular ways. When the system finally dies, this means that it was incapable of keeping itself alive, that it lost its autonomy [MP04].

For artificial systems, autonomy usually refers to systems capable of operating in the real-world environment without any form of external control for extended periods of time [Bek05]. Regarding autonomy, the emphasis is placed on aspects such as: absence (totally or partially) of human intervention, minimum dependence with the environment, and system cohesion [UIT06d].

Autonomy results from the combination of different capacities and characteristics: fault-tolerance, intelligence, knowledge and response time among others. Systems are designed stressing some aspects more than others, which leads to a kind of autonomy closer to one of the aforementioned aspects than to the others [UIT06c].

Autonomy in artificial systems has sometimes been developed inspired by biological or living systems. Properties usually considered in living organisms can be attributed to artificial systems: some form of autonomy, some form of situatedness, and some form of embodiment [ZHSL06]. An autonomous system, like a human being, is given instructions to perform a task, to later proceed on its own to learn, modify and use its understanding [Gyu06].

Nevertheless, the idea of biological self–governance has, somehow, evolved in artificial autonomy as the concept of the system being capable of operating in an environment following a predefined goal or mission [Hua04], [MA02]. Strictly speaking autonomous means just self-laws not self-objectives. However, in the context of autonomous systems research the distinction between laws and goals is not always made in detail.

Moreover, in the case of artificial systems, the quest for autonomy is differently pursued. Firstly, engineers do not try to achieve total autonomy because that might lead to unforeseen problems, when a system would follow its own objectives, and not the initial purpose it was built for. Secondly, aiming at total autonomy in engineered systems might be difficult to reach from both a technical and cost viewpoint [Gyu06]. Engineers, therefore, search for *bounded* and *made–to–fit* autonomy to clarify that the system must operate under constraints and goals imposed by design, and this level of autonomy is less costly to achieve [SR08]. In a sense, the engineering of autonomous systems is a search of robust, bounded autonomy.

3.3 Autonomy in Real Systems

There are many reasons to pursue the objective of autonomous artificial systems. Firstly, economical reasons as automated systems are less costly from an operational view of point, either due to higher performance, human personnel cost reduction, improved operating conditions, etc. Secondly, technical reasons such as obtaining higher production, safer working conditions, higher availability among others, have justified the quest for autonomy [SR08].

Although autonomy is desired for systems' performance and dependability, real systems are faced with several operating conditions that constraint the way they can operate autonomously [UIT06d], [DTC08]:

- External perturbances due to the interaction of the system with its environment, which might make the system to evolve in a different way, thus diverging from the desired operational state. Perturbances are not usually considered in the system's design, due to their unknown and random origin (when considered, the complexity of the system increases). However, they are usually modelled, e.g. statistically, to embed into the system the necessary mechanisms to deal with them.

- Increasing levels of abstraction due to the existence of more complex and abstract tasks to be performed by artificial systems. Low levels of abstraction imply tasks such as sensing, calculating and acting based on existing sensor lectures. Higher levels refer to perception, thought and action where more abstract elements and concepts are used for decision–making.

- Unmodelled dynamics that will appear when the system operates as specified, but it reaches an undesired operational state which was unpredicted. Real systems operate in real environments, which are usually unstructured and uncertain. Systems provided with autonomy react to compensate possible uncertainties in the environment, but a point where the system is deviated from its goals can be reached, leading to unexpected operational states.

Therefore, the general principle for autonomy in an artificial system is *adaptivity*, which enables the system to change its structure and way of operating to compensate for the perturbances and the effects of the uncertainty of the environment, while preserving convergence to their objectives. Different operational aspects are studied in artificial systems to enhance adaptivity [SLBH06]:

1. Cognition: systems which are tightly grounded to the physical substrate have reduced adaptivity, due to mechanical constraints, than systems with cognitive capacities based on knowledge, learning and intelligence.

2. Modularity: large systems can be structured as modules, where each one usually performs a specific function, interacting with the rest of the system through a well-defined interface. Interfaces make that dependencies between one module and the rest of the system are determined, allowing interchangeability of modules. An explicit structure, functional decomposition and defined dependencies are critical factors for adaptivity. Uncertainty, perturbances and planning may eventually require reconfiguration of system parts, or in the way they interact with each other.

3. Fault–tolerance: system adaptivity relies on its capacity to keep its objectives under real conditions, such as perturbances and uncertainty. Eventually, parts of the system may be damaged or malfunction during operation, compromising system's capacity to achieve objectives. Fault tolerance techniques have been developed to provide the system with mechanisms to react to these circumstances by adapting itself.

4. Soft–computing: in artificial intelligence, a series of techniques have been developed to make systems able to operate with uncertain, imprecise or abstract elements, such as neural networks, fuzzy logic, and expert systems.

A high level of adaptivity can be achieved by using the following design principles in artificial systems: minimal structure, encapsulation, homogeneity, isotropy of knowledge, scale, and scalability [SR08].

3.4 On the Concept of Autonomous System

Most research on autonomous systems regard autonomy as equivalent to the capability of movement in an environment. Hence, the main attention was paid to characterise *autonomous (mobile) robots* [MA02], [BCKS04], [CTSB04], [WVH04], [Bek05]. A more comprehensive attempt for robotic systems is the Autonomy Levels for Unmanned Systems (ALFUS) Framework [Hua04], where autonomy is described for such systems as:

> (A) The condition or quality of being self–governing. (B) A UMS's own ability of sensing, perceiving, analyzing, communicating, planning, decision–making, and acting, to achieve its goals as assigned by its human operator(s) through designed HRI. Autonomy is characterized into levels by factors including mission complexity, environmental difficulty, and level of HRI to accomplish the missions.

where UMS stands for unmanned system and HRI for human robot interaction. For an unmanned system, the key point is to be able to acquire information on the environment where it interacts, which will be used to plan its action to fulfil a pre–established goal. This could be made with some interaction with a human operator, therefore, different levels of autonomy could be considered.

Another focus on autonomous systems was *autonomous (intelligent) agent*, as "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [FG96]. Autonomy is, then, relative to something external to the agent, since an agent is not autonomous in an abstract sense but it is autonomous to some degree with respect to another entity, be it the environment, other agents, or even the developers of the agent.

Lately, the concept autonomous system has been extended to envision highly complex systems, where its behaviour is a function of multiple subsystems working in coordination, that exhibits some of the following capabilities [IBM01], [DTC08], [HM07]:

- self–reorganisation: to change its structure as required,
- self–configuration: to configure itself according to high–level goals, being able to adapt to changes,
- re–allocation of tasks: to allocate tasks according to the existence of new or free resources,
- self–optimization : to improve its own performance in an efficient manner,
- self–adaptation: to respond faster, more efficiently and effectively,
- self–repair: to correct faults, or to use existing resources to replace functionality,
- self–healing: to automatically detect, diagnose, and repair itself,
- self–protection: to use early warning to anticipate and prevent its failures,

- decision–making: to make selections from a set of decision alternatives,
- learning: to respond to new data or environments, and
- perception: to exploit and optimise the usage of sensors and actuators, by information analysis.

Hence, the concept of autonomous system has expanded its range to refer to autonomic systems, such as cognitive computers, intelligent systems and autonomic agent systems ([Wan07a], [Wan07b]). To be autonomic, a system must know itself as well as its boundaries and environment, reconfigure itself, continually optimize itself, repair or heal from malfunction, protect itself, and all these in a heterogeneous and uncertain world, while keeping its complexity hidden from the user. Autonomous and autonomic software–based systems are merging as a research topic [HM07], [NS06]. New paradigms and frameworks for such systems are being developed, such as autonomic computing [KC03], autonomic–oriented computing [JL04], and general–purpose autonomic computing [Cal09].

Part II

State of the Art

Chapter 4

FUNDAMENTALS ON ONTOLOGIES

4.1 Introduction

This chapter is devoted to summarise the state of the art on ontologies. The intention is not so much to provide a comprehensive review on ontology theories, as already made in [GPFLC04b], but to focus on ontology related concepts useful for the research. Firstly, some definitions on the concept of ontology are provided, with a special emphasis on the knowledge engineering approach. A review of different types of ontologies is made in the next section, as the considered types vary depending on the intended use of the ontology. Additionally, the design criteria usually considered when developing an ontology are also summarised, as they guide the ontological engineering process. Finally, the most common applications for ontologies are addressed.

4.2 What is an ontology

Several definitions on what an ontology is can be found in the literature. The definition depends strongly on the domain where the term is being used.

4.2.1 Philosophy

Ontology was a concept originally linked to Philosophy where it means the philosophy of being (ontos=being and logos=treatise). We do not want to accomplish a thorough description on the philosophical issues related to it. It goes far back to Aristotle to describe the existence of beings in the world.

4.2.2 Knowledge Engineering

The term ontology became relevant to the Knowledge Engineering community some years ago. It was borrowed from Philosophy as ontology was understood as a systematic account of Existence. For knowledge-based systems what exists is what can be represented [Gru93a].

Gruber's definition

One of the best known is the definition provided by Gruber [Gru93a] and further detailed in [Gru93b]:

An Ontology is an explicit specification of a conceptualization.

What a conceptualisation is has been described in different ways. It was introduced in [GN87] which defined it as "a set of extensional relations describing a particular state of affairs". It was further described as "an abstract and simplified view of the world of our interest and which we want to represent, i.e. objects, concepts and entities as well as the relationships among them" [Gru93a]. Conceptualisation can also mean an intensional semantic structure which encodes the implicit rules constraining the structure of a piece of reality [GG95].

The original Gruber's definition has been modified either specifying it to be "a *partial account* of a conceptualization" [GG95] or "a *formal* specification of a *shared* conceptualization" [BAT97].

It is important to clarify that, regardless of later modifications from the original definition, an ontology consists of classes, instances, functions, relationships and axioms. *Classes* correspond to entities in the domain. *Instances* are the actual objects which are in the domain. *Functions* and *Relationships* relate entities in the domain. *Axioms* constrain the use of all the former elements [Gru93a].

Guarino's definition

"Ontology" (with capital o) to refer to the philosophical sense and "ontology" when referring to the Knowledge Engineering sense was proposed as a subtle distinction by Guarino and Giaretta [GG95]. However, the proposal has not been widely followed by the rest of researchers. Therefore, "Ontology" can be found when referring to Knowledge Engineering or Artificial Intelligence related ontologies.

According to [Gua98] an ontology is " a logical theory accounting for the *intended meaning* of a formal vocabulary, i.e. its *ontological commitment* to a particular *conceptualisation* of the world. The intended models of a logical language using such vocabulary are constrained by its ontological commitment. An ontology indirectly reflects these commitments (and the underlying conceptualisation) by approximating these intended models" (Fig. 4.1).

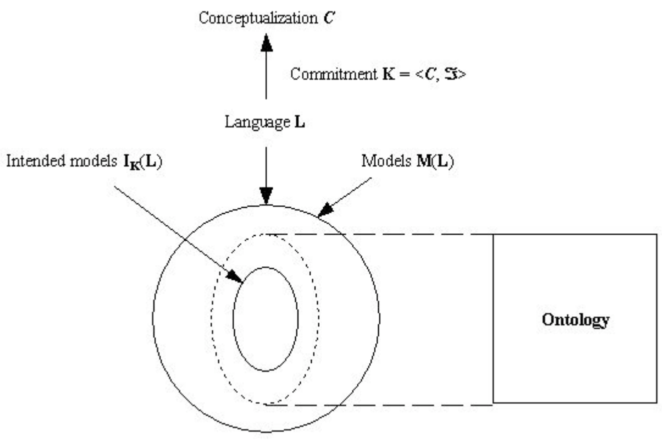


Figure 4.1: Elements in Guarino's definition (from [Gua98])

It is a new approach on what an ontology is: a combination or interaction of conceptualization, ontological commitments, language and models. As he considers, ontologies are language dependant (i.e. a developed ontology will look differently in English or in Spanish when finally written down) whereas conceptualizations are language independent (the concepts and relationships for a domain should be the same regardless the language used).

Other definitions

Although most widely known, Gruber's and Guarino's definitions are not the only ones provided by the literature. Some further definitions are:

An ontology has been described by Uschold [UG96] as "an explicit account of a *shared understanding* in a given subject area" .

A very detailed and summing-up definition was provided by [SBF98]: "An ontology is a formal, explicit specification of a shared conceptualization" where the different words mean:

Conceptualization: abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon.

Explicit: the type of concepts used and the constraints on their use are explicitly defined.

Formal: the ontology should be machine-readable.

Shared: an ontology captures consensual knowledge, i.e., not private to some individual, but accepted by a group.

A further definition of ontology was provided by [JU99] where an ontology is considered to be composed by both a vocabulary of terms and some specification of their meaning. The latter entails the indication on how the concepts are inter-related and the constraints for their interpretation. This definition highlights the importance of *amount of meaning* within an ontology, considering it to be directly related to restricting the possible interpretations on attributes and concepts in the ontology. The same idea is followed in [Gua98] where ontology is described as an engineering artifact consisting of a specific *vocabulary* and the *intended meaning* of the words in the vocabulary.

Additionally, [CJB99] considers ontology as a twofold definition. On the one hand, ontology is a *representation vocabulary* specialized for some domain. However, it is not the vocabulary itself what is important but the conceptualization that the terms included in the vocabulary attempt to capture. On the other hand, ontologies are *content theories* about objects, their properties and the existing relationships among those objects that are possible within a specified domain of knowledge. The key point of both definitions is that they constraint the ontology to be applied to a particular domain, therefore, focusing only on the relevant concepts for such.

All the aforementioned definitions share the idea that an ontology provides the description of a particular domain under a specific viewpoint, being it explicit, i.e. establishing a vocabulary and constraints to some level of formality. Moreover, a group commits to use such vocabulary with the intended meaning to communicate [Tam01]. One should bear in mind that in fact, there is no unique conceptualisation of a domain as a domain can be conceptualised in different ways depending on the viewpoint.

Thus, it is not unusual that an ontology could be interpreted as rather different things when applied. A comprehensive list of the possible interpretations is provided by [GG95]:

1. A philosophical discipline.
2. An informal conceptual system.
3. A formal semantic account.
4. A specification of a "conceptualization".
5. A representation of a conceptual system via a logical theory characterised by
   (a) Specific formal properties
   (b) Its specific purposes
6. The vocabulary used by a logical theory.
7. A (meta–level) specification of a logical theory.

4.3 Types of ontologies

If coming up with a sole definition of ontology is too difficult a task, which types of ontologies could be considered is also a troublesome one. Several classifications of possible types appear on the literature.

A first classification considers ontologies not from the point of view of the type of conceptualisation but from the kind of language [UG96]. Ontologies are classified as:

- Highly informal: the ontology as expressed using natural language.
- Semi–informal: the ontology is expressed in a structured and restricted form of natural language.
- Semi–formal: the ontology is expressed in an artificial and structured language.
- Rigorously formal: the ontology is expressed using defined terms with formal semantics, theorems and proofs of properties such as soundness and completeness.

However, many authors have sorted out ontologies paying special attention to the underlying conceptualisation or the terms described in the ontology. As an example, the classification provided by [Gua97], [Gua98] classify ontologies as:

- Top–level ontologies: this type describes general concepts or common–sense knowledge such as space, time, matter, event, etc., without considering a particular domain.
- Domain ontologies: this type describes the vocabulary and concepts of a generic domain such as medicine.
- Task ontologies: this type describes concepts for a generic task or activity such as selling.
- Application ontologies: this type describes concepts within a domain for a particular task, as an specialisation of the two previous types when domain entities perform certain activities.

Criteria, as defined in [NH97], can be used to characterize and to compare different ontologies:

- General: the purpose the ontology was created for, general or domain specific, size (number of concepts, rules and so forth),implementation platform and language (if done), publication (if done).
- Design process: how it was built and its evaluation.
- Taxonomy: ontology organization, one or several ontologies, time treatment, etc.
- Internal concept structure: properties, roles, part-whole relationships.
- Axioms: explicit axioms, expression of axioms.
- Inference mechanisms: how is reasoning done (if any).
- Applications: user interface, application in which the ontology was used.
- Contributions: strengths and weaknesses.

Ontologies have also been classified according to two dimensions as proposed by [vHSW97]:

1. Amount and type of structure of conceptualisation: the level of granularity of the conceptualisation.
    - Terminological ontologies: they specify the terms to be used to represent knowledge in a domain. They are more of a lexicon than an ontology.
    - Information ontologies: they specify the record structure of databases.
    - Knowledge modelling ontologies: they specify conceptualisations of knowledge.
2. Subject of conceptualisation: the type of knowledge modelled in the ontology.
    - Application ontologies: the ontology contains the definitions needed to model the knowledge for a particular application.
    - Domain ontologies: they express conceptualisations for a particular domain.
    - Generic ontologies: the concepts considered in the ontology are generic across many domains.
    - Representation ontologies (also called meta–ontologies): they provide a representational framework without making claims about the world. They are used to describe domain and generic ontologies.

Other authors [LD01] classify ontologies according to the information the ontology needs to express and the richness of its internal structure. Ontologies are therefore classified as: controlled vocabularies, glossaries, thesauri, informal is–a hierarchies, formal is–a hierarchies, formal instances, frames, value restriction and general logical constraints.

Different types of ontologies can be considered when combining the concepts proposed by several authors (Mizoguchi et al, van Heijst et al, Lassila et al) [GPFLC04b]:

- Knowledge representation ontology: it captures the representation primitives (classes, relations, attributes, etc) used to formalise knowledge under a given Knowledge Representation (KR) paradigm (frame and first order logic, description logic, etc). Examples are Frame Ontology [Gru93a] and the OKBC Ontology [CFF$^+$98].

- Upper or Top–level ontology: it represents common sense knowledge to be reused among domains. The ontology vocabulary contains terms related to things, events, time, space, etc. A good example is the Mereology Ontology [BAT97].

- Top–level/Upper–level ontology: it describes very general concepts as well as providing general notions under which all root terms in existing ontologies should be linked. However, the existing top–level ontologies provide different criteria to classify the most general concepts. Examples are the SUMO (Suggested Upper Merged Ontology) [NP01], Cyc ontology [LG90], BFO (Basic Formal Ontology) and DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [MBG$^+$03].

- Domain ontology: it is an ontology reusable in a given specific domain (medical, engineering, enterprise, etc).

- Task ontology: it describes the vocabulary related to a generic task or activity by specialising the terms in the top–level ontologies.
- Domain–task ontology: it is a task ontology reusable in a given domain but not across domains.
- Method ontology: it gives definitions of relevant concepts and relations applied to specify a reasoning process to achieve a particular task.
- Application ontology: it contains all the definitions needed to model the knowledge required for a particular application.

The *reusability* of an ontology is bottom-up from application ontology (the less reusable) to Knowledge Representation ontologies (the most reusable). The *usability* goes top-down. In general, the more reusable an ontology is, the less usable it becomes (and viceversa).

4.4 Design criteria

When building an ontology, it is worthwhile considering some design criteria as defined in [Gru93b]. These design principles allow to evaluate the created design with a focus on knowledge sharing:

Clarity: Ontologies are designed to communicate and to share the meaning of defined terms, as objective as possible. This could be achieved by means of formal axioms and complete definitions.

Coherence: Inferences should be consistent with the definitions, prevailing inferences from formal axioms to informal definitions in natural language.

Extendibility: The ontology design should foresee the necessity of defining new or special use terms in such a way that existing definitions are not modified.

Minimal encoding bias: The conceptualisation should be made at a knowledge level, without regarding on a specific symbol level or implementation language.

Minimal ontological commitment: Ontologies are designed for knowledge sharing by defining ontological commitments. However, those commitments should be minimal to allow the actors involved to instantiate the terms as required.

Not all criteria can be meet when designing an ontology. It is necessary to establish trade–offs between them [Gru93b] and to compromise between design and application [BAT97].

Moreover, all the actors involved should agree on how the vocabulary representing the conceptualisation is to be used. The term *ontological commitment* is used to describe "agreements about the objects and relations being talked about among agents, at software module interfaces or in knowledge bases" [Gru93a].

However, several problems arise when trying to adjust to ontological commitments. On the one hand, different objectives among the ontology definers, application developers and application users. On the other hand, conflicts appear to preserve ontological commitments when ontology evolution is required [NF05].

The more specific the domain to be modelled by the ontology, the more the ontological commitments [CJB98]. In other words, when moving from top–level ontologies to lower–level ontologies or taxonomies of a domain, more ontological commitments for the specific domain are needed [CJB99].

4.5 Roles and Applications of Ontologies

The roles that an ontology can play are still under development, as research on ontology evolves. Insofar, roles of ontologies described in the literature stress the benefit of knowledge sharing and acquisition. Based on [VB96], [JU99], [CJB99], [NM01], [WVV$^+$01], [VS02], [Kit06], a summary of such roles is provided:

1. Neutral authoring: an information artifact developed by an author in a single language to be used in several target systems.
2. Ontology as specification: in a given domain an ontology is created to support the development of software.
3. Knowledge sharing: the ontology is used to share vocabulary and terms among different users being them either persons or machines.
4. Ontology–based search: the ontology is used to look for information in a repository.
5. Knowledge acquisition: the ontology is used to understand the terms in a domain upon a common and agreed understanding.
6. Clarify the structure of knowledge: performing an ontological analysis of a domain allows to define an effective vocabulary of such domain and the underlying conceptualisation.
7. Reuse of knowledge: the ontology enables the reuse of knowledge to build new applications.
8. Integration of heterogenous information sources: ontologies are used to support the information integration task.

Ontology development initially addressed designing broad and general applications such as communication, interoperability or design of reusable components [UG96]. The tendency nowadays seems to be the design of ontologies for specific applications, with a particular domain or application in mind. Medicine, E–commerce, Semantic Web, Enterprise Modelling, Maintenance, Engineering, Knowledge Management, just to mention a few, are domains addressed by ontology developers.

It is not the attempt of this section to provide a thorough and detailed description of available applications. For interested readers excellent reviews are provided in [KMAM00], [GPFLC04b], [SSSS01]. The scope is narrowed in following chapters on how ontologies have been designed for related applications within the research focus of interest.

Chapter 5

METHODOLOGIES, TECHNIQUES, LANGUAGES AND TOOLS

5.1 Introduction

This chapter reviews the literature on methodologies, techniques, languages and tools for ontology development. Attention has been paid both to ontological engineering and to software engineering approaches. The first one, ontological engineering, has been widely used since the early ontology developments. The latter, software engineering, has evolved to support ontology development requirements.

5.2 Ontological Engineering

Ontological Engineering refers to the “set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies” [GPFLC04b]. Ontological Engineering is a research methodology which gives us the design rationale of a knowledge base, conceptualisation of the world of interest, semantic constraints of concepts together with sophisticated theories and technologies.

In the following sections, different techniques, methodologies and languages traditionally used by the ontology developers are introduced. Only those methodologies and methods used to develop ontologies from scratch are considered. There are methods for ontology re–engineering, learning, merging, alignment and evaluation, which are described in [GPFLC04b] that are not going to be considered here.

5.2.1 AI–based Techniques

To develop ontologies, three different AI techniques have been used:

First–Order Logic : This technique allows to represent classes, relations, functions, formal axioms and instances [Gru93a]. It is a traditional AI technique for knowledge representation, useful to avoid the incomplete and ambiguous expressions used in natural language. Knowledge is written in the form of predicates and rules. Reasoning is made by using them combined with rules of inference.

Description Logics : This technique allows to represent classes, roles and individuals (instances and properties) by using two elements [BHS07]: the TBox and the ABox. The first one contains terminological knowledge by using declarations that describe general properties of the ontology concepts. The second one, ABox, contains assertions belonging to individuals (instances).

Frames : This technique defines frames (concepts and classes), their instances (objects and individuals), their properties, and the relationships among them [Gru93a].

5.2.2 Methodologies and Methods for Ontology Development

The methodologies and methods traditionally used to develop ontologies are summarised in this section. Methods and methodologies have evolved as the research on ontologies matured. Typically, a particular research project can develop its own project methodology, which is no further used within the ontology community. Hence, only well–defined and established methodologies and methods are presented.

- The Cyc Method [LG90]: it is a method used to build a common sense knowledge database, CycKB, which could be used to implement intelligent systems. A language, CycL is used to implement assertions, rules or common sense ideas. The project was started in 1984 by Doug Lenat as part of Microelectronics and Computer Technology Corporation. AI strategy Cyc is a registered trademark owned by Cycorp, Inc. in Austin, Texas. The original knowledge base is proprietary, but a smaller version of it, OpenCyc, is provided with an open source license. More recently, Cyc has been made available to AI researchers under a research-purposes license as ResearchCyc. The method consists of three different stages: manual coding of pieces of knowledge, knowledge coding by tools that analyze natural language, and knowledge codification.
- Uschold and King's method [UK95], [UG96]: they proposed a method from their experiences in the development of the Enterprise Ontology as part of the Enterprise Project (Fig.5.1). The method consists of four processes: (1) to identify the purpose and the scope; (2) to build the ontology, divided into ontology capture, coding and integration of existing ontologies; (3) to evaluate the ontology, and (4) to document the ontology. It is not a full methodology as it does not specify the methods and techniques to be used, or the order to be followed.
- Grüninger and Fox's methodology [GF95]: they develop a method to build the TOVE ontology as part of the TOVE project within the enterprise domain (Fig. 5.2.

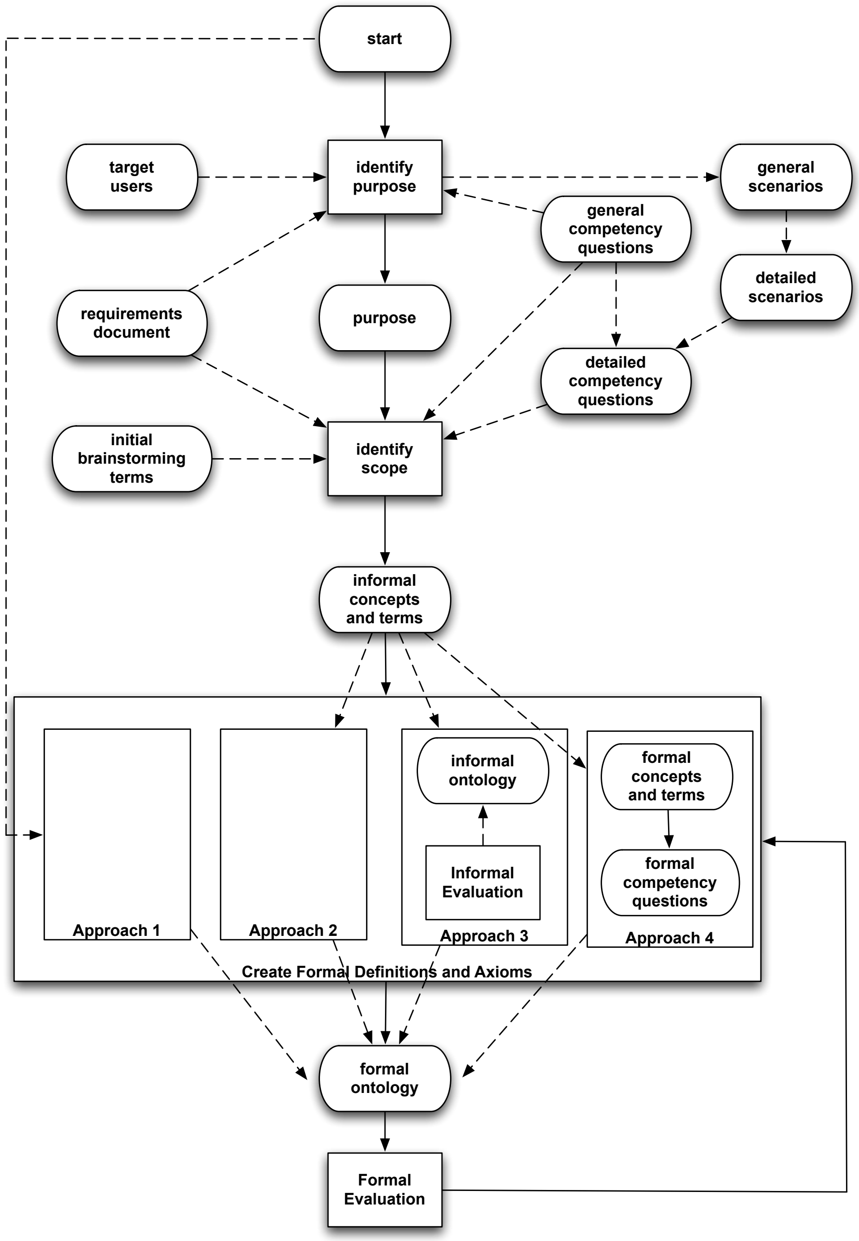


Figure 5.1: Uschold and King's Method (adapted from [UK95])

The methodology consists of six processes: to identify scenarios, to elaborate competency questions in an informal way, to specify the terminology using first order logic, to write down competency questions in a formal way, to specify axioms using first order logic, and to specify completeness theorems.

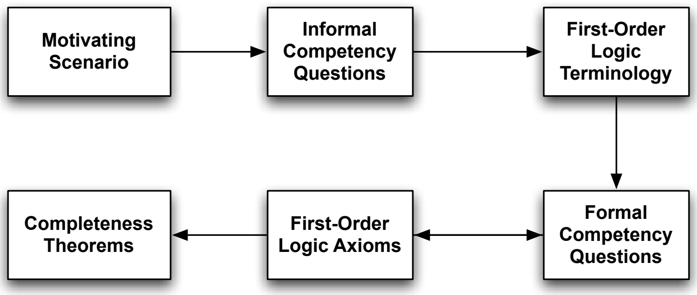

Motivating Scenario
Informal Competency Questions
First-Order Logic Terminology
Formal Competency Questions
First-Order Logic Axioms
Completeness Theorems

Figure 5.2: Grüninger and Fox's Methodology (adapted from [GF95])

- Noy's methodology [NM01]: it is a methodology proposed by Noy and colleagues. The processes considered are: to determine the domain and scope of the ontology; to analyze the possible reuse of other ontologies; to build the terminology; to define ontology elements (classes, their hierarchy, properties, their values, instances).

- KACTUS [BLC96]: it was developed within the Esprit KACTUS project aimed at the role of ontologies to support knowledge reuse. It consists of the following processes: to specify the application; a preliminary design based on top–level ontological categories, and the refinement of the ontology.

- METHONTOLOGY [FLGPJ97] : this methodology was developed by the Ontology Group at the Universidad Politécnica de Madrid, Spain. It allows to build ontologies either from scratch or reusing existing ones, based on evolving prototypes. Processes considered are both technical (ontology development) as well as management and support (Fig. 5.3).

  The ontology development process itself is divided into specification, conceptualization and implementation activities. Possibly, the most relevant one is the conceptualization when concepts, relationships, attributes, axioms, rules and instances are defined. It is one of the few methodologies to address the entire life cycle of ontology development (Fig. 5.4).

- SENSUS–based method [SRKR97]: it is a top–down approach to derive domain specific ontologies from huge ontologies. The starting point is the huge ontology, SENSUS, from which a domain ontology can be built. This method encompasses five processes: to manually link the seed terms to SENSUS; to add paths to the root; to add new domain terms, and to add complete subtrees.

- On–To–Knowledge [SSSS01]: this methodology was developed as part of the On–To–Knowledge project. It consist of the following processes: a feasibility study; a

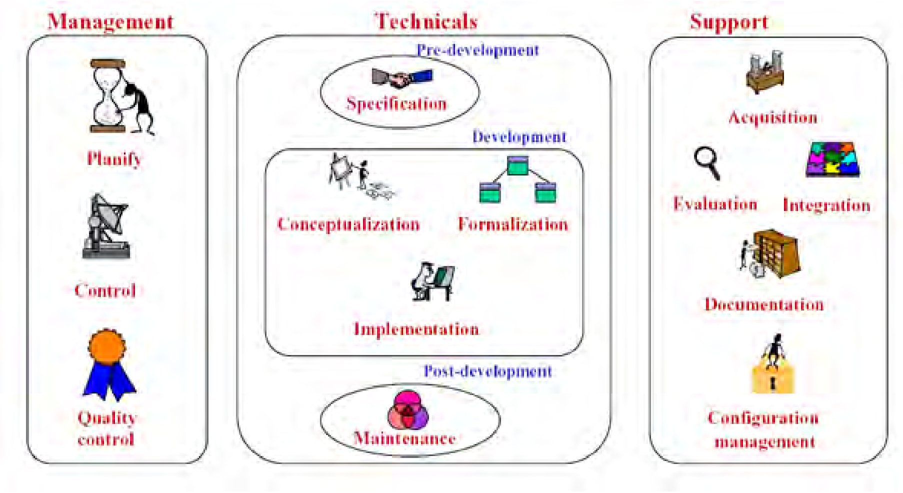


Figure 5.3: METHONTOLOGY Ontology Development Process (from [GPFLC04a])

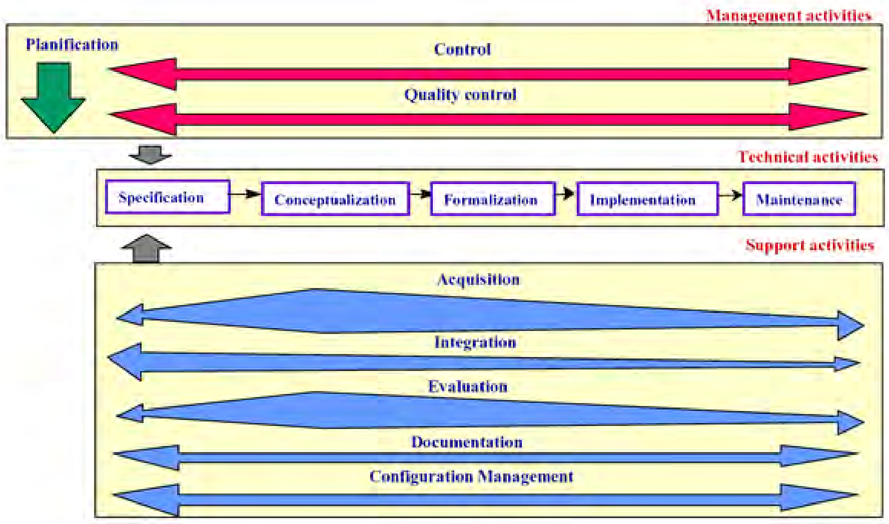
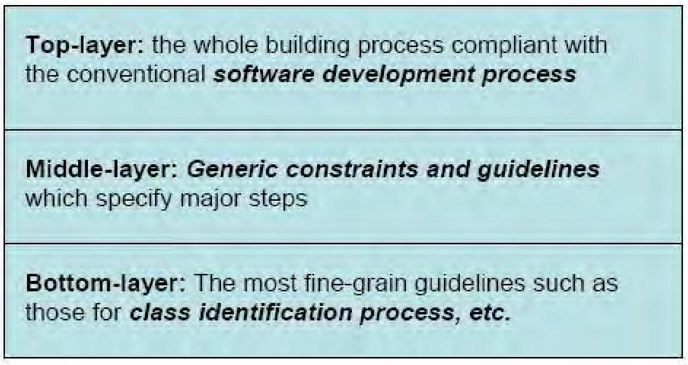


Figure 5.4: METHONTOLOGY life cycle (from [GPFLC04a])

kick–off; a refinement process; the evaluation of ontologies an associated software, and ontology maintenance.

- Three–layer model of guidelines [Miz04]: although not a methodology in the strict meaning of the term, a three–layer model of guidelines is proposed as shown in Fig. 5.5. A detailed list of guidelines are provided, with the main focus being on the middle and bottom layers.



Top-layer: the whole building process compliant with the conventional ***software development process***

Middle-layer: ***Generic constraints and guidelines*** which specify major steps

Bottom-layer: The most fine-grain guidelines such as those for ***class identification process, etc.***

Figure 5.5: Three–layer model of ontology building methodology (from [Miz04])

For a more extent description and comprehensive comparison of the aforementioned methodologies see [GPFLC04a].

5.2.3 Languages

Ontology languages are formal languages used to construct ontologies, by encoding both the knowledge about a specific domain as well as the rules and axioms that support the processing and reasoning of that knowledge. Good reviews and comparison of existing languages can be found in [Miz04], [GPFLC04b].

Existing ontology languages can be classified as follows:

- Traditional ontology languages: they are AI–based languages developed using the techniques described in Sec. 5.2.1.
  1. Frame–based: FLogic [KLW95], and OKBC (as a protocol to allow access to knowledge representation systems using primitives based on frames) [CFF$^+$98].
  2. Description–logic based: LOOM [Mac], [Bri93], and OWL [SWM04].
  3. First–order logic based: CycL [Cyc], Ontolingua and KIF [FFR96].
- Ontology Markup Languages: they are based on markup languages such as HyperText Markup Language (HTML) [RHJ99], and Extensible Markup Language (XML) [BPSM$^+$06]. These languages were developed not so much for ontology development but for data presentation and exchange on web services.

1. HTML–based: SHOE (later adapted to be XML–based).
2. XML–based: XOL, RDF(S), OIL, DAML+OIL, and OWL .

5.2.4 Ontology Development Tools

A variety of ontology development tools and suites have been developed to ease the design and definition of ontologies. Some of them strongly rely on a specific language, whereas others were developed to cater for several ontology languages.

1. Language–dependent: Ontology tools that depend on a specific ontology language to develop their knowledge model (indicated in brackets). Examples are Ontolingua Server (Ontolingua) [FFR96], OntoSaurus (LOOM) [ont], and WebOnto (OCML) [DMC99].
2. Language–independent Tools and environments are characterized by easy extensibility and integration with other applications, as well as their language independency to build the knowledge model (exception made with KAON). The ontology development environment integrates an ontology editor with other tools, and usually they are capable of supporting several ontology languages. Examples are Protégé [HJM$^+$07], WebODE [AV00], OntoEdit [SS01], and KAON [GSV04].

5.3 Software Engineering–based Approach to Ontology Development

A new way to understand ontology development is by using what could be considered inherited languages and techniques from Software Engineering. Although not developed with a focus on ontologies, software engineering practitioners have adopted ontologies as a common element for software development. As a consequence, languages and techniques have evolved to cater, totally or partially, for ontology development requirements.

The following sections describe both the languages, methodologies and tools available to develop ontologies by using software engineering principles.

5.3.1 Languages

5.3.1.1 Unified Modeling Language (UML)

The Unified Modeling Language (UML) [Obj09a], [Obj09b] is a language for specifying, visualizing, and documenting software systems, as well as modeling business and other non–software systems.

UML has features to describe specific views of a model using graphical representations, called UML diagrams. UML 2.2 has fourteen types of diagrams (Fig. 5.6), divided into two categories (structural and behaviour). Structure diagrams show the static structure of the objects in a system. Behaviour diagrams show the dynamic behaviour of the objects in a system, including their methods, collaborations, activities, and state histories. Within the behaviour diagrams, there are four interaction diagrams that represent different kinds of interactions.

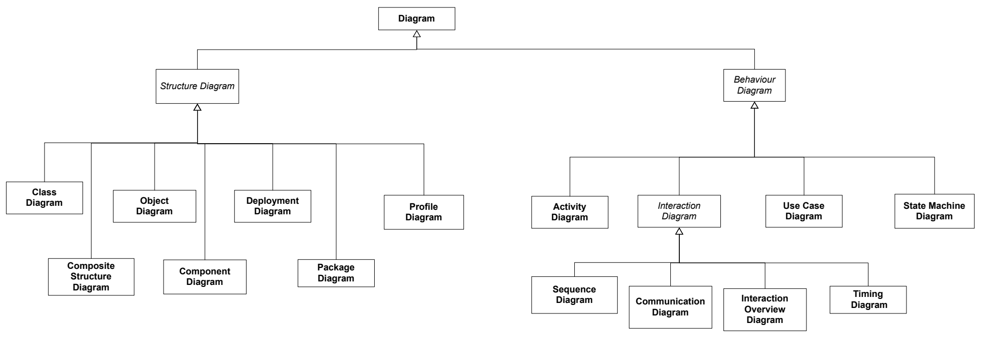


Figure 5.6: Taxonomy of UML Diagrams (adapted from [Obj09b])

UML includes the so–called UML Profiles, which are extensions of UML to adapt the basic UML constructs to specific purposes by using stereotypes, tag definitions, tagged values, and constraints.

UML and Ontologies

The research of the use of UML for ontology development falls under two distinctive groups. The first group includes the initial and recent attempts to extend UML to cater for ontology development requirements, especially when it comes to formal definitions and semantics. The second one, the application of UML without changing its core to specific applications. Both groups are described in the following sections.

Extending UML for Ontology Development

UML was first used by Cranefield [CHP01] based on the similarities among UML and ontologies concepts. The main ideas are:

- UML class diagrams can be used to model the taxonomy of concepts and relationships in an ontology.
- UML object diagrams can be used to model instances of an ontology.
- Object Constraint Language (OCL) [Obj06b] can be used to specify constraints of an ontology.

However, using UML for ontology development was not exempt of problems. Firstly, there is a lack of a formal definition in UML. Generally speaking, UML semantics is defined using a combination of a metamodel, constraints in a semi–formal language called Object Constraint Language (OCL) [Obj06b] and descriptions in plain English. Cranefield tackled these problems by integrating UML and RDF(S), by using a group of standards (XMI, UML tools, RDF(S) and eXtensible Stylesheet Language Transformations

(XSLTs) to transform UML XMI into Java classes and interfaces and RDF(S)). Secondly, even with this approach, handling attributes and associations between UML and RDF(S) led to several problems.

A further attempt was carried out by Baclawski and colleagues, who developed two different proposals. In the first one, they proposed to extend the UML metamodel by using RDFS and DAML+OIL languages [BKK$^+$02]. Their approach was to define mappings among UML and DAML+OIL (a traditional language used to develop ontologies). Their findings highlighted some incompatibilities among UML and DAML+OIL (e.g. monotonic, metalevels, modularity, properties, class constructors, cardinality constraints, subproperties and namespaces as summarised in [GDD06c]), which led them to propose a modification of the UML metamodel using a profile based on their metamodel.

However, the same authors realized that this solution was to some extent difficult to implement, as an extended UML metamodel will imply having concepts hardly used in ontology development. Therefore, they proposed an independent MOF–based language for Layer 3, called Unified Ontology Language (UOL) which addressed the notions of package, class, binary association, generalization, attribute and multiplicity constraints [BKS$^+$02].

Another approach focus on the use of UML for the Semantic Web. Falkovych and her colleagues [OK03] have proposed a transformation–based approach to extract ontologies from UML models. They developed rules to transform UML models into DAML+OIL ontologies. Nevertheless, this solution is not free from problems such as the impossibility to define properties and other primitives in UML [GDD06c].

A last proposal made by Gasevic and colleagues [GDD06c] have defined a metamodel based on OWL Full [Dev02], an ontology UML profile (OUP) [Dju04], [GDD06d] and the required transformations [GDDD04] , [GDD06a] within the MDA context. An editor called AIR has been developed to help in the ontology development [GDD06b]

UML usage for ontologies

UML is used in different domains to describe ontologies, mainly within the agent systems and the software engineering communities. These two groups are used to software techniques, and therefore, they were willing to analyze the use of such methods to develop new elements as ontologies, which prove to help in the two domains when it comes to knowledge management, communication, agents interaction and other capabilities.

Within the first group, agent systems, despite the aforementioned drawbacks, Cranefield has proposed the use of an UML–based ontology to define concepts and agents operations when agents are using a communication protocol [CP02a]. He also defined a UML profile for ontology modeling focusing on agent systems features and performances [CP02b].

UML has also been used within the agent community to model agent–oriented artifacts. An UML profile for agent–oriented software was proposed, introducing stereotypes to suit agent–based systems requirements [BP02]. Two new models called ontology and architecture models were defined, and further formalized as ontology and architecture diagrams using the newly defined stereotypes [BP01]. A framework, ParADE (Parma Development Environment) was implemented to support the introduced UML notation. An additional attempt was made to use UML for domain ontology modeling in a multi–agent system, by extending UML Class Diagram to support both communication among agents and real–time features [LLS03].

A second group is that belonging to software engineering developments. UML has been used to develop ontologies to support different software engineering activities or to define ontologies as software artifacts to be used in software–based systems (see chapter 4). An additional example is the method proposed by [NCL06], to extract ontological elements for a domain ontology, expressed in OWL, from existing UML models by using transformation rules.

Within the robotics domain, an ontology was developed to describe, upgrade and share knowledge regarding the environment for a robot fleet [CCPR02]. The ontology was developed in two stages: Ontology Identification Phase and Ontology Description Phase. As a result of the first one, a set of ontologies representing a particular viewpoint of the environment is represented as an Ontology Identification Model. The second one leads to an Ontology Description Model which moves from conceptualization to implementation.

Other uses concerning its application as ontology modeling language include research conducted within the industrial domain such as remediation methods for petroleum contaminated sites [WCH02], [Cha04]. A 2002 summary of existing applications of UML to ontology development was described in [KCL$^+$02].

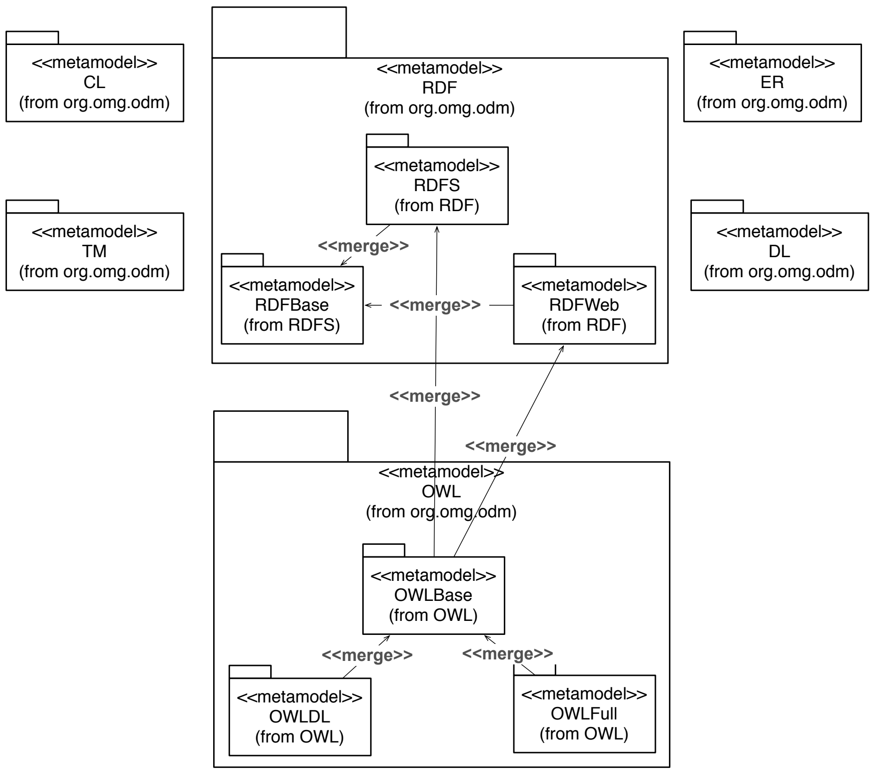
5.3.1.2 Ontology Definition Metamodel (ODM)

Some of the former attempts to expand the features of UML for ontology development have led to the definition of the Ontology Definition Metamodel (ODM) [Obj09c].

It is an OMG specification that

> defines a family of independent metamodels, related profiles, and mapping among the metamodels corresponding to several international standards for ontology and Topic Map definition, as well as capabilities supporting conventional modeling paradigms for capturing conceptual knowledge

The set of MOF 2 metamodels include two for formal logic languages (Description Logics, and Common Logic); three for descriptive representations (RDF Schema, OWL and Topic Maps), and two additional metamodels for software engineering approaches (UML 2.0 and Entity Relationship). EMOF (Essential MOF) [Obj06a] was used to define the ODM metamodels for easiness of definition and use.



<<metamodel>> CL (from org.omg.odm)
<<metamodel>> RDF (from org.omg.odm)
<<metamodel>> ER (from org.omg.odm)
<<metamodel>> RDFS (from RDF)
<<metamodel>> TM (from org.omg.odm)
<<metamodel>> DL (from org.omg.odm)
<<merge>>
<<metamodel>> RDFBase (from RDFS)
<<merge>>
<<metamodel>> RDFWeb (from RDF)
<<merge>>
<<merge>>
<<metamodel>> OWL (from org.omg.odm)
<<metamodel>> OWLBase (from OWL)
<<merge>>
<<merge>>
<<metamodel>> OWLDL (from OWL)
<<metamodel>> OWLFull (from OWL)

Figure 5.7: ODM Metamodels: Package Structure (from [Obj09c])

To allow the exchange among the defined metamodels, ODM provides both UML profiles, i.e. extensions to UML notation and tools for ontology modeling, and mappings.

Currently, ODM includes three profiles to be used for ontology descriptions using RDF, OWL and Topic Maps. UML Profile for RDF and OWL has been designed to allow for ontology development through reusing UML notation, taking into account:

- Reuse UML constructs when they have the same semantics as OWL. If not, use of UML stereotypes consistent with OWL semantics.
- Use UML 2.0 notation. If not, follow stereotype notation as described in SysML.
- Use of Foundation Ontology (M1) for RDF and OWL (as defined in Appendix A of [Obj09c]).

Additionally, there are available direct mappings (to and from) OWL, UML, and Topic Maps. However, only uni–directional mappings to Common Logic are considered in the current version. Mappings are regarded as informative, not normative, as ODM metamodels have been defined broad enough to allow different uses for different domains. Further developments will need to define specific constraints.

5.3.1.3 SysML

SysML is a general–purpose modeling language especially designed for systems engineering applications [Gro08a]. It was designed to support the specification, analysis, design, verification and validation of complex systems, that can consist of hardware, software, information, processes, personnel, and facilities.

It is considered to be particularly effective when specifying requirements, structure, behavior, allocation, and constraints on system properties during engineering analysis.

The underlying motivation was to develop a language to cater for system engineering, capable of managing the complexity inherent to such systems, improving communications among stakeholders, and modeling at multiple levels [FMS06].

SysML language reuses a subset of UML 2.1 [Obj09b] (UML4SysML), providing also extensions to satisfy the systems engineering applications (SysML Profile), as shown in 5.8.

UML4SysML is a metamodel package that merges the set of reused UML metaclasses. The SysML profile specifies the extensions to UML. The SysML profile package (see Fig. 5.9) contains the metaclasses extended in SysML packages (by using stereotypes, diagram extensions, and model libraries) as well as all the metaclasses reused as–is from UML (by referencing the UML4SysML package). It also imports the Standard Profile L1 from UML to use its stereotypes (see Fig. 5.10).

The diagrams used by SysML are shown in the SysML diagram taxonomy (Fig. 5.11). Some diagrams remain from the original UML 2.0, whereas others have been modified

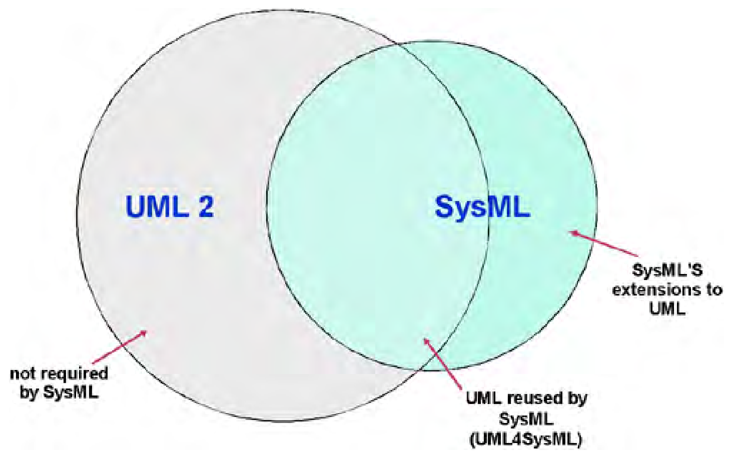


Figure 5.8: Relationship between SysML and UML (from [Gro08a])

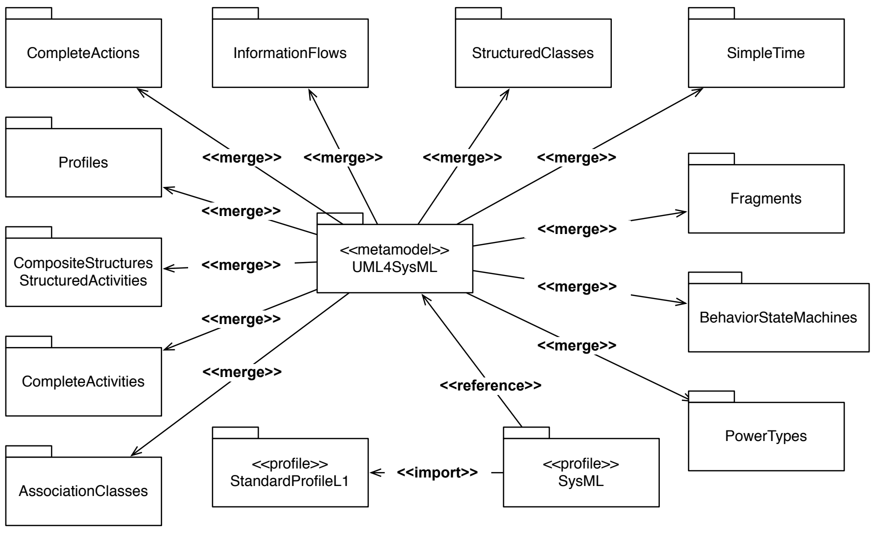


Figure 5.9: SysML extensions to UML (from [Gro08a])

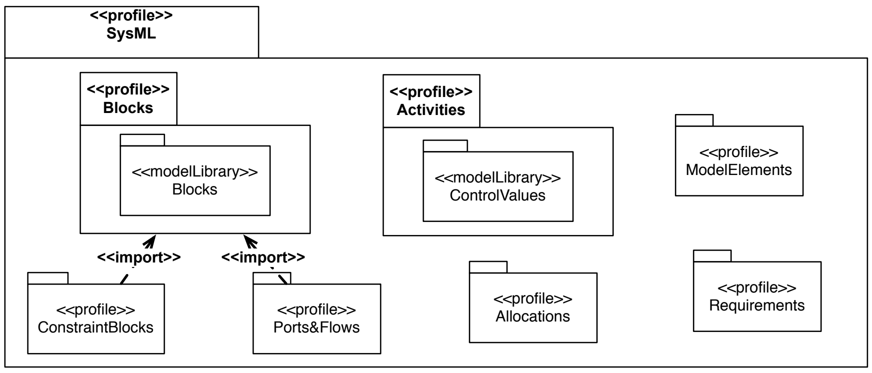


Figure 5.10: SysML packages structure (from [Gro08a])

or are completely new to SysML.

A first group, Structure Diagrams (Package Diagram, Block Definition Diagram, Internal Block Diagram, and Parametric Diagram), describe static and structural properties of the system, by using structural constructs either imported from UML4SysML or described in the SysML profile packages (ModelElements, Blocks, Ports and Flow, and ConstraintBlocks).

A second group, Behavior Diagrams (Activity Diagram, Sequence Diagram, State Machine Diagram and Use Case Diagram), describe dynamic and behavioral properties of the system, by using behavioral constructs either imported from UML4SysML or described in the SysML profile Activities package.

Additional crosscutting constructs, i.e. constructs that can be applied both to structure and behavior, are defined in the SysML profile packages (Allocations and Requirements), as well as in Profile and Models Libraries used to customize SysML for specific applications.

SysML supports both model and data interchange via XMI (XML Metadata Interchange) and the AP233 standard (ISO 10303-233 Application Protocol: Systems Engineering and Design).

SysML and Ontologies

There are not major applications of SysML to develop ontologies, as it has been recently adopted as specification. Available literature focuses on the features required for the new language to fulfill Systems Engineering requirements ([HTM05], [Boc06]) or attempt to apply the adopted specification for case studies (software verification and validation [ADHJ06], embedded systems [Bal06], model–driven development applying SysML within the RUP Software Engineering framework [BBCM06], product lifecycle management [Boc05]) or just mention SysML for further research within software engi-

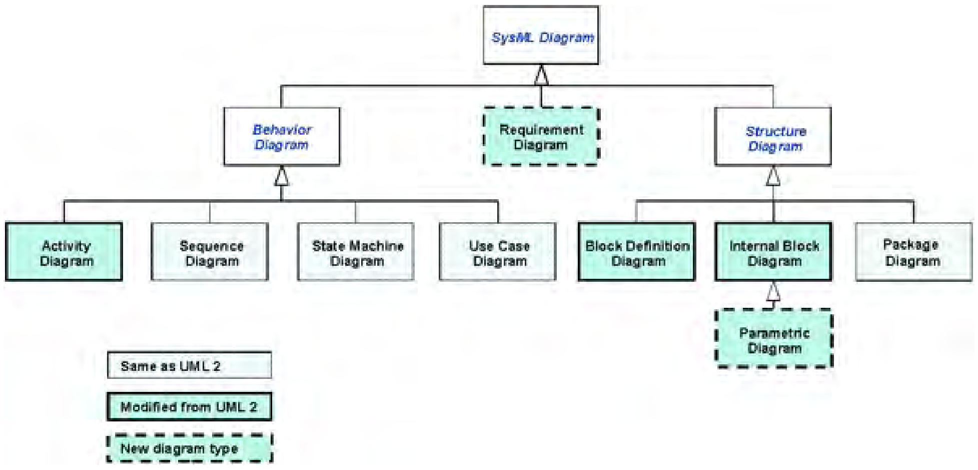


Figure 5.11: SysML Diagram Taxonomy (from [Gro08a])

neering [PB06].

Nevertheless, there is an underlying feeling that Software Engineering processes will benefit from SysML's features. Therefore, some ontologies already developed using UML will migrate to SysML to address such circumstance.

As a first example, an ontology has been developed using UML for a real–time credit–approval system [AT06a]. The ontology has four major components: architecture assets (subsystems, components, and interfaces), architecture decisions, stakeholder concerns, and architecture roadmap (which decisions to be implemented and when). The authors consider that the ontology would benefit when it comes to modeling the stakeholders concern by using SysML constructs such as requirements expressed in a Requirements Diagram.

A further attempt to explore the suitability of SysML is described in [OH05]. Ontologies were developed to integrate data and information for a Maritime Domain Protection System of Systems. The authors considered that using SysML to model stakeholders and systems views will improve the ontology development, and therefore, the overall model of the system.

5.3.2 Methodologies and Methods

REFSENO [TvW98] is a representation formalism and methodology to develop software engineering ontologies.

The formalism provides constructs for the traditional ontology elements: classes (called concepts), properties (called terminal concept attributes), relations (called non-terminal concept attributes), and instances. They are specified as tuples, and defined in tables.

Formulas are provided to define similarity functions, inferences, preconditions, and assertions. REFSENO allows for similarity–based retrieval. It also incorporates integrity rules (cardinalities and value range to be fulfilled by attributes, assertions, and preconditions).

As a methodology, REFSENO is considered to be an extended adaptation of Methontology, where the main steps are:

1. Planning
2. Specification of ontology requirements
3. Conceptualization (similar to the design phase for a software system development, although here it is the ontology itself)
4. Implementation (representation using the formalism and storage of the conceptualization using software tools)

Some benefits that could be obtained by using REFSENO for software engineering ontologies are [RVPG04]:

- Specifically designed for software engineering.
- Use of alternate representations.
- Clear terminology, with a distinction between conceptual and context–specific knowledge.
- Use of consistency criteria for ontology consistency checking.
- Object–centered, which allows for the use of UML as modeling language.

It has been used for developing ontologies related to software engineering maintenance, as described in [RVPG04], [GPR04].

Other methods have been commented upon by [Dev01], regarding the use of software engineering methods initially not defined to develop ontologies, such as object–oriented techniques and pattern design. Both software enginering methods share, to some extent, the need of an analysis and later design of concepts, relationships and properties within the domain. Although it is not the case that the result is called an ontology.

When it comes to the particular case of domain ontologies, some methods have been described. A first example, the IDEF5 ontology development process consisting of five activities (organizing and scoping; data collection; data analysis; initial ontology development; ontology refinement and validation). The activities are listed sequentially, overlaps however exist among them. The methodology is complemented by the definition of the IDEF5 Elaboration Language for knowledge capture and definition [IMDt94].

5.3.3 Tools

A review of existing UML– and SysML–based tools is provided here. The aim was to analyse the suitability of the different tools for ontology development. As a result, it was found that the SysML was provided as an additional plug–in within the UML tools. The SysML plug–ins did not contain any ontology–oriented feature, despite the efforts to use this language in ontological engineering.

Some of the UML tools were primarily tools for UML–based development, not so much ontological engineering. Nonetheless, they can be used to develop an UML–based ontology by using class diagrams (to represent the ontology concepts, and their relations) or object diagrams (to represent instances). There are other UML–based tools that contained some kind of plug–in to allow mapping an ontology, to be expressed on a traditional ontological language such as OWL.

5.3.3.1 UML–based tools

- DAML–UML Enhanced Tool (DUET) (http://projects.semwebcentral.org/): it is a tool developed as part of Components for Ontology Driven Information Push (CODIP) program, which aims at the development of reusable software components to build information distribution systems. DUET provides a UML environment for OWL, by using a UML profile for OWL. Version 0.3 is an add–in to Rational Rose or ArgoUML. A later version is a standalone application, geared to support ODM standard. It is a work in a progress, with versions updated to incorporate new features and standards.

- Visual Ontology Modeler (VOM) by Sandpiper (http://www.sandsoft.com): it is a visual application to build component–based ontologies, based on UML. The software provides ontology maintenance, as well as, the use of existing UML business data. VOM implements Sandpiper's UML Profile for Knowledge Representation, an extension of UML to model frame–based knowledge. It also includes libraries to represent the IEEE Standard Upper Ontology (SUO) and other international standards. It is an add–in to IBM's Rational Rose Software.

- Protégé UMLBackend (http://protegewiki.stanford.edu): Protégé is a free, open source ontology editor and knowledge–based framework. It supports frame–based ontologies and semantic web ontologies based on OWL. Although originally not a UML tool, it provides a UML back–end plug–in for Protégé 2.1.2, that enables the exchange of ontologies and UML class diagrams. This back–end allows Protégé to be used with traditional software tools.

- MagicDraw by No Magic, Inc. (http://www.magicdraw.com): it is a modeling tool that support UML 2.0. It is written on the Java platform, which allows it to be run on any major operating system. It contains different Profiles to address specific modeling requirements. There are different plug–ins to interact with IBM's Rational Rose and other software.

- Poseidon for UML by gentleware (http://www.gentleware.com): it is a UML–based tool that can be used to develop ontologies. Its current version supports UML 2.0.

It is written on the Java platform, which enable its use on any major operating system. There are different editions (community, standard, professional and embedded) to address different modeling and development requirements.

- Enterprise Architect by SparxSystems (http://www.sparxsystems.com): it is a UML modeling tool for Windows platform. It claims to support UML 2.1, UML 2.0 Profiles. It also provides support for MDA transformations.

- ARTiSAN Studio by ARTiSAN Software (http://www.artisansw.com/products/): it is an integrated suite of UML modeling tools to develop technical systems. It supports UML 2.0. It has been develop for Windows operating system (either single– or multi–user).

- Rational Rose by IBM (http://www-306.ibm.com): it is a set of UML and model–driven tools (Developer for Java, Developer for Visual Studio, Modeler, Developer for UNIX, Enterprise, and Technical Developer) to cover from modeling to development, through design. It only supports UML 1.4.

- Rational Software Architect by IBM (http://www-306.ibm.com): it is a model–driven development tool that supports UML 2.0. It could be used under Windows or Linux operating systems. It enhances the features available in Rational Software Modeler and Developer.

5.3.3.2 SysML–related tools

Most of the available tools already developed to support UML 2.0, are being extended with plug-ins to allow the use of the SysML language and capabilities.

- MDG Technology for SysML by Sparx Systems (http://www.sparxsystems.com): it is a plug–in for their UML modelling platform, Enterprise Architect, that supports the OMG SysML 1.0 specification. It allows the definition and use of the SysML diagrams, with an emphasis on requirements and allocations.

- Requirements Traceability and Modeling with SysML and Reqtify by ARTiSAN Software (http://www.artisansoftwaretools.com): it provides support mainly for requirements definition and traceability as part of their ARTiSAN Studio 6.1. It uses the ARTisAN's SysML Requirements Profile which supports the OMG SysML specification. It is unclear wether other SysML constructs are currently supported.

- SysML Toolkit by EmbeddedPlus (http://www.embeddedplus.com): it is a plug–in to support SysML within the IBM Rational Software Development Platform (RDSP) developed by EmbeddedPlus Engineering as third party for IBM Rational. It allows the sharing of model artifacts between system and software engineers, model validation and, easy upgrade of Rational Rose models. It claims to support SysML 1.1.

- IBM Rational Rhapsody by IBM (http://www.telelogic.com): it is an environment that supports UML 2.1 and SysML for systems engineering. As SysML features, it supports requirements management by using SysML Requirements, Use Case, Sequence, Activity, and Statecharts Diagrams. Requirements traceability are also supported by using the Rhapsody Gateway.

- SysML plug–in by No Magic, Inc. (http://www.magicdraw.com): it is a plug–in for MagicDraw, an UML–based tool. The plug–in supports SysML 1.2. SysML modeling is made by using the SysML perspective, which is the specific mode of the MagicDraw user interface for SysML modeling.

- Visio Stencil and Template for SysML 1.0 by Microsoft (http://softwarestencils.com): it is just a template that supports the SysML symbols to be used in Microsoft Office Visio.

Chapter 6

ONTOLOGIES FOR AUTONOMOUS SYSTEMS ENGINEERING

6.1 Introduction

This chapter reviews the state of the art on how ontologies have been developed and used in the two interrelated domains the research addressses. On the one hand, the domain of autonomous systems. On the other hand, the domain of software engineering.

Firstly, the review addresses the advantages of the use and development of ontologies for autonomous systems. Ontologies have provided useful mechanisms for autonomous systems' design and operation. Within this context, a summary of the different roles ontologies can play, with examples on which one of the aforementioned benefits achieves. However, the application of ontologies is not exempt of difficulties, which are discussed in the final section of this chapter. It would have been desirable to analyse the features of the different ontologies found throughout the review. However the ontology descriptions lacked of the necessary detail to carry out any comparison, focusing primarily on the role of the ontology within the particular autonomous system application.

The rest of this chapter summarises existing applications of ontologies for software development, both as domain ontologies and software artifacts. Software development has benefited from the use of ontologies, as they provide a common understanding of the concepts involved in the software process. Ontologies also ease the integration issues that usually appear while developing software applications. To gain some knowledge on how ontologies are used for software engineering purposes, we conducted a thorough review of existing research on this field that is explained in the last sections of this chapter.

6.2 Ontologies for Autonomous Systems

The approach to use ontologies within the domain of autonomous systems is closer to the computer science viewpoint of *normalisation or knowledge representation* based on a specification of a conceptualization, as opposed to defining the meaning of terms or to understanding the world [Roc06]. The underlying idea is to define concepts based on a computational language, to be manipulated by actors in the system. Ontologies are generally used as a representation–based mechanism based on a computational language, to describe the different entities participating in the design and operation of the autonomous systems: different domains, the environment, the objects the systems interact with, the possible actions to be taken, resources to be considered, etc. How the meaning of concepts defined in the ontologies are understood by different components of the autonomous system is not so much the focus as how the different actors can use a common conceptualisation of the problem under scrutiny, to come up with a possible solution.

There is still the controversy whether ontologies describe mental concepts or mind–independent entities in the world. The term concept itself, for engineering purposes, refers to entities that are created by modelers to represent real entities in the world. Concepts are computational elements that exist through their representations in software, such as UML diagrams or other ontology–related languages.

6.3 Advantages of Ontologies for Autonomous Systems

The use of ontologies for autonomous systems' operation has reported a set of benefits that could be summarized as follows [CJB99], [NM01], [WLB04], [SSM$^+$04]:

1. Ontologies clarify the structure of knowledge: performing an ontological analysis of a domain allows to define an effective vocabulary, assumptions and the underlying conceptualization. The analysis also allows to separate domain knowledge from operational or problem–solving one.

2. Ontologies help in knowledge scalability: knowledge analysis can result in large knowledge bases. Ontologies help to encode and manage in a scalable way.

3. Ontologies allow knowledge sharing and reuse: by associating terms with concepts and relationships in the ontology as well as a syntax for encoding knowledge in them, ontologies allow further users to share and reuse such knowledge.

4. Ontologies increase the robustness: ontological relationships and commitments can be used to reason about novel or unforeseen events in the domain.

5. Ontologies provide a foundation for interoperability among heterogeneous agents and system's elements.

6.4 Ontology Roles in Autonomous Systems Applications

Several examples have been analysed to exemplify some of the possible uses of ontologies for autonomous systems, understanding as such the different views described in Sec. 3.4. These roles match to a great extent the general ones (see Sec. 4.5) considered for ontologies, regardless of the domain of application. Table 6.1 summarises the different benefits achieved, and the roles each one of the ontologies plays for a particular domain.

Ontology Role	Achieved Benefit	Mobile Robots	Multi–Agent Systems	Autonomic Computing
Clarify the structure of knowledge	Structure of domain knowledge	[SBU$^+$03], [SB04], [SBM05]	[CD00], [MPO05]	[TT03], [SAL06]
Specification	Knowledge scalability	[UPS$^+$03], [WVH04], [PUS]	[CPN00], [CPNH05]	[MSSS$^+$07]
Domain knowledge sharing and reuse	Knowledge sharing and reuse	[SM05], [HB06]	[MPO04], [WVV$^+$01], [BAT97]	[SSM$^+$04]
Ontology–based search	Robustness increase	[BAM$^+$04], [Woo04], [Eps04], [JBK$^+$04]	[LSRH97], [CPL05]	[SASS04]
Integration of heterogenous sources	Interoperability	[CTSB04] , [IBGM08]	[CD00], [MPO05], [DMQ05], [Tam01], [NF05]	[LSAF06]

Table 6.1: Ontologies for autonomous systems: roles and benefits

6.5 Current Research on Ontologies for Autonomous Systems

This section describes in more detail the ontologies considered in the previous section to exemplify the different roles an ontology can play for autonomous systems engineering. The analysis addressed the subdomains with the domain of autonomous systems where ontologies have been developed an applied: mobile robots, agents, and autonomic computing.

6.5.1 Ontologies for mobile robots

Ontologies have been applied for autonomous or mobile robots description and operation, generally as a knowledge representation–based use to characterise the domain, the tasks or the environment the mobile robot is placed in:

- Ontologies to describe the environment: a key point for autonomous robots is to have a precise and detailed characterisation of the environment, as knowledge or central repository of the objects in it [SBU$^+$03] or the location they are moving in [SB04], [SBM05]. Mobile robots need to construct and manipulate representations of the surrounding environment built up by means of sensors. Spatial knowledge representation based on ontologies have been used in [UPS$^+$03], [WVH04], [PUS].
- Ontologies to describe or reason about actions and tasks: mobile robots are faced to real–time and complex tasks which might require extremely large knowledge to be stored and accessed.The ontologies have been used to structure this knowledge and its different levels of abstraction [BAM$^+$04], to describe task–oriented concepts [Woo04], as metaknowledge for learning methods and heuristics [Eps04] or to define concepts related to actions, actors and policies to constraint behaviour [JBK$^+$04].
- Ontologies as common conceptualisation: Specialised modules in a robotic architecture communicate using a representation–neutral language based on a common ontology, as they must communicate to perform their functions [CTSB04] or for knowledge modelling through a preliminary ontology representing the intelligent autonomous system for action learning and sharing [IBGM08].
- Ontologies to reuse domain knowledge: ontologies have been used to describe robots as objects by describing its structural, functional and behavioural features [SM05] or to characterise the domain and subdomains of robotics [HB06].

6.5.2 Ontologies for Agents

The use of ontologies in (intelligent) agents addresses the necessity for agents of knowledge share and exchange [TCFW05]. Hence ontologies have been developed with this purpose, playing several roles as gathered in [BAS06].

- Common Access to Information: The underlying idea is to use the ontology to share terms among users, either humans or agents. This approach was implemented as an agent–based platform, ForEV [MPO04], using a multiple ontology approach [WVV$^+$01]. Ontologies and agents technologies have also been combined to solve the semantic heterogeneity problem in e–commerce negotiations and transactions [CD00], [MPO05].
- Ontology–based Search: this is one of the oldest areas in which ontologies have shown their usefullness for agent-based applications [LSRH97], and [CPL05]. Ontologies are used as common metadata vocabularies, to allow document creators to know how to mark up their documents so that agents can use the information in the supplied metadata.

- Reuse of Knowledge: The ontology enables the reuse of knowledge to address the domain knowledge needs of potential new applications. As an example, the PHYSSYS ontology was constructed to describe the knowledge for physical systems. The reusability aspects were addressed by using ontology projections to describe technical components, physical processes and mathematical relations [BAT97].

- Integration of Heterogenous Information Sources: The ontology is used to support the integration information task, such as the definition of an ontology–based agent system related to different domains as described in [NF05].

- Ontologies for Modeling: Ontologies are used to model the concepts the agents need and the internal operations or tasks that agents carry out. As an example, an ontology associated with the FIPA Request Interaction Protocol was defined in [CPN00], [CPNH05]. In this case, concepts referred to message types, the reason for the request and the precondition to be fulfilled.

- Ontologies for Semantic Interoperability: Agents are highly heterogeneous in real applications. Semantic interoperability is defined as "the problem of achieving communication between two agents that work in the same or overlapping domains, even if they use different notations and vocabularies to describe them" [DMQ05]. Common, global or shared ontologies are used to overcome the semantic heterogeneity among agents. Commitments to the shared ontology permit the agents to interoperate and cooperate while maintaining their autonomy. A common ontology built up either by sharing, merging or translating ontologies has been proposed in [Tam01], [NF05].

6.5.3 Ontologies in autonomic computing

Lately, ontologies have been used within autonomic computing developments, as knowledge representation to provide support for information exchange and integration :

- Interoperability between system's elements by providing a shared understanding of the domain in question: autonomic systems require knowledge from different sources to be represented in a common way. An ontology-based knowledge representation that will use data from information models while preserving the semantics and the taxonomy of existing systems. This will facilitate the decomposition and validation of high level goals by autonomous, self-governing components [LSAF06].

- The analysis of the system and the domain: ontological component models [TT03] have been developed to describe the autonomic system, as extension of the Bunge-Wand-Weber (BWW) models. Another example describes how ontologies support self-knowledge (of the system, of the environment) by using object-oriented information and data models augmented with ontologies [SAL06].

- A formalization of shared understanding which allows machine processability: the use ontologies for the representation of knowledge and its integration into a general systems architecture supporting autonomic behavior. The knowledge of domain experts (concerning processes and situations) can be directly encoded in ontologies whose syntax is not to far from natural language [Ceb05].

- An explicit representation of the semantics of data and associated rules: ontologies have been used in correlation engines to support reasoning capabilities, which allow to provide new services, such as verification, justification, and gap analysis [SASS04].
- A reuse of knowledge domain, allowing its extension: when an autonomous system has to operate in a new domain, a related domain model has to be built, most of the time from scratch. Having ontology-based definitions of possible domains will aid to reuse them and to allow autonomous system's operation. The ontology can be extended as required for new functionalities or systems [SSM+04]. This is an important aspect in the ASys research programme.
- Knowledge scalability: ontologies have been used to model context information, which is of high relevance in next generation networks, where users and devices can have different forms of context [MSSS+07].
- Verification: when systems are faced to a high heterogeneity of data and semantics, having ontology axioms and rules to verify the model play a major role [SSM+04].

6.6 Ontologies for Software Engineering

Ontology–driven software development or engineering has been defined as an approach that, based on ontologies, takes into account semantic constraints, adapting in a dynamic way to new constraints [Tan05]. It could be considered a particular case of model–driven software engineering [BCT05], where models are based on ontologies at different levels of abstraction.

Software engineers have paid attention to the development and use of ontologies for software development. On the one hand, ontologies have been used as domain ontologies to describe software engineering processes or technology. On the other hand, ontologies themselves can act as software artifacts to aid and to support software processes.

6.7 Roles of Ontologies for Software Development

Ontologies, within software design and development, can be used with the following objectives according to Ruiz and Hilera [RH06]:

- Specification: ontologies are used to specify either the requirements and components definitions (informal use) or the system's functionality.
- Confidence: ontologies are used to check the systems design.
- Reusability: ontologies could be organized in modules to define domains, subdomains and their related tasks, which could be later reused and/or adapted to other problems.
- Search: ontologies are used as information repositories.

- Reliability: ontologies could be used in (semi)–automatic consistency checking.
- Maintenance: ontologies improve documentation use and storage for system's maintenance.
- Knowledge acquisition: ontologies could be used as a guide for the knowledge acquisition process.

For Hesse [Hes05], two main roles for ontologies have been considered :

- Ontologies for the *Software Engineering Process*: the definition, reuse and integration of software components is aided by the use of ontologies as the conceptual basis.
- Ontologies for the *Software Engineering Domain*: the use of ontologies to describe the structure and terminology of the software engineering domain itself.

6.8 Benefits and Issues on the Use of Ontologies

Ontologies provide benefits regarding the process of software development, which could be summarized as follows [FMR98], [FGD02], [LQL05], [RBF04], [ACGB+06], [RH06]:

- To provide a *representation vocabulary* specialized for the software process, eliminating conceptual and terminological mismatches.
- The use of ontologies and alignment techniques allow *to solve compatibility problems* without having to change existing models.
- Ontologies might help *to develop benchmarks* of software process by collecting data on the Internet and the use of the Semantic Web.
- To allow both *to transfer knowledge* and *to simplify the development cycle* from project to project.
- Ontologies promote *common understanding* among software developers, as well as being used as domain models.
- To ease the *knowledge acquisition process*, by sharing a same conceptualization for different software applications.
- As a mean to allow *to reduce terminological and conceptual mismatches*, by forcing to share understanding and communications among different users during the ontological analysis.
- To provide for *a refined communication* between tools forming part of an environment.
- Ontologies, when as *machine–understandable representations*, aid in the development of tools for software engineering activities.

Although ontologies are considered a useful element within software engineering activities, some issues should still be born in mind when developing ontology–based software development projects [Hes05]:

- The ontology–based approach is adequate for those software development projects that belong to a set of projects within the same domain.
- The ontology–based approach allows to extend the notion of reusability to the modelling phase, not only the usual implementation one. Therefore, ontologies could be considered reusable model components in the system. Model–driven developments can benefit from the use of ontologies as model reuse mechanisms.
- The ontology–based approach affects all the software development process phases, from requirement analysis and domain analysis to integration, deployment and use of the developed software.
- The ontology–based approach allow ontologies to be used to facilitate software development in the long term, as well as addressing interoperability and re–use issues.

Furthermore, ontologies should exhibit some specific properties to facilitate their use within the software engineering community [GPHS06]:

- Completeness: to assure that all areas of software development are covered. It could be achieved by paying attention to the different activities carried out by software development enterprises.
- Unambiguity: to avoid misinterpretations. Ambiguity could be avoided by using both concise definitions of concepts and semi–formal models.
- Intuitive: to specify concepts familiar to users' domain.
- Genericity: to allow the ontology to be used in different contexts. It could be done by keeping the ontology as small as possible, to achieve maximum expressiveness while being minimal.
- Extendability: to facilitate the addition of new concepts. It could be achieved by providing appropriate mechanisms defining how to extend the ontology.

6.9 Categorisation of Ontologies for Software Engineering

To clarify the usage of ontologies within the software engineering and software technology, a taxonomy of ontologies has been proposed by Ruiz in [RH06]. Two different categories are considered and further detailed:

1. *Ontologies of Domain*: to represent (partial) knowledge of a particular domain within Software Engineering and Technology.
    - Software Engineering (SE)

    - Generic
    - Specific: requirements, design, construction, testing, maintenance, configuration management, quality, engineering tools and methods, engineering process, and engineering management.

  - Software Technology (ST)
    - Software: programming techniques, programming languages, and operating systems.
    - Data: data structure, data storage representations, data encryption, coding and information theory, and files.
    - Information Technology and Systems: models and principles, database management, information storage and retrieval, information technology and systems applications, information interfaces, and representation.

2. *Ontologies as Software Artifacts*: ontologies are used as some kind of artifact during a software process.
   - At Development Time
     - For Engineering Processes: when ontologies are used during development and maintenance processes.
     - For Other Processes: when ontologies are used during complementary customer–supplier, support, management, and organization processes.
   - At Run Time
     - As Architectural Artifacts: when ontologies are part of the system software architecture.
     - As (Information) Resources: when ontologies are used as resource at run time by the system.

An additional categorisation of ontologies for software engineering has been provided by Happel in [HS06]. Two dimensions for comparison are considered: the role of ontologies (at run time vs. at development time) and the kind of knowledge (domain vs. infrastructure). Combining these two dimensions, four different areas are defined:

- Ontology–driven development (ODD): ontologies are used at development time to describe the problem domain. Integration efforts with software modelling languages within Model–Driven Development fall under this category.

- Ontology–enabled development (OED): ontologies are used at development time to support developers with their activities. Examples are component search or problem–solving support.

- Ontology–based architectures (OBA): ontologies are used at run time as part of the system architecture. Business rules approaches are an example.

- Ontology–enabled architectures (OEA): ontologies are used at run time to provide support to the users. An example are Semantic Web services.

Comparing Ruiz and Happel categorisations, it should be pointed out that the latter focus only on the category Ontologies as Software Artifacts, previously described in Ruiz's taxonomy. Both consider the same temporal dimension (at run time v.s at development time), as described in [Gua98]. Additionally, a closer analysis of the definitions in Happel's categories show that the author considers ontologies to play a similar role. Firstly, as main artifacts for engineering (ODD) or architectural purposes (OBA). Secondly, as support artifacts for development (OED) or information (OEA).

6.10 Current Research on Ontologies for Software Engineering

This section provides a literature review of current research on the use of ontologies within Software Engineering. The attempt is to cover most of the current research under different topics.

To achieve some level of organization within the several extant ontologies, environments and projects, a criterion should be considered. An analysis of the roles (Sec. 6.7) and categorisations (Sec. 6.9), leads to two conclusions. Firstly Ruiz's taxonomy covers a wider range of categories than Happel's, including those ontologies used to describe Software Engineering concepts, which is useful for the focus of our research. Secondly, the two categories in Ruis's taxonomy could be mapped to the two main roles defined beforehand (see Table 6.2).

Roles	*TaxonomyCategories*
Ontologies for Software Engineering Domain	Ontologies of Domain
Ontologies for Software Engineering Process	Ontologies as Software Artifacts

Table 6.2: Mapping of Roles and Taxonomy Categories

Therefore, in the following subsections, the ontologies have been classified according to the mapping of roles and the taxonomy described in Table 6.2. To point out that the category Software Technology described in Ruiz's taxonomy as part of Ontologies of Domain has not been included in our literature review. It is mainly relevant to ontologies describing techniques and languages in technology and systems, thus falling apart from the research focus of this dissertation, on how ontologies could improve the Software Engineering process.

6.10.1 Ontologies for the Software Engineering Domain: Ontologies of Domain

Ontologies are used, in general, to represent (partial) knowledge of a particular domain within Software Engineering. Table 6.3 summarises the examples found in the literature, which are later described.

Category	Subcategory	Example
Generic		Ontologies based on the SWEBOK guide [MA05b], [MA05a], [SCR05]
Specific	Requirements	Generic Requirement Analysis Method based on Ontologies (GRAMO) [GdFB04] and ONTODM [GF03]
	Design	Ontology–based Domain–driven Design [Hru05] Ontologies in Software Design [Kal00]
	Software Maintenance	Software Maintenance Ontologies [DAdO03], [AdOD06], [RVPG04], [VAO+05]
	Software Quality	Software Quality Ontology [FGD02] Software Measurement Ontology [BVG06] Ontology for Software Metrics and Indicators [MO04]
	Software Engineering Process Description	Ontology–based Development Environment (ODE) [FGD02], [FNM+03], [FRPM04], [Fal04], [FRM05] Onto–ActRE framework [LG05] Ontology based Requirements Elicitation (ORE) [Sae04] Software Process Ontology (SPO) [LQL05] Ontology for Software Development Methodologies and Endeavours [GPHS06]

Table 6.3: Examples of Ontologies of Domain

1. *Generic*

   Two different examples, reviewed in [ACGB+06], can be found as **Ontologies based on the SWEBOK guide**. The Guide to the Software Engineering Body of Knowledge (SWEBOK) [Soc04] seeks to identify, describe and organise knowledge and practices applicable to most projects most of the time.

   The first example considers three phases [MA05b], [MA05a]: proto–ontology construction, internal validation cycle and finally, external validation and possible extension after analysisng several ontology development methodologies The SWEBOK proto–ontology contains over 6,000 concepts and 1,200 facts or instances of concepts.

   Another example is the Onto–SWEBOK project [SCR05], where essential software engineering concepts are mapped into OpenCyc. Such concepts encompasses the idea that a software engineering deals with *artifacts* created by *agents* as a result of *activities* guided by *rules*. The emphasis is on how to represent activities and artifacts.

2. *Specific*

- Software Requirements Ontologies provide an unambiguous terminology to be shared by all stakeholders involved in the development process, therefore aiding in the conceptualisation of software requirements. Fruthermore, ontologies can be as generic as needed allowing its reuse and easy extension.

  Generic Requirement Analysis Method based on Ontologies (GRAMO) [GdFB04] is an ontology–based technique for the specification of domain models in the analysis phase of Multi–Agent Domain Engineering. The underlying ontology ONTODM [GF03] is a frame–based generic ontology for the construction of domain models, that represents the knowledge of techniques for the specification of the requirements of a family of multi-agent systems in an application domain.

- Software Design

  A first example is the **Ontology–Based Domain–Driven Design** [Hru05]. The research focuses on the development of software applications for a specific domain, based on the use of ontologies. Hruby distinguishes between *domain ontology* (which specifies the structure of concepts to be applied to all systems in the domain) and *application functionality* (which specifies the functionalities and user requirements which might differ from application to application).

  Additionally, ontologies have been used as domain knowledge **to check the initial phases of software systems design**, with the aim of detecting conceptual errors within the domain knowledge [Kal00].

- Software Maintenance

  Software maintenance is a knowledge–intensive process, where knowledge of activities, legacy software, system architecture, problem requirements coming from different sources is handled to maintain the software. Therefore, ontologies for software provide a solid basis to conceptualise such knowledge.

  A first example is a **Software Maintenance Ontology** developed by [DAdO03], where the knowledge required to maintain a software system is organized as five different sub–ontologies: system, computer science skills, maintenance process, organizational structure and application domain. A recent version is described in [AdOD06]. An additional example is that of a **Maintenance Ontology** developed by [RVPG04], as part of a software environment, MANTIS [RGPP02], developed to manage maintenance projects. The ontology consists of four subontologies: products, activities, process organisation, and agents. Due to some similarities among the previous ontologies, an attempt to merge them is described in [VAO^{+}05].

- Software Quality

The importance of quality in software is a topic under study within software engineering. However, what it is understood by software quality is not fully defined. To tackle this variability, a **Software Quality Ontology** [FGD02] was developed as part of Ontology–based Domain Engineering (ODE). Different competency questions regarding quality characteristics, relevance, metrics and paradigms were considered, using LINGO as modelling language.

Likewise, within software quality, several standards and proposals coexist for software measurement terminology. To address this issue, a **Software Measurement Ontology (SMO)** [BVG06] was developed using different standards as input. Concepts, definitions, and relationsips were identified using REFSENO [TvW98] as methodology. The ontology was organised in four sub–ontologies: software measurement characterization and objectives, software measures, measurement approaches and finally, measurement. A similar effort is the **Ontology for Software Metrics and Indicators** [MO04] developed using METHONTOLOGY [FLGPJ97] as the development methodology. UML as the modelling language, allowed to define concepts related to quality, performance, and measurement.

- Software Engineering Process Description

  Ontologies have been used as domain ontologies to describe software engineering processes. A first set of applications can be grouped as those focused on developing an ontology for the engineering process description of the main concepts involved in software processes, such as process, project, activity, artifact, resource, procedure, and so on. A second group would include those projects developing an ontology to reflect standards and methods commonly used in SE.

  Within the first group, a comprehensive effort was developed by Falbo et al. [FGD02], [FNM+03], [FRPM04], [Fal04], [FRM05]. An **Ontology–based Development Environment (ODE)** considers three main activities: domain analysis in the form of ontology development with a systematic approach, infrastructure specification made by mapping the ontology into object models, and finally, infrastructure implementation by developing Java components. The ODE's architecture consists of different levels: ontological (to describe the ontologies themselves); meta level (classes related to the knowledge of the software engineering domain), and base level (classes that implement ODE's applications).

  A second example is the **Ontology–based Active Requirements Engineering Framework (Onto–ActRE framework)**[LG05] developed to elicit, represent, and analyse the myriad of elements and factors involved in the development of software–intensive systems. The framework includes different models: goal–driven scenario composition (to model goals and objectives at different level of abstraction); requirements domain model (to model requirements from top–level to sub–domains); viewpoint hierarchy (to model viewpoints from different stakeholders, system's and environment's concerns);

other domain taxonomies (to classify domain concepts, properties and relationships). As a result, the Problem Domain Ontology (PDO) is obtained

Finally, **Ontology–based Requirements Elicitation (ORE)** [Sae04], [KS06] proposes a requirement elicitation method based on a domain ontology. The artifacts used to elicit requirements are a requirement list provided by users in natural language, a domain ontology and finally, both a mapping and inference rules between the requirements list and the ontology concepts and relationships.

Within the second group, ontologies to find commonalities among standards and methods previously defined by the SE community, a **Software Process Ontology (SPO)** [LQL05] analyses two different software process models such as Capability Maturity Model (CMM) and ISO/IEC 15504 to define key concepts in software processes (which are later mapped from one to another). The analysis allows for the definition of an ontology–based software process model framework which encompasses a Software Process Ontology (SPO), which defines the process model at a conceptual level. The SPO is used to build two different ontologies to fulfil for the two analyzed models. The use of the SPO and its extensions consists in allowing an user to check the description of the processes and practices recommended by the reference model.

Another example is the **Ontology for Software Development Methodologies and Endeavours** [GPHS06] with the focus on considering concepts and methodologies to be used both by method engineers and software developers when developing software. The ontology considers both a metamodel and a three–layer architecture based on domains (metamodel, method and endeavor) that can be used to specify methodologies and then, apply them to real endeavours.

6.10.2 Ontologies in the Software Engineering Process: Ontologies as Software Artifacts

Within this category, ontologies are used in the software process as some kind of *artifact*, either at run time or at development time. At development time, an ontology can be used as an artifact for engineering or other processes. An ontology used at run time could be either part of the system software architecture or an information resource. This section reviews some of the existing research on the use of ontologies as sofware artifacts, both at development and at run–time as shown in Table 6.4.

1. At Development Time

   - *For Engineering Processes*

   To support software developers in domains not familiar to them, a **Domain Oriented Software Development Environment (DOSDE)** [dOZR^{+}04]

Category	Subcategory	Example
At Development Time	For Engineering Processes	Development process: DOSDE [dOZR+04] and EOSDE [OVRT06] Maintenance process: MANTIS [RGPP02]
	For Other Processes	Ontology–based Software Engineering for Multi–site Software Development [WCD05], [WCC05]
At Run Time	As Architectural Artifacts	Ontology Driven Architecture (ODA) [For01], [Knu04] Software Components in an Application Server [OESV04]
	As Information Resources	Ontology–based Infrastructure for Intelligent Systems [Ebe03] Ontology Driven Web Information System Environment (WISE) [TLQ+06]

Table 6.4: Examples of Ontologies as Software Artifacts

was developed considering two types of knowledge: related to the application domain and to the tasks. The first kind of knowledge was implemented as a *domain ontology*, divided into sub–ontologies to model the main concepts of the domain. The second kind as a *task ontology* combined with a *Problem Solving Method (PSM)* into a single model (PST) that considered three different levels to describe both the tasks and the way to carry them out. Additionally, the sub–ontologies of the domain ontology were mapped to the tasks of the PST.

An extension of DOSDE, to support also organizational knowledge has been implemented as an **Enterprise–Oriented Software Development Environment (EOSDE)** [OVRT06]. Part of the environment is the Enterprise Ontology, fundamental to support common vocabulary that represents useful knowledge for software developers within an organization. It consists of different subontologies (intellectual capital, behavior, artifacts, structure and general strategy) to address different aspects of the enterprise.

For engineering processes, with a focus on maintenance, a **Software Maintenance Environment MANTIS** [RGPP02] was developed to handle maintenance projects. Maintenance Ontologies [RVPG04] were used as common conceptual framework in MANTIS. The ontologies allow to share knowledge among those participating in a software maintenance project. The ontology has also been used in a knowledge management system.

- *For Other Processes*

The development of multi–site software projects involves the development teams to reside in different geographically dispersed sites. Therefore, this kind of projects are prone to misunderstandings within teams. Ontologies serve as a common conceptualisation, thus easing communication.

Ontology–based Software Engineering for Multi–Site Software Development [WCD05], [WCC05] has focused on this kind of problems by using ontologies as both common conceptualization and communication mechanism among teams. The proposed solution considers two different ontologies: Generic ontology (to define terms, vocabulary, semantic interconnections, and rules of inference) and Application–specific ontology (to specify object–oriented development for a particular project). Benefits of this approach are the use of an explicit, formalized, machine–readable and consistent concepts among multi–site developers.

2. At Run Time

- *As Architectural Artifacts (ontology–driven software)*

  The W3C's Software Engineering Task Force has proposed the **Ontology Driven Architecture (ODA)** [For01], where an ontology describes the properties, relationships and behaviors of the components required in a software development process.

  The *tourism domain within the Semantic Web* is a particular case of the above described in [Knu04]. Ontologies are used as domain models not only for code generation, but they as executable artifacts at run–time.

  Similarly, an ontology–based approach has been used to aid in the development of *software components in an application server* [OESV04]. The ontology is used as a conceptual model both for development and run–time of software components. The ontology is divided into generic modules, to model both semantic and syntactic metadata. Additionally, domain modules have been implemented to formalize knowledge specific to a certain application.

- *As Information Resources (ontology–aware software)*

  Ontologies have been used for data integration belonging to several sources in the **Ontology–based Infrastructure for Intelligent Systems** [Ebe03], tested as part of a help system for online learning. Another example of ontologies as information resources can be found in the **Ontology Driven Web Information System Environment (WISE)** [TLQ$^+$06], where ontologies are used as domain models to develop a Web Information System. Two ontologies are considered: the domain ontology and the behaviour ontology. The first one models static entities such as system structures and stored data

[TLP+05a]. The second one describes dynamic processes such as system activities and interactions [TLP+05b].

Part III

Research and Application

Chapter 7

OASYS ONTOLOGICAL ENGINEERING

7.1 Introduction

Ontological engineering refers to the different activities in the development process, the methodologies to support it, and the languages and tools used for the deployment of an ontology. The ontological engineering processes related to the Ontology for Autonomous Systems (OASys), focus of this research, have been described in several chapters in this dissertation. Chapters in Part III are dedicated to decribe OASys' design features, structure, contents, deployment, and finally, its usage.

This chapter introduces general requirements to be addressed during the ontology development, specifying the requirements and guidelines on how the ontology should be. Based on them, the ontology features are finally implemented, which are described in following sections. Finally, a detailed account and description of OASys structure and contents is given, with a thorough description of the key concepts, relationships, and other ontological elements considered, emphasising the underlying ideas for their inclusion in the current state of the ontology.

7.2 OASys Requirements

For ontological engineering, a key aspect is to state the ontology's purpose, which will drive its development and further contents. Knowing what the ontology is to be developed for, allows to focus on the essential ontological elements to be included, leaving aside out of scope ones. Additionally, it is necessary to define the type of ontology based on the subject of conceptualisation to consider. The level of abstraction, generality, and reusability of the ontological terms to be gathered in the ontology changes when considering an upper–level ontology from a domain one. Different design criteria serve as guidelines to support this development. The methodology and the language chosen for the ontology deployment are also specified and justified in this section.

Purpose: OASys will define the concepts and constructs to be considered in the definition of the structure, function and behavior of autonomous systems. The ontology will also capture and exploit the concepts necessary to support the engineering process of any autonomous system. The ontology will provide building elements to construct any autonomous system, by characterising goals, system and control system features.

Type of ontology: OASys will be a domain ontology to describe the autonomous system domain. A domain ontology provides the concepts and their relationships within a domain, about the activities in that domain and about the theories and principles guiding that domain [GPFLC04b]. Being a domain ontology allows a high level of usability as it captures the domain knowledge in a problem–solving independent manner, being its reusability constraint to autonomous systems related aspects.

General Structure: OASys will need to address two different aspects in its structure. On the one hand, the necessity to cover both general and autonomous systems knowledge. On the other hand, its purpose of providing ontological elements both for the autonomous system's analysis and the engineering process.

Design criteria: To assure coherence and quality, OASys will be developed bearing in mind the design criteria summarised in Sec. 4.4 and extended in [GPFLC04b], which are objective guidelines for the ontology development. Among the different possible criteria, OASys will pay special attention to clarity, extendibility, minimal encoding bias, minimal ontological commitment and standardisation.

Knowledge acquisition: OASys will cover the wide domain of autonomous systems. Therefore, the ontology constructs should consider elements from a myriad of theories and domains to cater for different types of autonomous systems. At a more abstract level, theories related to general systems, mereology, topology for their description and definition will be analysed. Different systems engineering theories will be considered as a metamodel for a later refinement for autonomous systems engineering.

Regarding the domain itself of autonomous systems, a wide range of topics will be studied such as: Intelligent Autonomous Systems, Robotics, Agent Technology, Control Engineering, Continuous Processes, and Cognitive Science. Special attention will be paid to the underlying ideas and theories developed as part of the long–term ASys project [SR08], as backbone of the ontology. The research will not only provide the concepts for the autonomous systems characterisation, but also will propose a framework to develop such systems.

Methodology: The purpose of this research is not to develop a brand new methodology for ontology development, but to use a well-established and documented one among the several methodologies available in the literature (see Section. 5.2.2). The chosen methodology should be suitable enough for the development of OASys, and might need to be adjusted as needed.

Formalisation: OASys will be formalised using software engineering general- and specific-purpose languages, such as UML [Obj05] and SysML [Gro08a]. The reasons to choose these languages are twofold: software engineering techniques to develop ontologies are mostly UML–centered, as reviewed in Sec. 5.3 and Ch. 6; secondly, the developed ontology will support the model–driven engineering process in the ASys project [SHH+09], [SHG+09].

UML is not an ontology development language, hence it has been considered to have some drawbacks for such task (lightweight ontology, incomplete semantics, software heritage, lack of inference mechanisms, etc). However, using UML brings some benefits such as its widespread use among engineering practitioners, its support of a graphical representation, and its independence of a specific programming language. Moreover, these shortcomings have been addressed by the Ontology Definition Metamodel (ODM) (see Sec. 5.3.1.2) which defines metamodels, mapping and profiles to allow the interaction between UML and traditional ontological languages such as OWL and RDF.

Reusability: A main aspect of the ontology is that of its reusability, i.e., the possibility of being further used outside the ASys project. The ontology is born with the objective of providing the grounds for any autonomous system development, where such autonomous systems cover a wide range of different subdomains and applications. Even more, OASys will provide a conceptualisation of larger reach, entering the extended domain of autonomy that includes both artificial systemas and natural ones (e.g. animals).

7.3 OASys Features

This section describes how the aforementioned requirements have been finally addressed in the ontological engineering process of OASys.

7.3.1 Structure

OASys has been organised according to two orthogonal dimensions: one consisting of different levels of abstraction to separate general knowledge from particular one; the other focusing on the separation between the autonomous system analysis and its engineering process.

To address the different levels of abstraction, OASys has adopted a layered structure considering the underlying ideas of progressive domain focalisation as described in [SAS$^+$99], where a complex problem such as the design of intelligent autonomous controllers is tackled by decomposing the analysis in stages, from domain to application. This view agrees with the necessity of considering different levels of abstraction, from general systems to specific applications. Therefore, the ontology has considered the domain of a general system, as well as the more specific of autonomous systems, where concepts are further specialised for particular autonomous systems.

The more abstract level is the System Domain Layer. This layer contains the concepts related to the General Systems Theory, and additional Mereotopological concepts to express mereological and topological relationships in any system. For this layer, existing ontologies already defined have been analysed, and considered for its integration within OASys. A second layer, the ASys Domain Layer centralises the ontological engineering process of this research, gathering the concepts and related elements to describe and charaterise an autonomous' system structure, function and behaviour. The focus lies on the autonomous system domain as conceptualised in the framework of the ASys research project, however being general enough to be re-used in the development of any autonomous system. In general, the upper layer defines the concepts that might be reified in the lower one. The lower layer specialises and refines the concepts to be used for a real application within the Autonomous Systems domain.

To tackle the separation between the autonomous systems' characterisation and engineering, OASys has been structured in two main ontologies, shown in Fig. 7.1. The **ASys Ontology** gathers the concepts, relations, attributes and axioms to characterise an autonomous system. The **ASys Engineering Ontology** collects the ontological elements to describe and support the construction process of an autonomous system. Each one of the two ontologies expands through different layers to cater for concepts from higher level of abstraction to lower ones: from general concepts to characterise a system (System Domain Layer), to autonomous systems concepts (ASys Domain Layer) describing any autonomous system.

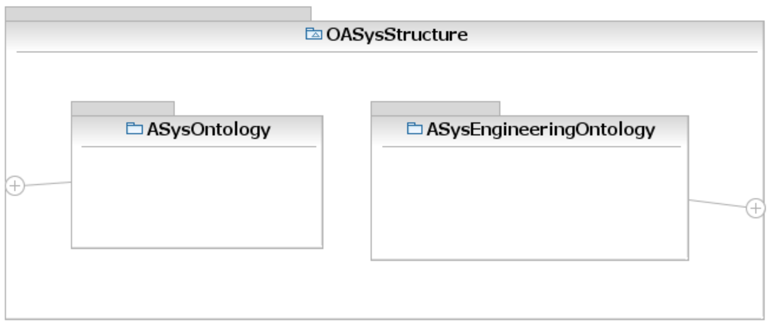


Figure 7.1: OASys Structure: Ontologies

7.3.2 Design Criteria

The design criteria have been followed throughout the development of OASys. The following sections explained how the original requirement has been tackled and finally accomplished during the definition and development of the ontology.

7.3.2.1 Clarity

For the requirement of clarity, well–known ontologies and glossaries have been reviewed to include their terms as required for OASys. The aim was to benefit from concepts which have existed and long used in the scientific community. This process was specially important for the System Domain Layer concepts belonging to well–established theories of general systems, mereology, topology and systems' engineering.

The concepts of the ASys Domain Layer mainly come from the ASys team research efforts. The different available documents have been carefully analysed to extract the ontological elements, checking for mismatches or commonalities. Those concepts have been later discussed with the group members to commit to the desired meaning for our research.

Finally, all the ontological constructs (concepts, relationships, attributes, axioms) have been defined in natural language. These definitions have been gathered in the Annex.

7.3.2.2 Extendibility: subontologies and packages

To cater for OASys extensions in the future, concepts within one layer have been classified using two different elements. The first element is that of *subontologies*, which will allow to modify concrete aspects in the ontology without the necessity of changing it all. Different subontologies are designed to classify concepts closely related to a domain viewpoint.

The second one is the use of *packages*, if needed, to further classify the concepts within a particular subontology according to a concrete aspect. The possibility of packaging related elements foresee the increase in the number of ontological elements as the ontology evolves in time.

The ontology's scalability profits from these two-leveled organising elements, allowing thus the extension or modification of the ontology without major changes to its structure and composition. Within a layer, new subontologies can be added to extend the existing ones with the purpose of addressing a new viewpoint when the ontology is to be extended. Within a subontology, new packages can be added from scratch, i.e., with brand new concepts focusing on a particular aspect to address a deeper level of organisation. Existing packages can be modified or updated by adding or removing concepts.

Hence, different subontologies have been considered in OASys, to contain domain–related concepts. Next, packages have been defined within each subontology to gather aspect–related concepts. As mentioned, the subontologies and packages will ease the process of modification, extension or update of the ontology as new subdomains or applications are considered.

- **ASys Ontology**

  For the ASys Ontology, two subontologies have been developed to address the different level of abstraction for an autonomous system description (Fig. 7.2). The *System Subontology* contains the ontological elements necessary to define any system structure, behaviour and function based on general and well–established theories. The *ASys Subontology* specialises and refines the previous concepts, adding autonomous systems specific ones.

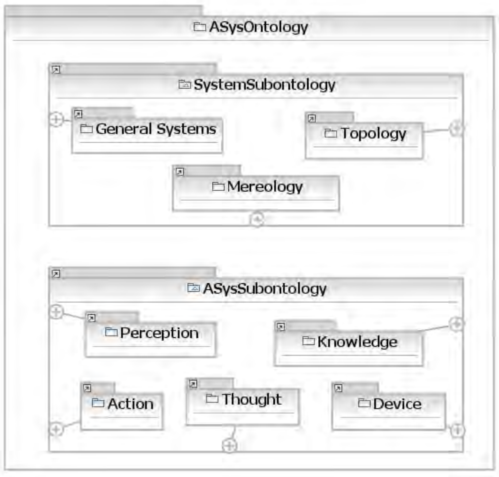


Figure 7.2: ASys Ontology: focus on the system as it is

At the upper level of abstraction, the *System Subontology* contains different packages gathering the concepts for any system's definition (Table 7.1): General Systems, Mereology, and Topology.

At a lower level of abstraction, the *ASys Subontology* consists of several packages to address an autonomous system description. The ontological elements within the different packages included in this subontology, refine and specialise the concepts and relationships defined in the System Subontology at the System Domain Layer. The purpose and content are detailed in Table 7.2.

Package	**Purpose**	**Related Theory**
General Systems	To gather concepts to characterise any kind of system's structure, function and behaviour	General Systems Theory
Mereology	To provide general concepts for whole–part relationships	Mereological and Meronymic Theories
Topology	To collect general concepts for topological connections	Topological Theories

Table 7.1: System Subontology Packages

Package	**Purpose**	**Related Theory**
Perception	To gather concepts to describe the perceptive and sensing process in an autonomous system	Perception theories, ASys project research on perception
Knowledge	To compile concepts to describe the different kinds of knowledge used by an autonomous system	Cognitive Science, ASys project research on model–based knowledge
Thought	To characterise concepts to describe the thinking (decision-making and goal-orientation) process in an autonomous system	Cognition, goal–oriented general theories and ASys project research
Action	To collect concepts about the operations carried out and performing actors in an autonomous system	Multi–agent theories, Control theory, ASys project research on control, actions and cognitive processes
Device	To provide concepts to describe the different devices in an autonomous system	ASys project research on testbeds

Table 7.2: ASys Subontology Packages

- **ASys Engineering Ontology**

For the ASys Engineering Ontology, two different subontologies have been considered as shown in Fig. 7.3. The *System Engineering Subontology* gathers the concepts related to any system engineering process as general as possible based on system's engineering and software engineering methodologies. The *ASys Engineering Subontology* contains the specialisation and additional required elements to describe an autonomous system's engineering process.

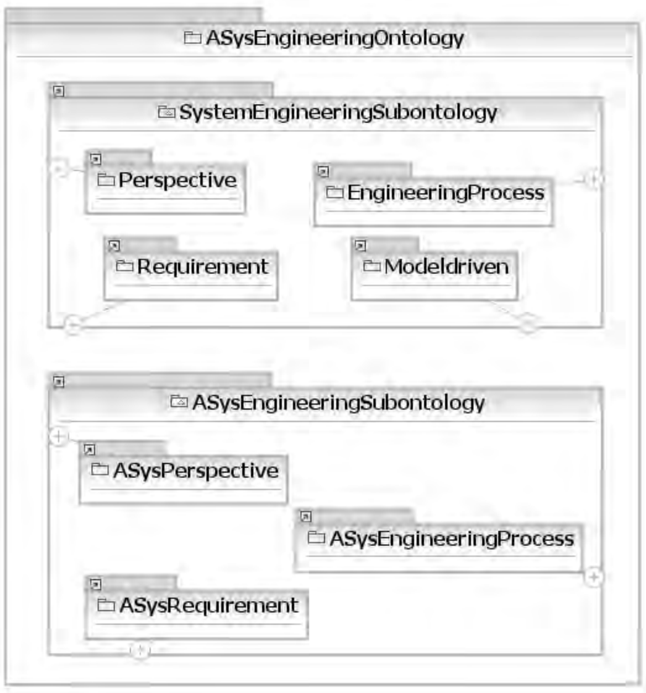


Figure 7.3: ASys Engineering Ontology: focus on the construction process

The *System Engineering Subontology* provides the concepts for any system's engineering process based on already developed theories for software engineering, organised into packages to address different aspects of system's engineering. The purpose and related theory for each package is described in Table 7.3.

The *ASys Engineering Subontology* provides the concepts for an autonomous system engineering process, with different packages considered to gather aspect related elements, as specialisation of the upper level concepts. To mention that this subontology does not yet include an ASys Model–driven package (as a specialisation of the upper layer one), since the specific model–driven development for autonomous systems in the ASys research project will be later specified in forthcoming stages.

Package	Purpose	Related Theory
Requirement	To provide concepts to describe system's requirements	Requirements theories
System Perspective	To gather concepts to describe the different viewpoints to consider in a system	System's engineering theories
Engineering Process	To summarise concepts to describe the engineering process	Engineering metamodels and theories
Model–driven	To collect concepts to describe model–driven engineering	Model–driven theories

Table 7.3: System Engineering Subontology Packages

The purpose and content of the packages is specified in Table 7.4.

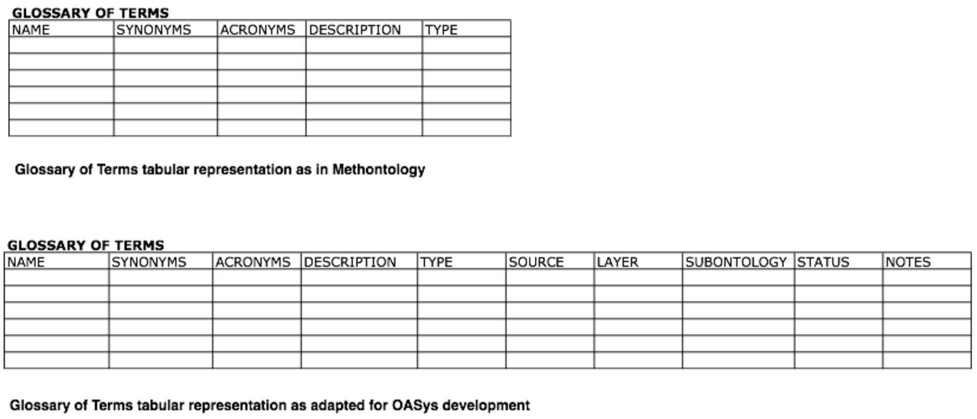
Package	Purpose	Related Theory
ASys Requirement	To cater for concepts to describe common requirements in an autonomous system	Requirement theories applied to ASys research
ASys Perspective	To gather concepts to describe an autonomous system from different aspects	ASys research theories
ASys Engineering Process	To collect concepts to describe the construction process of an autonomous system	Engineering theories applied to ASys research

Table 7.4: ASys Engineering Subontology Packages

7.3.2.3 Minimal encoding bias: intermediate representations

To assure as much as possible the minimal encoding bias, i.e. the incorrect conceptualisation of concepts based on the final implementation language syntax or appearance, intermediate tabular representations and graphs have been used to define the different ontological constructs.

As guideline, the suggested tables in METHONTOLOGY have been used initially in the process for the different conceptualisation tasks (glossary of terms, binary relations, attributes, constants, axioms, rules), implemented as Excel workbooks. Although not the best possible tool, its easiness of use and availability for every team member, made it feasible as starting tool to conceptualize the ontology. The methodology allows for enough flexibility to adapt or modify this tabular representations for the specific ontology under development. Hence, additional fields such as source (to show document, external ontology or expert as input), layer and subontology to specify where the ontological element was defined, status (both personal and research group draft or approval stage), and notes have been added to detail different tables as needed (Fig. 7.4).



GLOSSARY OF TERMS

NAME	SYNONYMS	ACRONYMS	DESCRIPTION	TYPE

Glossary of Terms tabular representation as in Methontology

GLOSSARY OF TERMS

NAME	SYNONYMS	ACRONYMS	DESCRIPTION	TYPE	SOURCE	LAYER	SUBONTOLOGY	STATUS	NOTES

Glossary of Terms tabular representation as adapted for OASys development

Figure 7.4: OASys Minimal Encoding Bias: Intermediate Tabular Representation

7.3.2.4 Minimal ontological commitment: fundamental concepts to be specialised

As required, on each layer only the fundamental concepts have been described. Lower layers elaborate on these concepts by adding new ones to provide a deeper level of detail. Furthermore, only those relevant concepts to the domain of autonomous systems will be initially considered. New concepts will be added as new applications are built while using OASys.

7.3.2.5 Standardization: adopted naming convention

Choosing a naming convention for ontology modeling and adhering to it makes the ontology easier to understand and helps to avoid some common modeling mistakes. Therefore, for the modeling of ontologies, the following set of conventions for ontology elements (i.e. classes, properties, relationships, axioms, rules, and instances) have been adopted:

1. Capital Letters
    - Concept's names start with a capital letter. Whenever the name contains more than one word, such name is made of the concatenated words being capitalised each one of them, e.g., SystemModel.
    - Attribute's names use lower case. If the name contains more than one word, concatenated words are used where the first word is all in lower case, and each one of the rest is capitalised, e.g., goalStatus.

- Relation's names follow the same convention as for attributes.
- Axioms and rules are expressed in natural language, referring to defined concepts

2. Use of singular

   For the name of the elements in the ontology (concepts, attributes, relations, etc), the singular form is preferred, e.g, System as opposed to Systems.

3. Additional conventions

   - Ontology's, subontology's and packages' names will capitalise each of the terms, e.g, System Subontology or Topology.
   - Ontologies and subontologies will add the word Ontology and Subontology to their name respectively, e.g. System Subontology. Packages will be named in a similar way, e.g., Knowledge Package.
   - Words such as concept, attribute and so forth should not be used as part of an element's name.
   - Although acronyms are given within the ontology, the elements' names should not contain them to avoid confusion.

7.3.3 Inputs and sources

Knowledge acquisition for ontology development can be made by considering different sources among others: documents, existing ontologies, and experts. Documents such as articles, technical reports, or books serve as an input source for the ontological elements to be considered in an ontology. Existing ontologies should also be reviewed, as the domain might have already been conceptualised, however with a different viewpoint or purpose. These existing ontologies should be selected, evaluated, and finally fully or partially reused, paying attention to the level of granularity (if the existing ontology covers the same level of detail as in the ontology under development). Domain experts also act as a possible source for the conceptualisation, since they provide their terminology, i.e., the words and terms they are familiar with when talking about the domain.

For the purpose of the development of the ontology, mainly documents and existing ontologies have been considered. Documents have been analysed to come up with existing terminology and definitions for the different domains, subdomains, applications and aspects considered in the ontology's structure. The sources included articles, technical reports describing body of knowledge, and books. As underlying focus, the research and ideas developed in the ASys research programme. Existing well-established glossaries and ontologies have been studied to establish their integration in the ontology. Available experts in some of the fields addressed by the ontology have been questioned to acquire some concepts, however not being the main source.

Table 7.5 gathers the different sources used in the ASys Ontology subontologies and packages. In a similar way, Table 7.6 summarises the inputs considered for the ASys Engineering Ontology specification, including the subontologies and packages.

SUBONTOLOGY	PACKAGE	SOURCES
System Subontology	General Systems	[Kli69], [Kli85], [Kli91], [KE03],
	Mereology	[Bor97], [Gui05], [KA07], [MWM07], [MBW$^+$08]
	Topology	[MWM07], [MBW$^+$08]
ASys Subontology	Device	[RJB04], [HH08], [dlM09b]
	Perception	[UIT06e], [UIT06a], [Lóp07]
	Knowledge	[SR08], [Lóp07], [HSL08], [SHR07], [AT06b]
	Thought	[UIT06d], [UIT06b]
	Action	[Lóp07], [HSL08]

Table 7.5: ASys Ontology Sources

SUBONTOLOGY	PACKAGE	SOURCES
System Engineering Subontology	Requirement	[IEE90], [Gro08a], [Obj09b], [SR08]
	Perspective	[Ins00]
	Engineering Process	[Gro08b]
	Model–driven	[Obj03], [SV06]
ASys Engineering Subontology	ASys Requirement	[SLB$^+$05], [SLB07], [HSL08]
	ASys Perspective	[RJB04], [MWM07], [vL08], [FMS08]
	ASys Engineering Process	[SMP99], [Dou04], [Seg05], [SR08], [Est08]

Table 7.6: ASys Engineering Ontology Sources

7.3.4 Methodology: METHONTOLOGY–based Development Process

From the available methodologies and methods for ontology engineering, METHONTOLOGY [FLGPJ97] has been chosen as a starting point since:

1. The stages for the development process are well and clearly defined in an ontology life cycle.
2. The methodology comprises different and further tasks to be considered, such as the ontology maintenance.
3. The conceptualisation activity is decomposed into different detailed tasks, with a proposed order.
4. Intermediate representations, such as tables and graphs, can be easily understood both by domain experts and ontology developers.
5. It allows for flexibility from different viewpoints: the process (tasks can be revisited if new concepts are added), the representation (tables can be modified as needed) and evolving prototypes (a new prototype is obtained when adding, changing or removing terms).

METHONTOLOGY proposes both an ontology development process, as well as an ontology life cycle closely intertwined. The development process refers to which activities are performed when building the ontology. The ontology life cycle identifies when these activities should be carried out, by a set of stages that define which activities to perform in each stage and how they are related. Both the development process and the life cycle activities have been initially followed for the development of OASys. Some additional guidelines described in [NM01], [Miz04], and [BA07] have been also considered. Some activities proposed in the methodology had to be adjusted (adding more detailed activities, extending, etc) to cater for the development of OASys:

- Management activities: they consist in scheduling, control and quality assurance activities. The scheduling activity identifies the tasks to be performed, their arrangement, and the time and resources needed. The control activity guarantees completion of tasks as intended. The quality assurance activity checks the quality of each methodology output (ontology, software and documentation).

  These activities have been performed by establishing in an overall way the schedule, the time and the effort needed to develop OASys during the research, as part of a personal planning. Control and quality have been performed ad–hoc both by the ontology developer as well as the supporting team.

- Development activities: these activities are grouped within METHONTOLOGY into pre–development, development, and post–development activities.

  The pre–development activities usually consider a feasibility study as well as the environment where the ontology will be used. For the purpose of the research, the outcome of these activities has been the state of the art and research objectives.

The development activities encompass the specification, conceptualization, formalization, implementation, and maintenance of the ontology. As a result, a prototype is obtained which evolves as new versions are considered. Each activity has been carried out during OASys development. To point out that, due to the layered and subontology–based structure of OASys, the development activities have been applied to each layer, and within it, each subontology and package has been developed following the overall process:

1. Specification: in general terms, the aim is to find an answer to questions such as which domain is considered, which use is given to the ontology, and who will use the ontology. Formally, it can be done using competency questions or scenarios where the ontology will be used.

2. Conceptualisation: the objective is to organise and to structure the knowledge acquired using external representations that are independent of the knowledge representation and implementation paradigms in which the ontology will be formalised and implemented afterwards. An informally perceived view of a domain is converted into a semi–formal model using intermediate representations based on tabular and graph notations. A succession of tasks is followed, such as building glossary, taxonomies, relations, detailing previous elements, and axioms.

   For each one of the ontologies, subontologies and packages developed, the conceptualisation tasks have been followed and revisited as needed. The outcome have been different tables and graphs containing the ontological elements considered, expressed in natural language. Examples of them will be provided in the corresponding section explaining the subontology and packages in detail.

3. Formalisation: the goal of this activity is to formalise the conceptual model. There are ontology development tools that automatically implement the conceptual model into several ontology languages using translators (see Sec. 5.2.4). To satisfy the software engineering focus, OASys has been described using a software engineering general- and specific-purpose language such as UML [Obj05].

4. Implementation: this activity builds computable models using ontology implementation languages. There are many ontology languages and they do not have the same expressiveness nor do they reason the same way.

During post-development, the maintenance activity updates and corrects the ontology if needed and it can be reused by other ontologies or applications. Further use of OASys as part of the long–term ASys project will deal with the maintenance and re–use of OASys.

- Ontology support activities: they are performed at the same time as the development-oriented activities. Usually, such activities consider knowledge acquisition, integration, evaluation, and documentation. For OASys development, mainly knowledge acquisition, evaluation and documentation have been carried out.

7.4 OASys Description

This sections describes the ontologies, subontologies and packages considered in OASys, paying special attention to the description of the ontological engineering process of each one of the different packages. Each package has been developed considering the proposed development activities of METHONTOLOGY, from the package's purpose to its final UML formalisation.

7.4.1 ASys Ontology

The ASys Ontology gathers, at different levels of abstraction, the ontological constructs necessary to describe and characterise an autonomous system. The ontology comprises two subontologies (Fig. 7.5): the System Subontology to describe the higher level of abstraction concepts for systems, and the ASys Subontology which contains the autonomous systems domain ontological elements. The following sections describe in detail the packages and ontological constructs in each one of the subontologies.

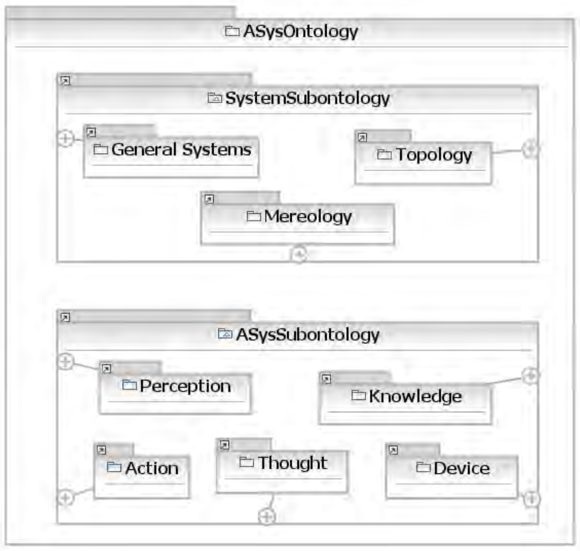


Figure 7.5: ASys Ontology: subontologies

7.4.1.1 System Subontology

The purpose of the System Subontology is to define concepts and relationships to the higher level of abstraction within OASys, to be used when describing any system. These concepts provide a common conceptualisation that can be used to describe any kind of autonomous systems, to be used by autonomous systems engineers.

To organise different aspects that usually co–exist in the characterisation of a system, the concepts have been grouped into different packages (Fig. 7.6). The main package, named General Systems, contains the concepts whose main source is the General Systems Theory as described in Section 2.3. This package is complemented with other two. The first one, called Mereology, contains the concepts to express traditional mereological and meronymic whole–part relations. A second package, called Topology, is used to gather concepts related to topological connections between parts.

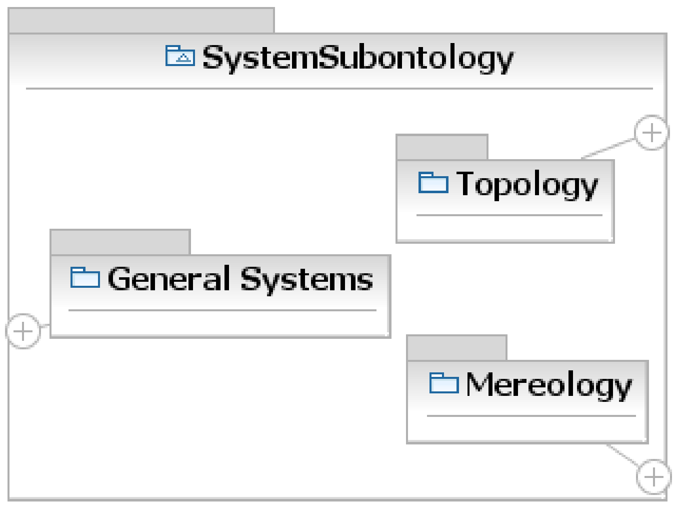


Figure 7.6: System Subontology Packages

General Systems Package

- Purpose and scope: The General Systems Package has been developed to gather concepts related to General Systems Theory, as a theory to describe any kind of systems. This generality was a key element to consider the General Systems Theory as the higher level of abstraction required to define any autonomous system.

- Specification: The elements considered in this package are mainly centered on some of the concepts and ideas thoroughly developed in [Kli69], further detailed in [Kli85], [Kli91], and [KE03].

- Conceptualisation: From the wide range of concepts developed in the General System Theory, only a set of concepts have been considered in the package. The criterion to include these concepts refers to the different possible characteristic traits of a system from both a structural and a behavioural viewpoint, which include the basic definitions (external quantities with the resolution level, a given activity, permanent behaviour, real UC-structure and real ST-structure) considered in [Kli69]. Hence, the General Systems Package has been organised in three different packages (Fig. 7.7): Resolution Level, Structure and Behaviour. The first one provides those concepts related to the quantities, the resolution level and the system's activity. The second one provides the ontological elements to describe the different structures considered in the theory. The last one refers to behaviour related elements.

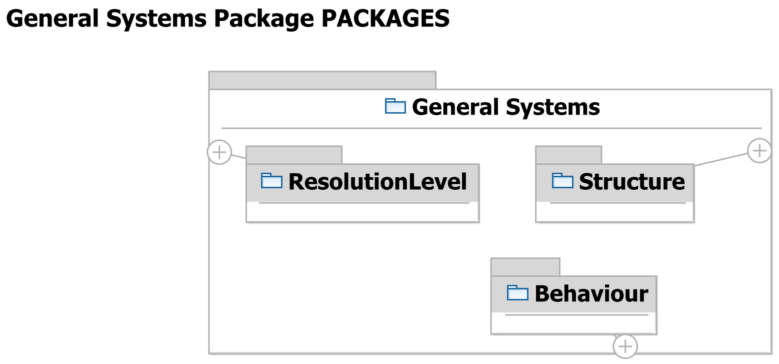


Figure 7.7: General Systems Package subpackages

The conceptualisation tasks obtained a Glossary of Terms described in Sec. 11.1.1.1 in the Annex. Taxonomies of concepts have been looked for to find generalisation relations among concepts (an example given in Fig. 7.8). Furthermore, relationships among concepts where characterised both as tabular and ad–hoc binary representations to express related concepts, name of relation, multiplicity and other properties.

- Formalisation: The concepts, attributes and relationships have been formalised using UML modelling elements. Concepts have been modelled as classes, with attributes as required in the conceptualisation. Relations have been modelled using, from the available range of associations provided by UML, mainly binary directed association (for binary relationships) and association generalization and generalization sets (for taxonomies). The different packages considered in the General Systems Package are depicted in Fig. 7.9, Fig. 7.10, Fig. 7.11, and their definitions are gathered in the Annex Section 11.1.1.1.

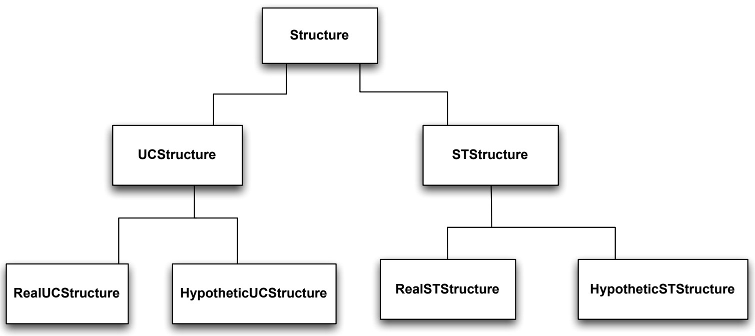


Figure 7.8: An example of taxonomy in the General Systems Package

Mereology Package

- Purpose and scope: The Mereology package has been developed to gather concepts and relationships related to mereological and meronymic aspects, i.e. part–whole relationships, both at meta–level and at system's level.

- Specification: The concepts considered in the package (at the meta–level) are a selection of those already developed by [Bor97], [Gui05], [KA07], and [MWM07]. The aim of the research was not to develop a new mereological theory, but to exploit existing concepts from the different existing Mereological Theories (see Fig. 7.12) in the domain of autonomous systems engineering. The concepts considered in the package at the system's level have been chosen from those defined in [MBW^{+}08].

- Conceptualisation: The meta–level and system's concepts proposed in the different selected sources have been mapped, aligned and merged. This has been made to ensure that the concepts and relations had the same granularity. The analysis also allowed to detect possible overlappings or similar definitions. In the end, a set of concepts, relations and axioms have been chosen as the most appropriate to be used for the characterisation of any (autonomous) system. The glossary of selected terms, as long with the material source, can be seen in Sec. 11.1.1.2 in the Annex.

  Different concepts have been used to describe possible part and whole concepts which exist in any system (Fig. 7.13), to distinguish between aggregate and composite elements. Moreover, both mereological and meronymic relations have been considered. The mereological concepts and relations will allow, among others, to describe structural (aggregation or composition) and spatial relationships (contained in or placed in) among the whole and the different parts. The meta–level names have been mainly chosen from those in [MWM07], with additional concepts from [Bor97] and [KA07]. The meronymic relations have been included to allow the description of part–whole relationships that do not share the same mereological properties, i.e., transitivity among the related elements, which are special interest

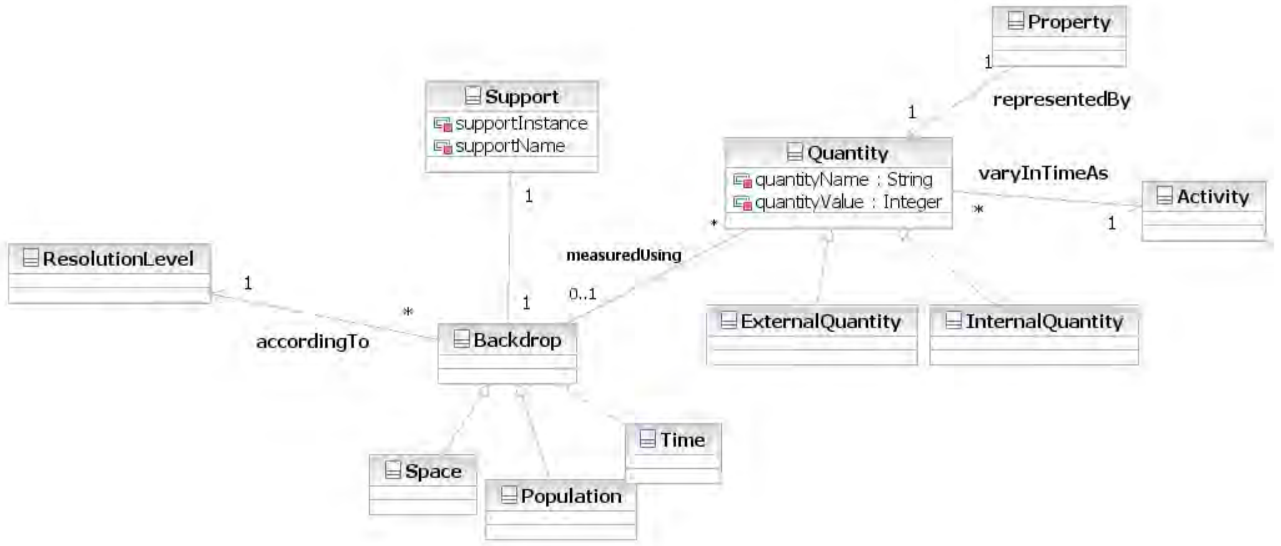


Figure 7.9: General Systems Package: Resolution Level

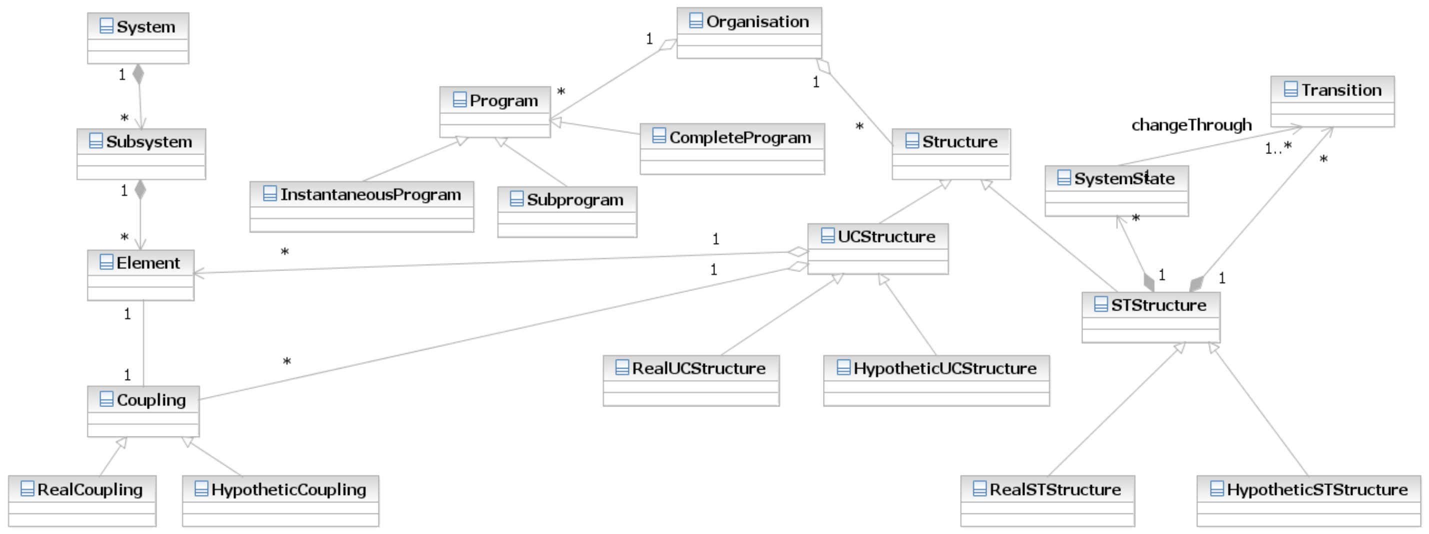


Figure 7.10: General Systems Package: Structure

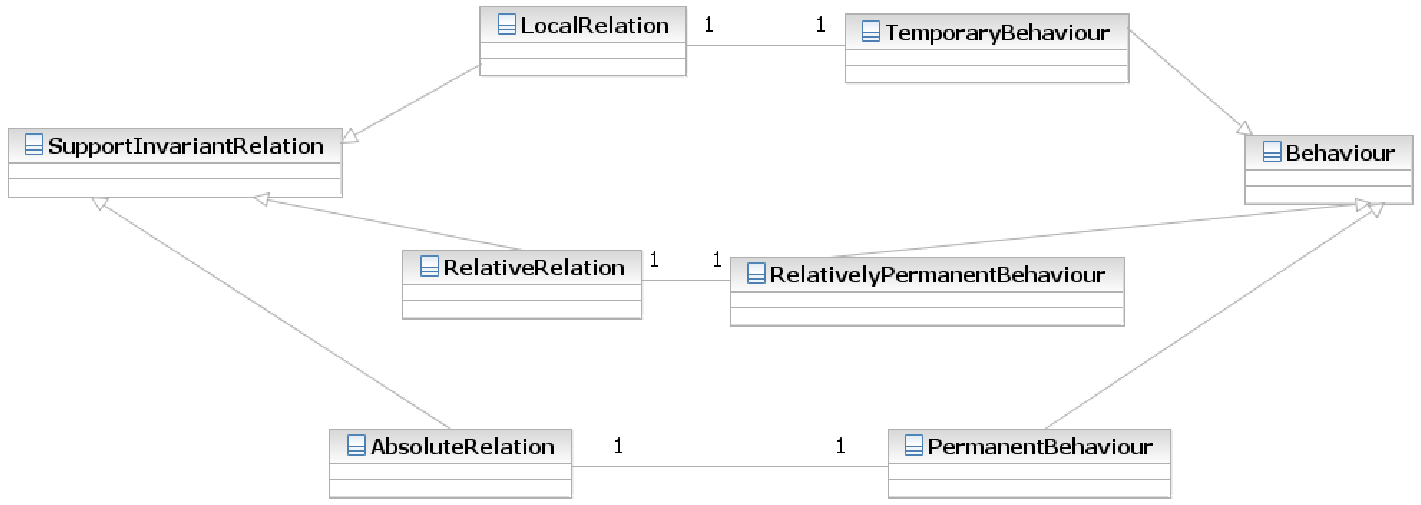


Figure 7.11: General Systems Package: Behaviour

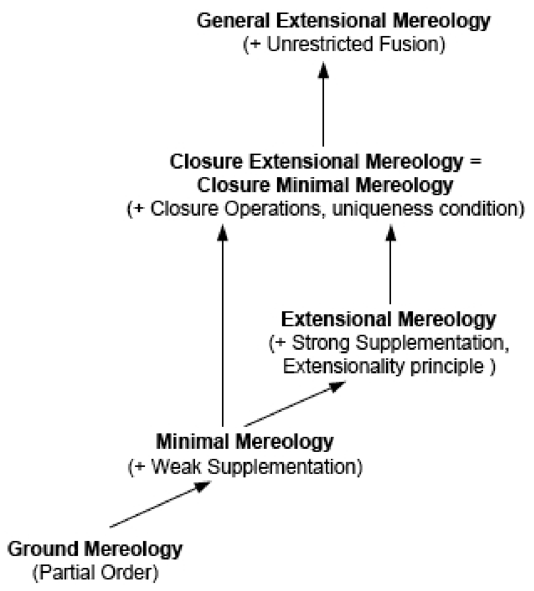

General Extensional Mereology
(+ Unrestricted Fusion)
Closure Extensional Mereology =
Closure Minimal Mereology
(+ Closure Operations, uniqueness condition)
Extensional Mereology
(+ Strong Supplementation,
Extensionality principle)
Minimal Mereology
(+ Weak Supplementation)
Ground Mereology
(Partial Order)

Figure 7.12: Relationships among Mereological Theories (from [Gui05])

for conceptual modelling and cognitive sciences, as proposed in [Gui05] and [KA07]. This latter used as source for the names used in this subontology. A possible taxonomy gathering both mereological and meronymic relations is given in Fig. 7.14.

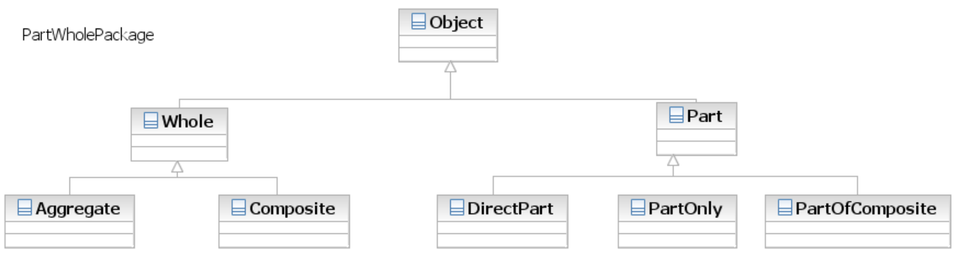


Figure 7.13: A taxonomy of part–whole concepts

- Formalisation: Some of the defined concepts and relations (such as PartOf, PartOfComposite, isPartOf, and isExclusivelyPartOf at meta–level or SubsystemOf, ExclusiveSubsystem at upper level) have been explicitly defined for further extensions of the mereological concepts for other ontology languages that do not provide building primitives for part–whole relations [MWM07]. Nevertheless, UML as modelling

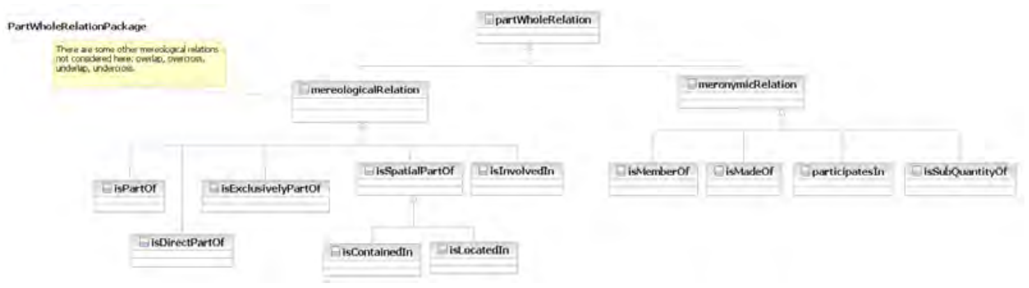


Figure 7.14: A taxonomy of part–whole relations

language owns elements to model aggregation and composition relations by using, respectively, the aggregation and composition associations. The difference between aggregation and composition in UML resides in the existence dependency among whole and parts, with composition being a stronger form of aggregation where the existence of the part depends on the existence of the whole. An example of the implementation of meta–level and system's level concepts and relations is, respectively, shown in Fig. 7.15 and Fig. 7.16.

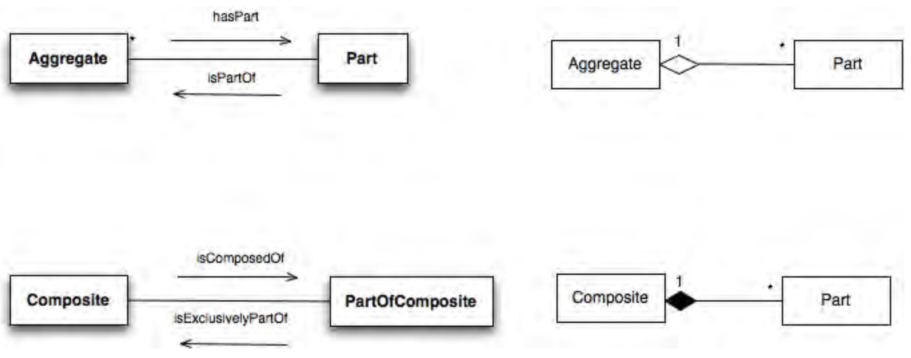


Figure 7.15: Whole-Part relations at meta–level using UML associations

Regarding other mereological and meronymic relations defined in our taxonomy, UML does not provide a direct modelling element. Guizzardi in [Gui05] proposes a modification of the existing UML aggregation and composition, using new icons for the rest of relations. However, Keet [KA07] discards this extension, as it will clutter diagrams (as well as not being supported by UML current specifications or development tools). As initial approach, we will be using directed associations with rolenames to match the ontological names of the relations (additional OCL constraints might be used to express the axioms and constraints related to a particular relation). If further usage of the subontology shows a high necessity of these relations, it might prove useful to sterotype the existing UML aggregations and composition associations with the new additional names and roles.

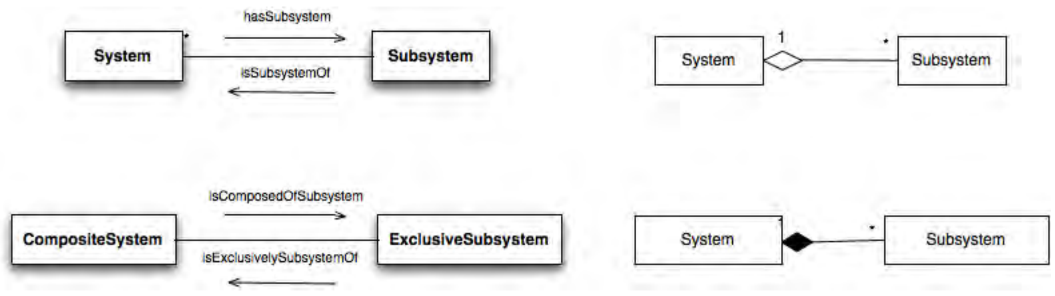


Figure 7.16: Aggregation and composition at the system's level, expressed in UML

Topology Package

- Purpose and scope: The Topology package has been developed with the purpose of collecting those topological concepts and relations existing in a system. When characterising a system, especially from a structural viewpoint, it will be necessary to specify which elements and subsystems are connected with a particular one. This will allow to obtain a topological description of the system under consideration.

- Conceptualisation: The concepts, relations and axioms considered within this package have been selected from those defined in [MWM07] at a meta–level, and [MBW$^+$08] at the system's domain level (where the topological relations ranges and domain are restricted to systems). It was not the aim of the research to come up with new topological terms, as the existing ones already cater for the research focus. A summary of the chosen concepts and relations is given in Sec. 11.1.1.3 of the Annex.

7.4.1.2 ASys Subontology

The purpose of the ASys Subontology is to provide a consensual knowledge model of the domain of autonomous systems, by addressing autonomous systems' structure, behaviour and function. The subontology is organised in different packages (Fig. 7.17) which address the different aspects and functionalities in an autonomous system. Functionalities include the perception, thought, and action activities based on knowledge that an autonomous system performs.

Perception Package

- Purpose and scope: The Perception Package gathers concepts and relationships related to the perception processes that take place in an autonomous system.

- Specification: The elements considered in this package are some of the concepts and ideas thoroughly developed in [UIT06e], and [UIT06a]. A detailed description

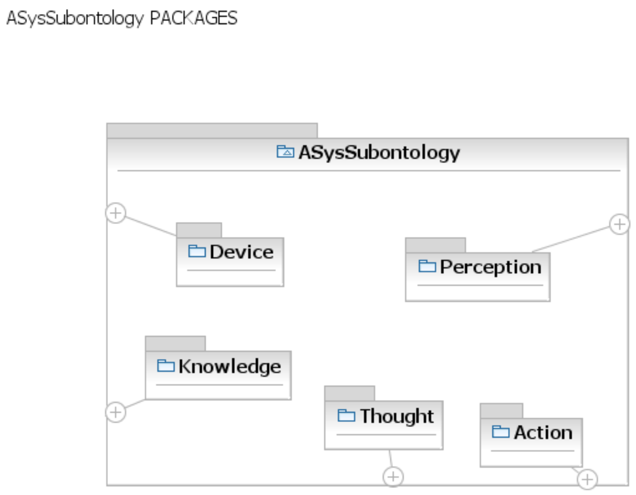


Figure 7.17: ASys Subontology packages

of the processes of perception and sensing, with an emphasis on its features, its context and specially cognitive aspects of such processes is given in [Lóp07].

- Conceptualisation: The conceptualization tasks of the package followed those suggested in the chosen methodology. Firstly, a Glossary of Terms has been obtained, in the form of an intermediate tabular representation. Taxonomies of concepts have been defined. Next, relation among concepts have been characterised both as tabular and ad–hoc binary representations to express related concepts, name of relation, multiplicity and other properties, using when needed the mereological relations. With the previous elements, a concept dictionary has been made (Table 7.7). Finally, instance attributes have been identified.

- Formalisation: the concepts, attributes, relationships and related elements, obtained in the former task, have been formalised using UML modelling elements. Concepts adopted the form of classes. Taxonomy relations have been modelled by using UML generalization associations. Cardinalities have been expressed in the related associations. Mereological relations have been formalised using aggregation and generalization associations. UML does not distinguish between class and instance attributes [GPFLC04b], hence attributes have been specified in the corresponding section on each concept class. The overall formalisation is given in Fig. 7.18 as an UML diagram, with the ontological elements definitions in Annex Section 11.1.2.2.

Knowledge Package

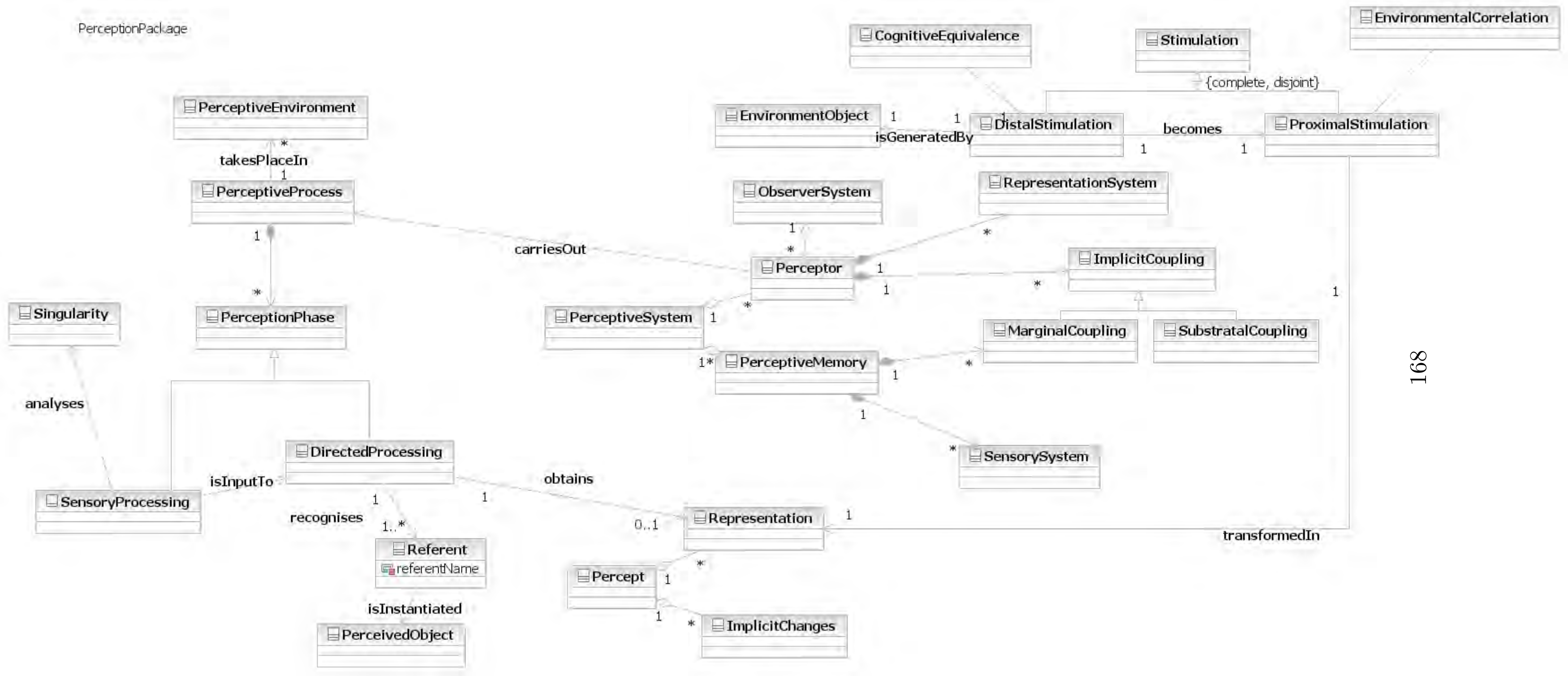
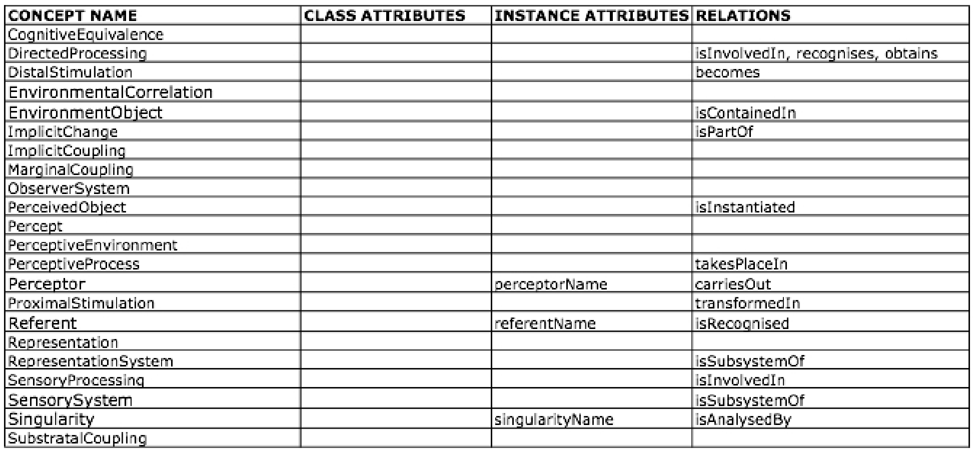


Figure 7.18: ASys Subontology: Perception Package



CONCEPT NAME	CLASS ATTRIBUTES	INSTANCE ATTRIBUTES	RELATIONS
CognitiveEquivalence			
DirectedProcessing			isInvolvedIn, recognises, obtains
DistalStimulation			becomes
EnvironmentalCorrelation			
EnvironmentObject			isContainedIn
ImplicitChange			isPartOf
ImplicitCoupling			
MarginalCoupling			
ObserverSystem			
PerceivedObject			isInstantiated
Percept			
PerceptiveEnvironment			
PerceptiveProcess			takesPlaceIn
Perceptor		perceptorName	carriesOut
ProximalStimulation			transformedIn
Referent		referentName	isRecognised
Representation			
RepresentationSystem			isSubsystemOf
SensoryProcessing			isInvolvedIn
SensorySystem			isSubsystemOf
Singularity		singularityName	isAnalysedBy
SubstratalCoupling			

Table 7.7: Concept Dictionary for the Perception Package

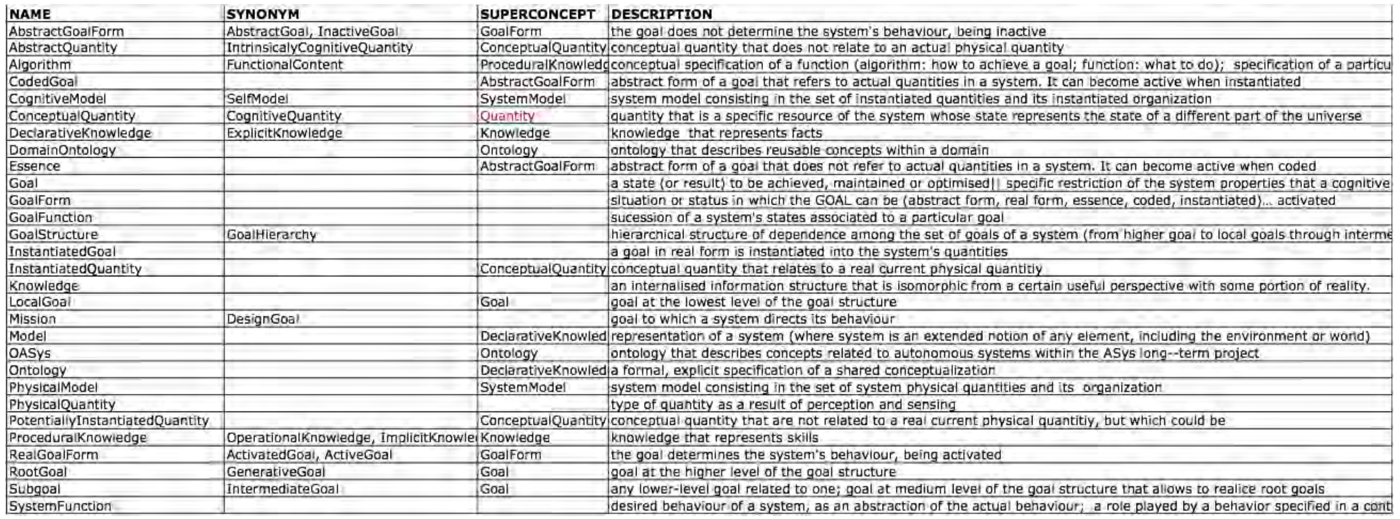
- Purpose and scope: The Knowledge package has been considered to gather concepts and relationships related to knowledge assets used by an autonomous system during its thinking, and evaluating operations.

- Specification: Different publications from ASLab members' production ([SR08], [Lóp07], [HSL08], [SHR07], [AT06b]) have been used as main inputs, whenever complemented with external sources to clarify concepts.

- Conceptualisation: The conceptualization task of the Knowledge Package mapped those of the adopted methodology, obtaining a Glossary of Terms as an intermediate tabular representations as shown in Table 7.8, which is further detailed in Sec. 11.1.2.3 in the Annex.

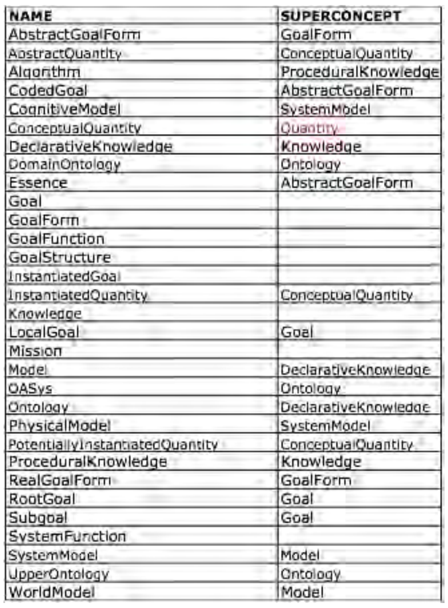
  Taxonomies of concepts have been defined as shown in Table 7.9, where the field Superconcept shows the parent concept of the related concept. A few existing relation among concepts where characterised. Relevant attributes have been identified for the key concept of Goal. A concept dictionary specified each concept, by detailing its attributes and relations. Intermediate graphical representations have been used to characterise concept taxonomies, one example of such is shown in Fig. 7.19.

  Knowledge has been defined as internalised information structure that is isomorphic from a certain useful perspective with some portion of reality. Initially, the information structures, which an autonomous system uses for thinking and evaluating operations, have been acquired by definition and learning (e.g. goals, algorithms, etc) or by transformation from inputs of other processes (e.g. conceptual quantities from physical quantities) at a conceptual level. Later on, the conceptual elements will be grounded, as required during other operations, into real physical elements



NAME	SYNONYM	SUPERCONCEPT	DESCRIPTION
AbstractGoalForm	AbstractGoal, InactiveGoal	GoalForm	the goal does not determine the system's behaviour, being inactive
AbstractQuantity	IntrinsicalyCognitiveQuantity	ConceptualQuantity	conceptual quantity that does not relate to an actual physical quantity
Algorithm	FunctionalContent	ProceduralKnowledg	conceptual specification of a function (algorithm: how to achieve a goal; function: what to do); specification of a particu
CodedGoal		AbstractGoalForm	abstract form of a goal that refers to actual quantities in a system. It can become active when instantiated
CognitiveModel	SelfModel	SystemModel	system model consisting in the set of instantiated quantities and its instantiated organization
ConceptualQuantity	CognitiveQuantity	Quantity	quantity that is a specific resource of the system whose state represents the state of a different part of the universe
DeclarativeKnowledge	ExplicitKnowledge	Knowledge	knowledge that represents facts
DomainOntology		Ontology	ontology that describes reusable concepts within a domain
Essence		AbstractGoalForm	abstract form of a goal that does not refer to actual quantities in a system. It can become active when coded
Goal			a state (or result) to be achieved, maintained or optimised|| specific restriction of the system properties that a cognitive
GoalForm			situation or status in which the GOAL can be (abstract form, real form, essence, coded, instantiated)... activated
GoalFunction			sucession of a system's states associated to a particular goal
GoalStructure	GoalHierarchy		hierarchical structure of dependence among the set of goals of a system (from higher goal to local goals through interme
InstantiatedGoal			a goal in real form is instantiated into the system's quantities
InstantiatedQuantity		ConceptualQuantity	conceptual quantity that relates to a real current physical quantitiy
Knowledge			an internalised information structure that is isomorphic from a certain useful perspective with some portion of reality.
LocalGoal		Goal	goal at the lowest level of the goal structure
Mission	DesignGoal		goal to which a system directs its behaviour
Model		DeclarativeKnowled	representation of a system (where system is an extended notion of any element, including the environment or world)
OASys		Ontology	ontology that describes concepts related to autonomous systems within the ASys long--term project
Ontology		DeclarativeKnowled	a formal, explicit specification of a shared conceptualization
PhysicalModel		SystemModel	system model consisting in the set of system physical quantities and its organization
PhysicalQuantity			type of quantity as a result of perception and sensing
PotentiallyInstantiatedQuantity		ConceptualQuantity	conceptual quantity that are not related to a real current physical quantitiy, but which could be
ProceduralKnowledge	OperationalKnowledge, ImplicitKnowle	Knowledge	knowledge that represents skills
RealGoalForm	ActivatedGoal, ActiveGoal	GoalForm	the goal determines the system's behaviour, being activated
RootGoal	GenerativeGoal	Goal	goal at the higher level of the goal structure
Subgoal	IntermediateGoal	Goal	any lower-level goal related to one; goal at medium level of the goal structure that allows to realice root goals
SystemFunction			desired behaviour of a system, as an abstraction of the actual behaviour; a role played by a behavior specified in a cont

Table 7.8: Excerpt of the Glossary of Terms for Knowledge Package



NAME	SUPERCONCEPT
AbstractGoalForm	GoalForm
AbstractQuantity	ConceptualQuantity
Algorithm	ProceduralKnowledge
CodedGoal	AbstractGoalForm
CognitiveModel	SystemModel
ConceptualQuantity	Quantity
DeclarativeKnowledge	Knowledge
DomainOntology	Ontology
Essence	AbstractGoalForm
Goal	
GoalForm	
GoalFunction	
GoalStructure	
InstantiatedGoal	
InstantiatedQuantity	ConceptualQuantity
Knowledge	
LocalGoal	Goal
Mission	
Model	DeclarativeKnowledge
OASys	Ontology
Ontology	DeclarativeKnowledge
PhysicalModel	SystemModel
PotentiallyInstantiatedQuantity	ConceptualQuantity
ProceduralKnowledge	Knowledge
RealGoalForm	GoalForm
RootGoal	Goal
Subgoal	Goal
SystemFunction	
SystemModel	Model
UpperOntology	Ontology
WorldModel	Model

Table 7.9: Taxonomy of Concepts in Knowledge Subontology

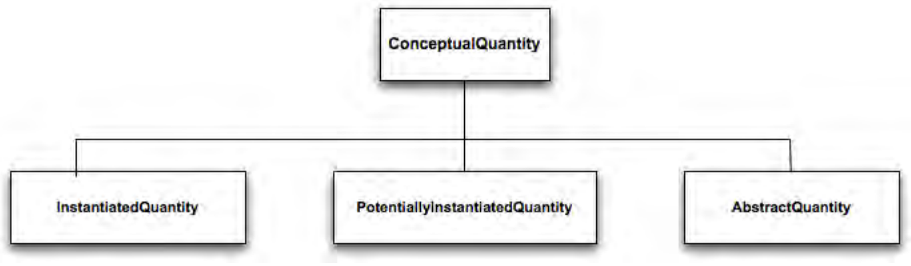


Figure 7.19: An example of taxonomy in Knowledge Package

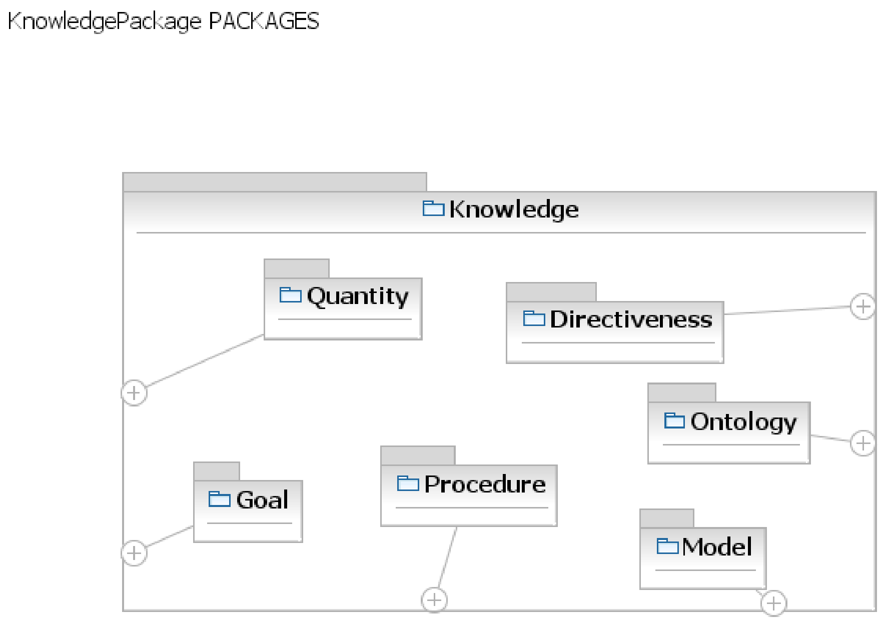
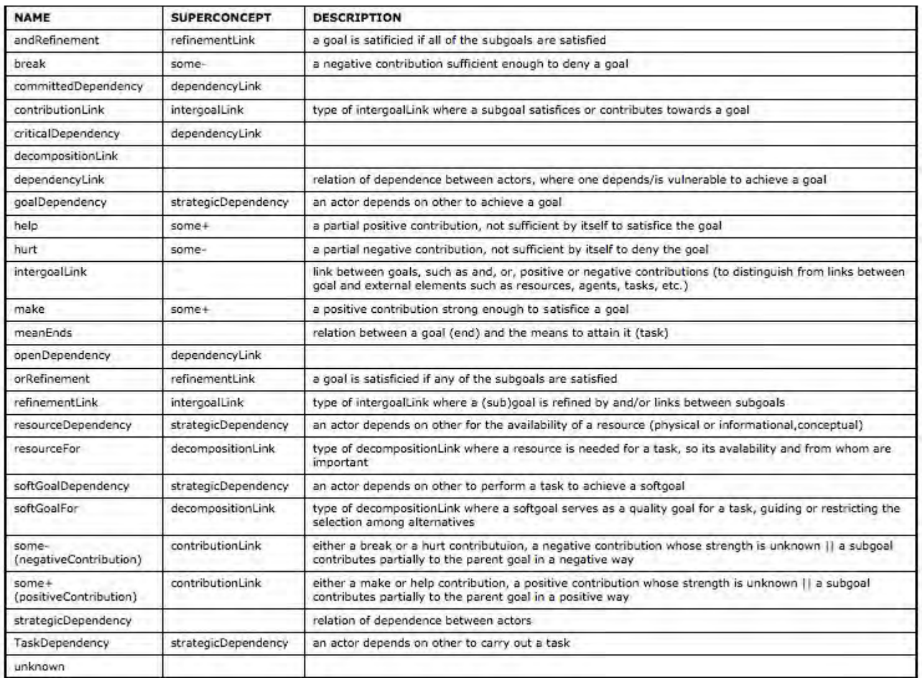


Figure 7.20: Packages in Knowledge Package

in the autonomous system. Hence the Knowledge Package has been organised in different packages to address the different kinds of information structure (Fig. 7.20).

The Quantity Package contains different Quantity related terms. ConceptualQuantity represents the quantities used by the autonomous system on its operations, being either InstantiatedQuantity, PotentiallyInstantiatedQuantity, and AbstractQuantity according to its relation, possible relation or lack of relation with real physical quantities. The Model Package refers to the models used by the autonomous system, both the SystemModel and the WorldModel. The Ontology Package contains concepts about the different ontologies an autonomous system might use as asset during its operation. The Directiveness Package conceptualises terms related to the purposiveness of an autonomous system. The Goal Package shows the different goals the autonomous system should aim at during its operation. Goals are actually organised in GoalStructure, to model the different goal level of abstraction (from RootGoal to LocalGoal) and their relationships (refinement, contribution, dependency, etc). The Knowledge Package only shows the fact that the GoalStructure is built up with different possible types of Goal, not specifying the actual relationships among them. This latter will be made during the use of OASys, where Goal and possible GoalRelationship (as specialisation of mereological relations) will be specified to match those of the real system (see Table. 7.10).



NAME	SUPERCONCEPT	DESCRIPTION
andRefinement	refinementLink	a goal is satificied if all of the subgoals are satisfied
break	some-	a negative contribution sufficient enough to deny a goal
committedDependency	dependencyLink	
contributionLink	intergoalLink	type of intergoalLink where a subgoal satisfices or contributes towards a goal
criticalDependency	dependencyLink	
decompositionLink		
dependencyLink		relation of dependence between actors, where one depends/is vulnerable to achieve a goal
goalDependency	strategicDependency	an actor depends on other to achieve a goal
help	some+	a partial positive contribution, not sufficient by itself to satisfice the goal
hurt	some-	a partial negative contribution, not sufficient by itself to deny the goal
intergoalLink		link between goals, such as and, or, positive or negative contributions (to distinguish from links between goal and external elements such as resources, agents, tasks, etc.)
make	some+	a positive contribution strong enough to satisfice a goal
meanEnds		relation between a goal (end) and the means to attain it (task)
openDependency	dependencyLink	
orRefinement	refinementLink	a goal is satisficied if any of the subgoals are satisfied
refinementLink	intergoalLink	type of intergoalLink where a (sub)goal is refined by and/or links between subgoals
resourceDependency	strategicDependency	an actor depends on other for the availability of a resource (physical or informational,conceptual)
resourceFor	decompositionLink	type of decompositionLink where a resource is needed for a task, so its avalability and from whom are important
softGoalDependency	strategicDependency	an actor depends on other to perform a task to achieve a softgoal
softGoalFor	decompositionLink	type of decompositionLink where a softgoal serves as a quality goal for a task, guiding or restricting the selection among alternatives
some- (negativeContribution)	contributionLink	either a break or a hurt contributuion, a negative contribution whose strength is unknown || a subgoal contributes partially to the parent goal in a negative way
some+ (positiveContribution)	contributionLink	either a make or help contribution, a positive contribution whose strength is unknown || a subgoal contributes partially to the parent goal in a positive way
strategicDependency		relation of dependence between actors
TaskDependency	strategicDependency	an actor depends on other to carry out a task
unknown		

Table 7.10: Goal Relationships for Goal Structure

The Procedure Package has been been conceptualised as having Algorithm and GoalFunction as elements. Algorithm represents the conceptual form of the functions of the autonomous systems. The actual implementation of the Algorithm will be made later on.

- Formalisation: UML modelling elements have been used to formalize the different packages conceptualized in the former phase, each one as an UML class diagram (Fig. 7.21a, Fig.7.21b, Fig. 7.21c, Fig. 7.21d, Fig.7.21e and Fig. 7.21f).

Thought Package

- Purpose, scope and specification: The concepts enclosed in this package refer to the main ideas described in [UIT06d], and further defined in [UIT06b], where the autonomy of a system is defined by:

  > performance, standing for efficient algorithms, capacity for grounding, capacity for algorithm generation and capacity for objective (goal) generation... They refer to the substrate on which its implemented, and the knowledge and abstract processes it carries out

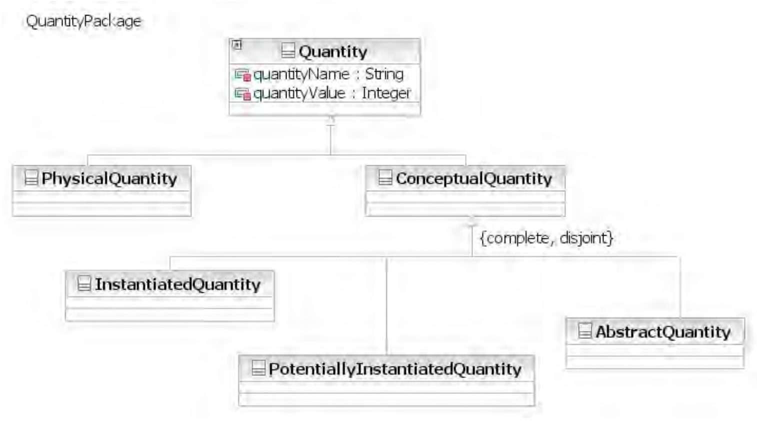
  The Thought package focused, therefore, on regarding thinking as any knowledge–based abstract processing where knowledge refers to the concepts defined in the Knowledge package (Fig. 7.21).

- Conceptualisation: Bearing in mind the aforementioned described purpose and scope, the Thought Package paid special attention to Goal– and Algorithm–related abstract processes concept, which follow under the concept ConceptualOperation considered in the Action Package. For Goal related ones, concepts such as GoalGeneration, GoalLifeCycle, and GoalReconfiguration have been considered. For Algorithm based ones, FunctionalDecomposition which implies processes carried out with Algorithms have been conceptualised.

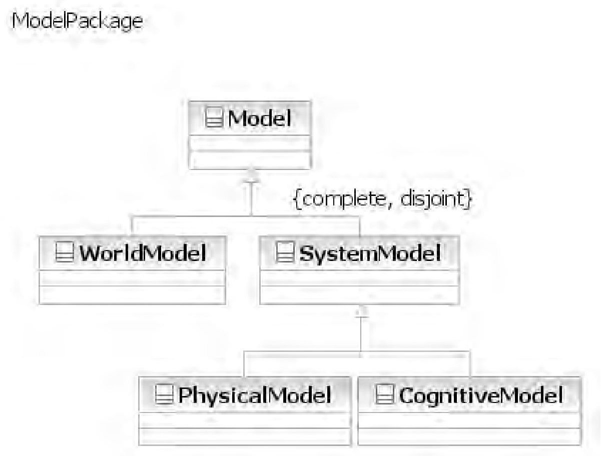
- Formalisation: UML modelling elements have been used to formalize the package elements conceptualized in the previous stage. Concepts have been formalised as UML classes. Whole–part relations have been formalised using the mereological relations already defined, especially the involvedIn as this package refers to process taking part in other process. The UML diagram shown in Fig. 7.22, whose ontological elements are specified in the Annex Sec. 11.1.2.4.

Action Package

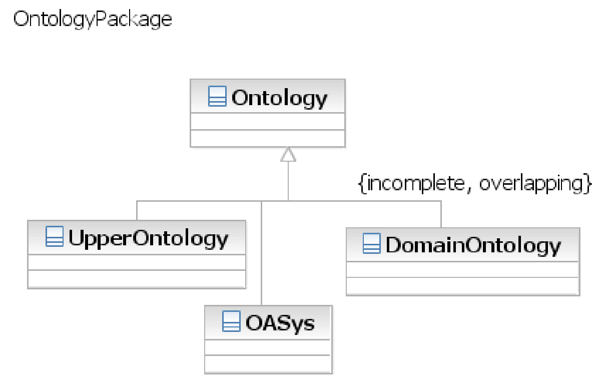
- Purpose and scope: the Action package defines the ontological elements related to the different activities and entities involved in the generation of actions of an autonomous system.



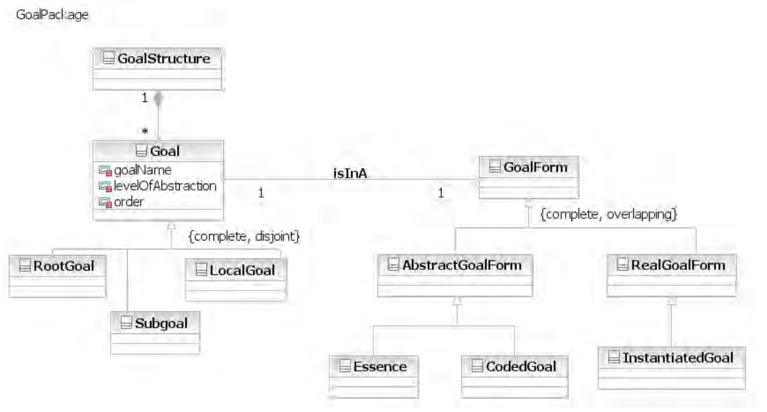
(a) Quantity Package



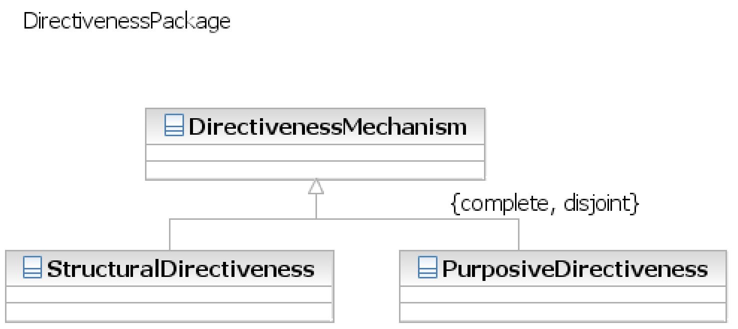
(b) Model Package



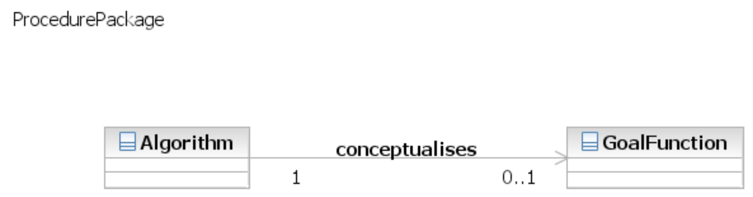
(c) Ontology Package



(d) Goal Package



(e) Directiveness Package



(f) Procedure Package

Figure 7.21: Packages in Knowledge Package

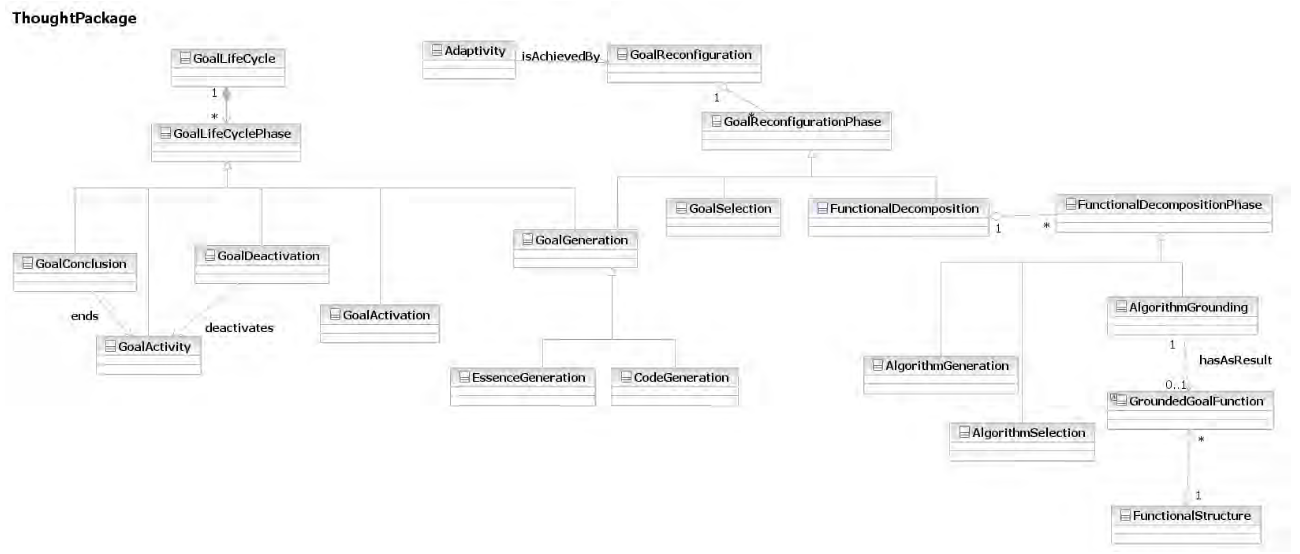


Figure 7.22: ASys Subontology: Thought package

- Specification: as sources [Lóp07] and [HSL08] that played a specially role for the conceptualization of the concepts in this package.

- Conceptualisation: A Glossary of Terms has been obtained as the main result of the conceptualization, which is partially shown in Table 7.11, and further detailed in Annex Sec. 11.1.2.5. The Glossary has been complemented with taxonomies of concepts and graphical representation of relationships among concepts in the package.

  Key point in this package is the conceptualisation of the aspects of what, who and how the autonomous system acts. Firstly, the concept Operation is introduced to reflect the possible different tasks or activities which might be carried out by an autonomous system. These operations involve the processes upon system's quantities. A first group considers operations focused on only conceptual quantities, conceptualised as ConceptualOperation. Another important type of operation is that of Actuation, to conceptualise those operations that involved some kind of transformation from conceptual quantities to physical quantities such as Perception, Grounding and Embodiment. A third concept refers to possible operations with already grounded quantities, defined as GroundedOperation. Finally, there are some operations which imply some kind of interaction with the system's environment, such as Sensing and Action. All these concepts have been related in the taxonomy shown in Fig. 7.23.

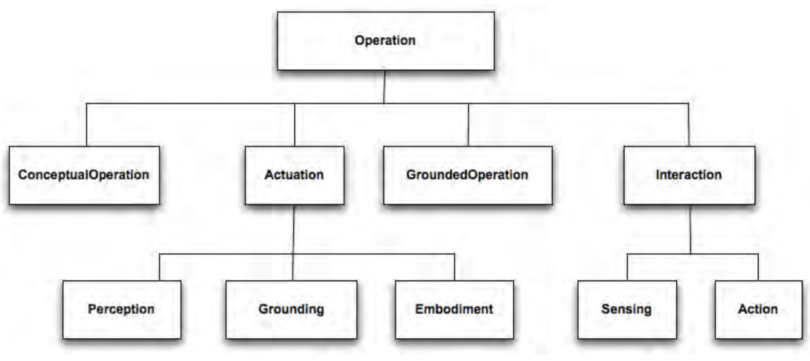
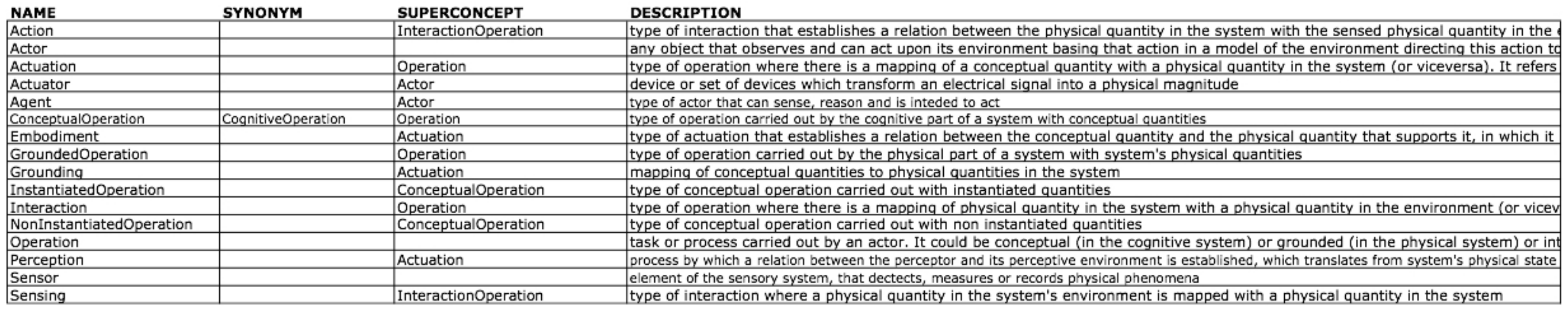


Figure 7.23: Operation taxonomy in the Action package

  Secondly, the conceptualisation of the elements that perform those operations, defined as Actor in general. This concept, Actor, can be further detailed as required in the concepts of Agent (with a software focus) and Actuator (with a hardware orientation). Finally, within the Action Package, concepts related to the control aspects of autonomous system's operation have also been considered.

- Formalisation: UML modelling elements, classes for concepts and generalization



NAME	SYNONYM	SUPERCONCEPT	DESCRIPTION
Action		InteractionOperation	type of interaction that establishes a relation between the physical quantity in the system with the sensed physical quantity in the
Actor			any object that observes and can act upon its environment basing that action in a model of the environment directing this action t
Actuation		Operation	type of operation where there is a mapping of a conceptual quantity with a physical quantity in the system (or viceversa). It refers
Actuator		Actor	device or set of devices which transform an electrical signal into a physical magnitude
Agent		Actor	type of actor that can sense, reason and is inteded to act
ConceptualOperation	CognitiveOperation	Operation	type of operation carried out by the cognitive part of a system with conceptual quantities
Embodiment		Actuation	type of actuation that establishes a relation between the conceptual quantity and the physical quantity that supports it, in which it
GroundedOperation		Operation	type of operation carried out by the physical part of a system with system's physical quantities
Grounding		Actuation	mapping of conceptual quantities to physical quantities in the system
InstantiatedOperation		ConceptualOperation	type of conceptual operation carried out with instantiated quantities
Interaction		Operation	type of operation where there is a mapping of physical quantity in the system with a physical quantity in the environment (or vicev
NonInstantiatedOperation		ConceptualOperation	type of conceptual operation carried out with non instantiated quantities
Operation			task or process carried out by an actor. It could be conceptual (in the cognitive system) or grounded (in the physical system) or int
Perception		Actuation	process by which a relation between the perceptor and its perceptive environment is establised, which translates from system's physical state
Sensor			element of the sensory system, that dectects, measures or records physical phenomena
Sensing		InteractionOperation	type of interaction where a physical quantity in the system's environment is mapped with a physical quantity in the system

Table 7.11: Excerpt of the Action package Glossary of Terms

association and sets for taxonomies, were used to formalize the concepts and relationships conceptualized in the previous stage, as the UML class diagram shown in Fig. 7.24.

Device Package

- Purpose and scope: This package provides the concepts to describe the different devices, and their properties, that are part of an autonomous system (as refinement of the systems' general concepts).

- Specification: different publications such as [RJB04], [HH08], [dlM09b] have been used as main inputs. They focus on describing the physical features and structures of the different systems and subsystems, as well as the autonomous system required functionalities.

- Conceptualisation: A Glossary of Terms has been obtained as the main result of the conceptualization, complemented with taxonomies of concepts and graphical representation of relationships among concepts in the subontology.

  The concepts gathered in this layer focus on the terms and relationships currently used when referring to the applications under development. General terms such as system or subsystem could, generally, be specialized in concepts such as Node, Device, Component, Artifact and so forth, which are closer to the software definition and hardware deployment. The different types of connections, connection points and ports considered can be specialised to model physical (devices interconnected) or informational links (such as RS-232, Ethernet, etc) between devices.

  It must be pointed out that although the conceptualisation phase should be independent of the formalization and implementation phases, an exception had to be made during the conceptualisation of this layer. The concept of Device posed some controversy. On one hand, device is a concept well–established among developers and engineers, who use it loosely to refer to any physical element within a system's development. On the other hand, device in the UML language constraints this meaning to be a physical *computational* resource where artifacts may be deployed for execution. As the ASys project will be using the two kinds of devices, the duplicity of meaning made it necessary to rearrange it by considering a ComputationalDevice Package (to consider computational devices such as computers, servers, etc) and PhysicalDevice Package (for other types of devices such as cameras, wrists, GPS, etc) as part of the Device Package (Fig. 7.25).

- Formalisation: UML elements (classes, attributes, associations) have been used to formalize the concepts and relationships conceptualized in the previous stage. The ComputationalDevice Package can be seen in Fig. 7.26, and the PhysicalDevice Package in Fig. 7.27, being described in the Annex Sec. 11.1.2.1.

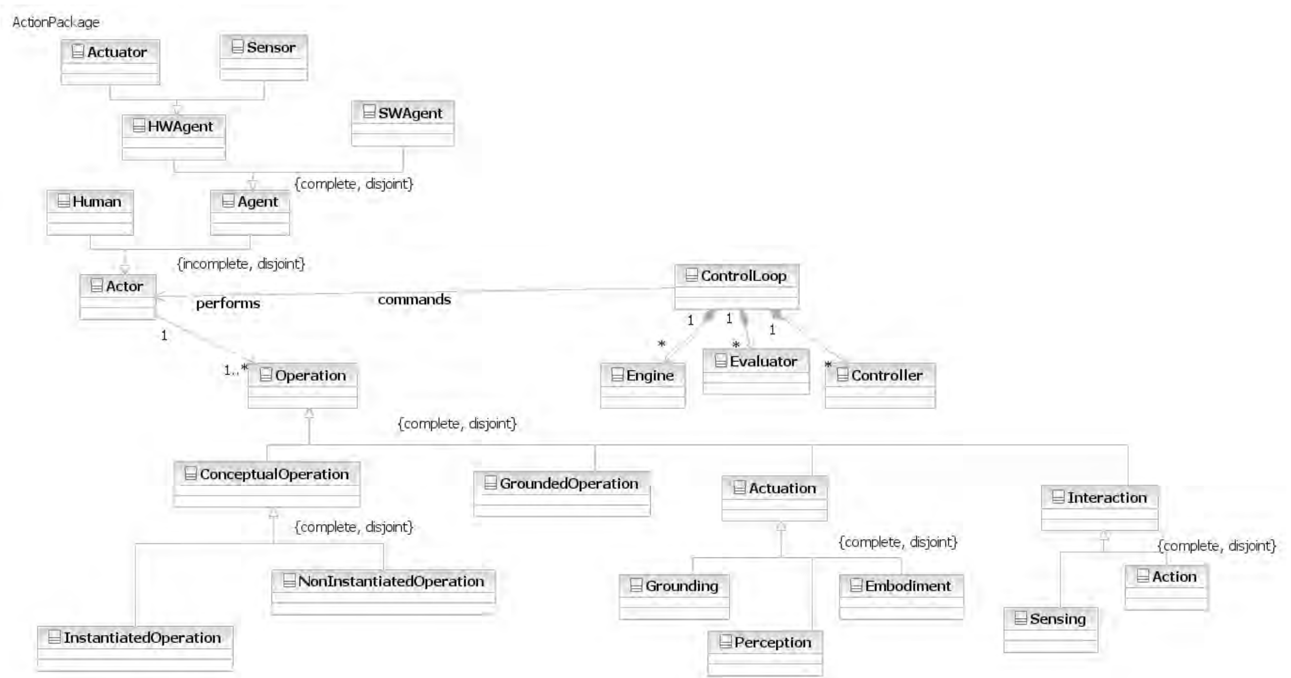


Figure 7.24: ASys Subontology: Action package

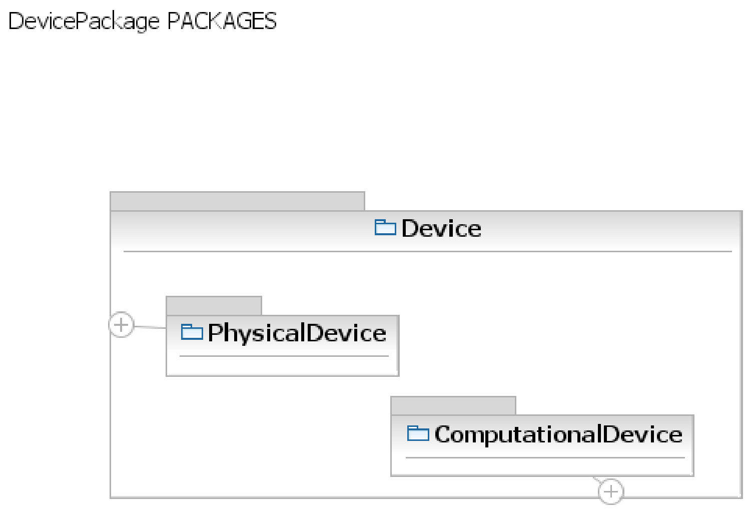


Figure 7.25: Packages in Device Package

7.4.2 ASys Engineering Ontology

This ontology is the second part of OASys, dedicated to system's engineering related ontological elements. It comprises two subontologies to address at a different level of abstraction the ontological elements to describe the autonomous system's engineering development process (Fig. 7.28): the System Engineering Subontology gathers the concepts related to general systems' engineering, whereas the ASys Engineering Subontology gathers in a similar way the elements for ASys engineering, as specialisation of the former ones.

7.4.2.1 System Engineering Subontology

This subontology gather concepts and relationships related to process and software system's engineering. It is based on different metamodels, specifications and glossaries which have been long used for software–based developments. The subontology contains different packages to address different aspects to consider, as needed, within a system's development: requirements, engineering process, perspective, and model driven (Fig.7.29).

Requirement Package

- Purpose and specification: The Requirement Package gathers the concepts about the system's requirements which are usually established during a system's engineering process. The package elements have been chosen from already existing sources on requirements definition and classification such as [IEE90], [Obj09b] and [Gro08a]. The first one has been chosen as a traditional source, which has been

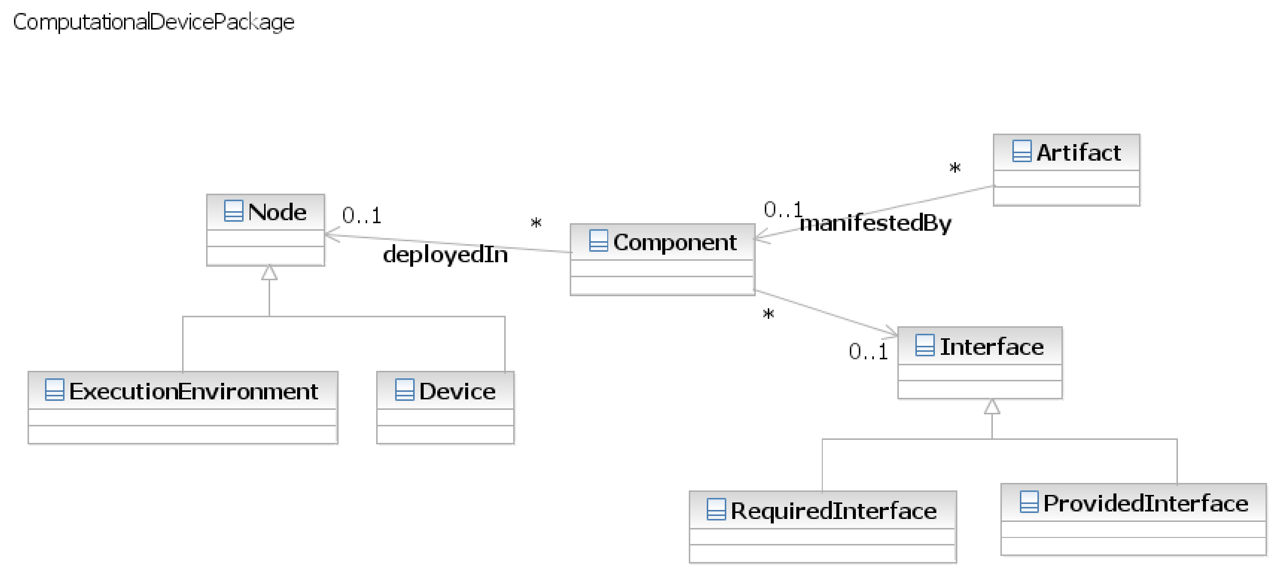


Figure 7.26: DevicePackage: ComputationalDevice Package

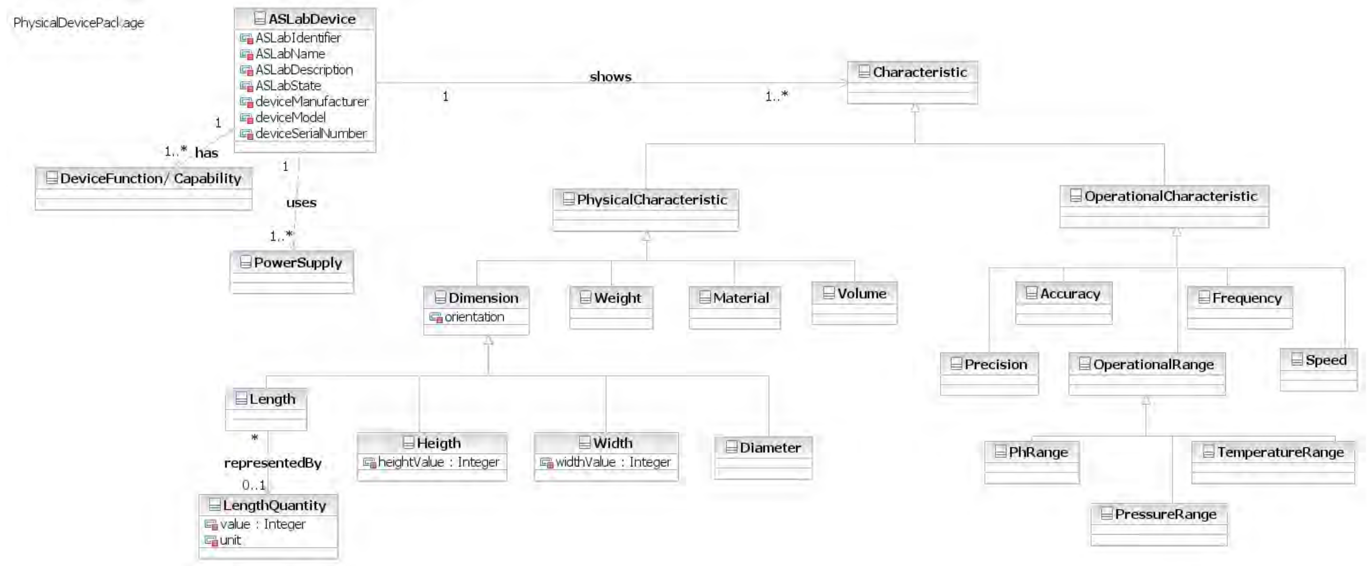


Figure 7.27: DevicePackage: PhysicalDevice Package

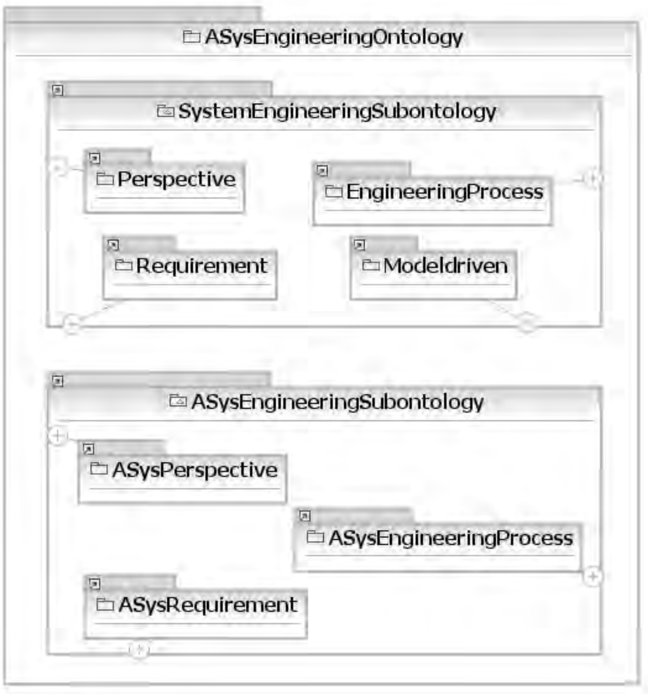


Figure 7.28: ASys Engineering Ontology

SystemEngineeringSubontology PACKAGES

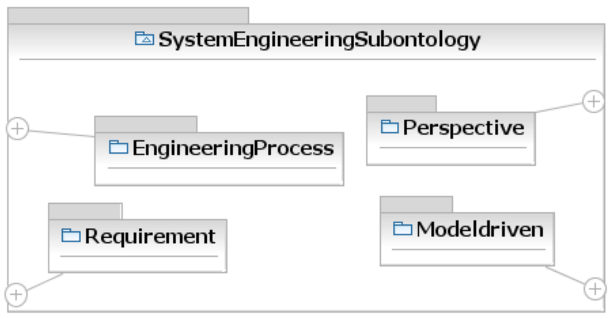


Figure 7.29: System Engineering Subontology Packages

extended by UML and SysML sources especially focused on system's engineering.

- Conceptualisation: this phase defined the concepts, taxonomies, relations, and attributes related to requirements as defined in the aforementioned sources. A Requirement is defined as a capability or condition that must (or should) be satisfied by a system. It is generally identified by a name, some description text, its source, a level of risk (high, medium, low), and some kind of method to verify it (test, analysis, demonstration or inspection). Requirements have been lately categorised as functional, interface, performance, physical or design requirements (Fig. 7.30).

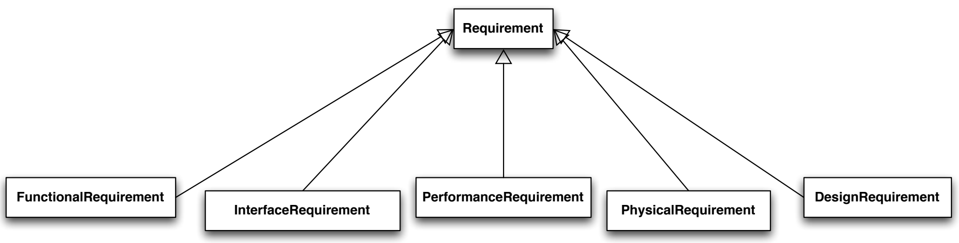


Figure 7.30: Requirements Taxonomy

Requirements are usually captured using use cases, as a mean to describe system or subsystem's requirements considering a particular subject (system or subsystem) and use case actors, as external entities to the system which interact with it. A requirement can be derived from other, or it could be satisfied or verified by other elements.

- Formalisation: The different concepts and relationships have been formalised into a UML class diagram (Fig. 7.31), and their definitions can be found in the Annex Sec. 11.2.1.1.

Engineering Process Package

- Purpose and specification: This package collects general concepts to describe the stages and steps in a system's engineering process taking as basis the Software and Systems Process Engineering Meta-model (SPEM 2.0) [Gro08b], which is a a process engineering meta-model developed by the OMG. This conceptual framework is used to define software and systems development processes and their components. It has been chosen as main source taking into account that SPEM has been designed to define any software and systems development process, without adding specific features for particular development domains or disciplines. SPEM 2.0 could be also used as a UML profile, i.e. an extension of the UML metamodel, thus allowing a future UML implementation through stereotypes of the necessary concepts of ASys Engineering.

- Conceptualisation: The concepts and relationships in this subontology aim at defining general terms to be used in any system's engineering process, without constrain-

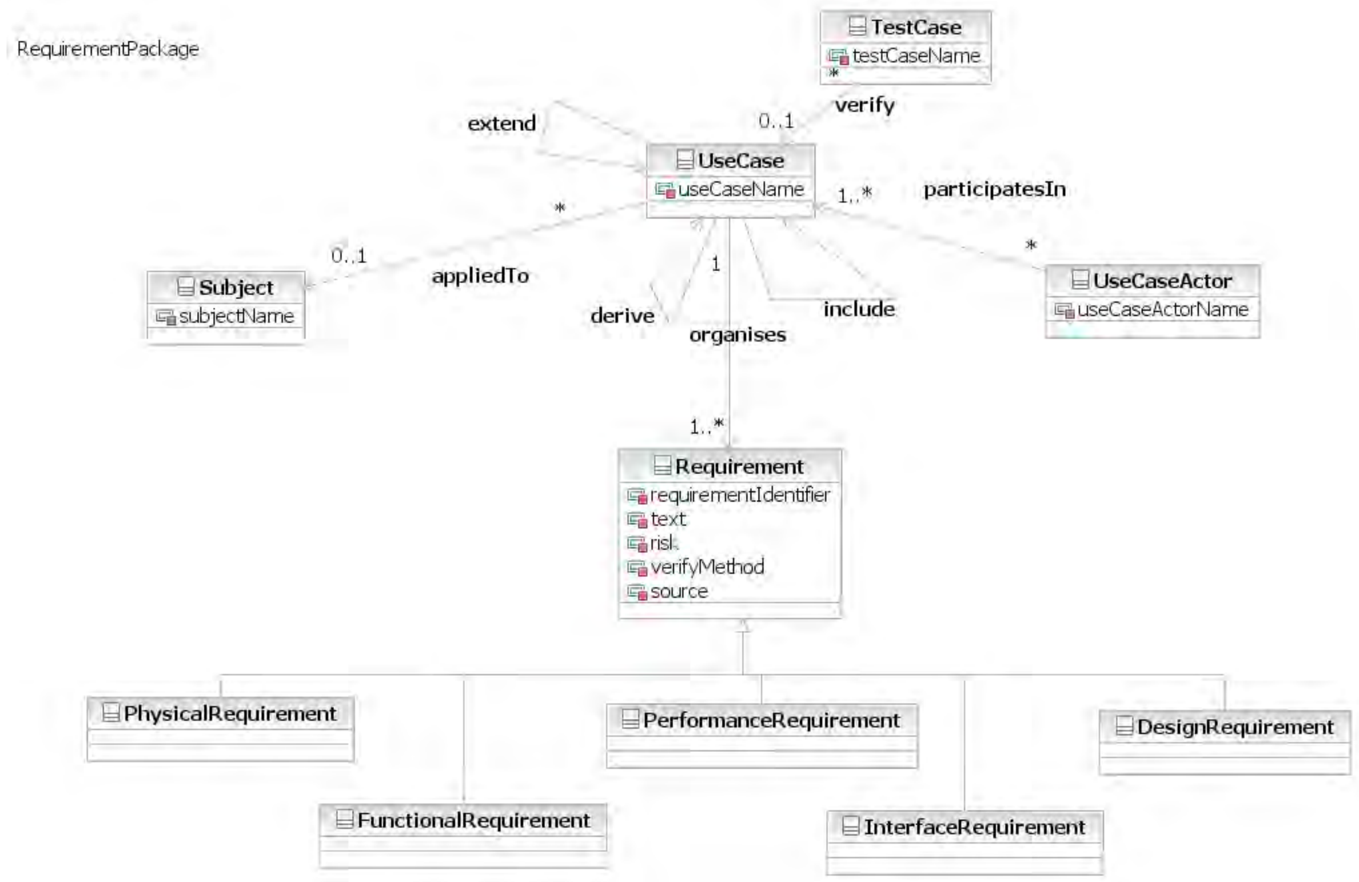


Figure 7.31: SystemEngineering Subontology: Requirement Package

ing it to a particular engineering process. The underlying idea it to re-use these concepts to characterise different engineering processes that will be further develop for ASys engineering and construction, or parts of an ASys.

- Formalisation: The different ontological elements have been formalised in an UML class diagram showing concepts as classes, relation as UML associations or generalization associations (Fig. 7.32). The concepts, relations and additional package elements definitions are given in the Annex Sec. 11.2.1.2.

Perspective Package

The Perspective Package gathers the ontology elements necessary to consider the different possible views, and viewpoints when developing any system's architecture. The concepts and relations have been chosen from those described in [Ins00] which provides a conceptual framework with the most common terms for this purpose. The Perspective Package has been formalised as an UML diagram (Fig. 7.33), whose elements are summarised in the Annex Sec. 11.2.1.3.

Model–driven Package

The Model–driven Package provides fundamental concepts and relations related to model–driven (software) engineering, which refers to a range of development approaches that are based on the use of (software) modelling as a primary form of expression. The ontological elements in this package have been chosen from those in [SV06], and [Obj03]. They will be later specialised and refined for the ASys project research efforts in the ASys Engineering Subontology.

The formalisation of the concepts and relationships has been made as an UML class diagram (Fig. 7.34), and the chosen ontological elements are summarised in the Annex Sec. 11.2.1.4.

7.4.2.2 ASys Engineering Subontology

The ASys Engineering Subontology is the ontology which addresses autonomous systems' engineering processes, by gathering concepts and relationships to describe a generic engineering process or lifecycle for an autonomous system. It is a refinement or specialisation of the concepts already defined in the System Engineering Subontology, organised in different packages (Fig. 7.35).

Generic means that it defines concepts and relationships related to an ASys engineering process, in a way general enough to be adjusted to future, custom processes. It is not the aim of the research to fully develop an ultimate engineering process for Autonomous Systems, but to suggest a possible process suitable enough for initial developments that will exemplify the use of OASys. Specific processes concepts belonging to later developed processes or lifecycles could be easily included in this subontology as further packages.

ASys Requirement Package

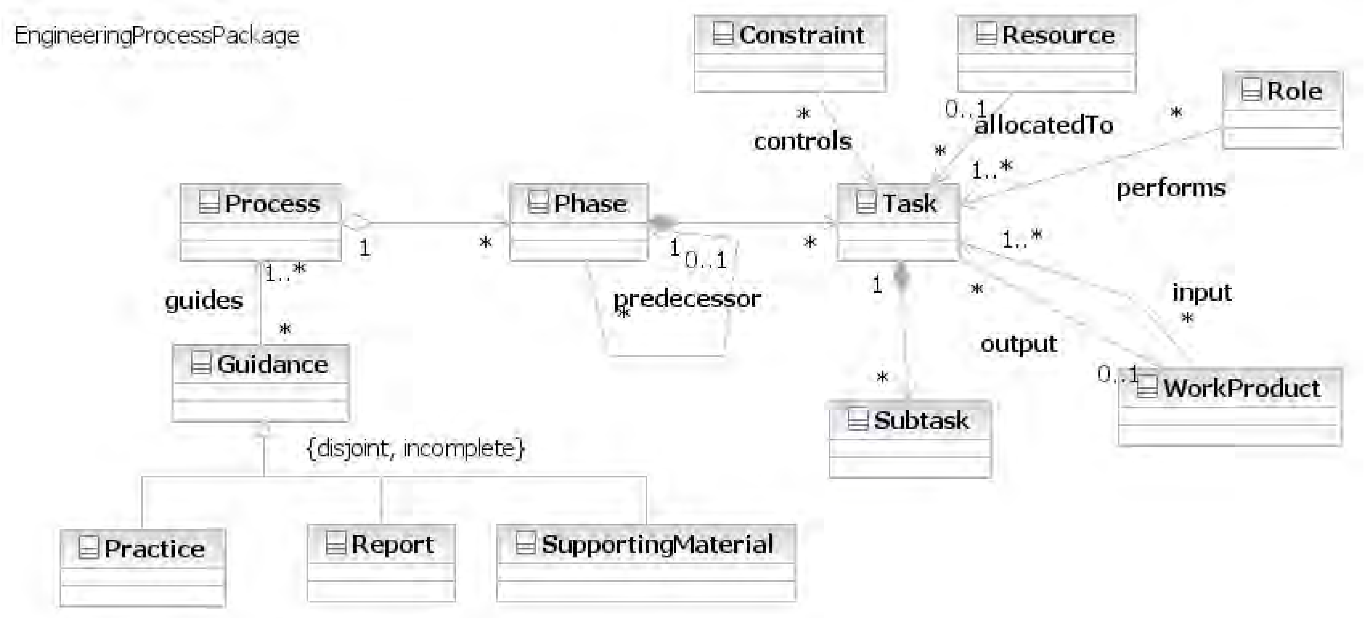


Figure 7.32: SystemEngineering Subontology: Engineering Process Package adapted from [Gro08b]

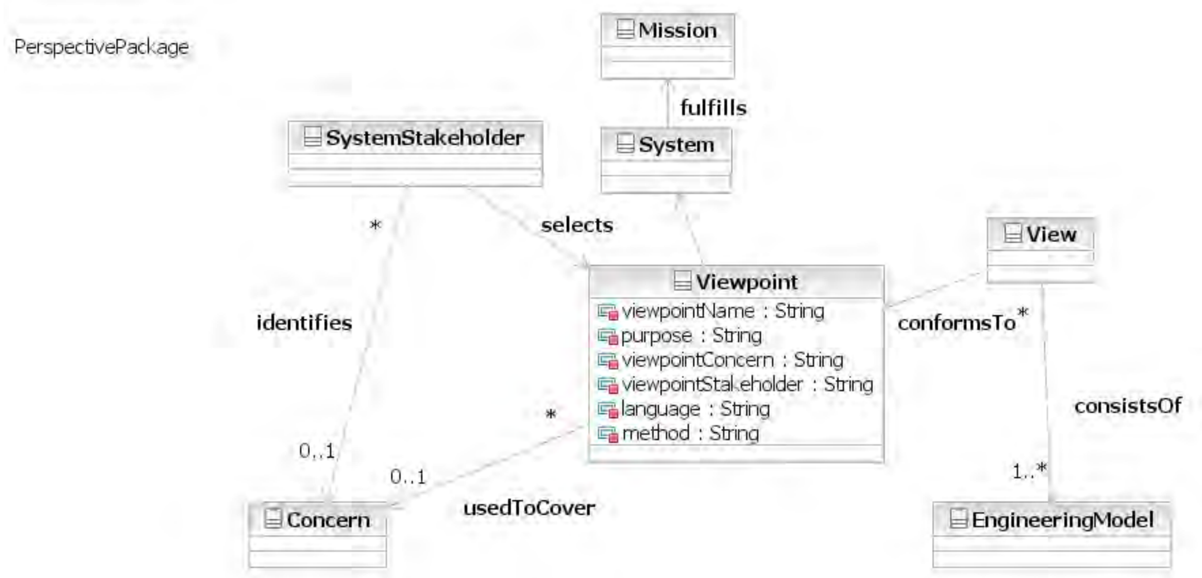


Figure 7.33: SystemEngineering Subontology: Perspective Package adapted from [Ins00]

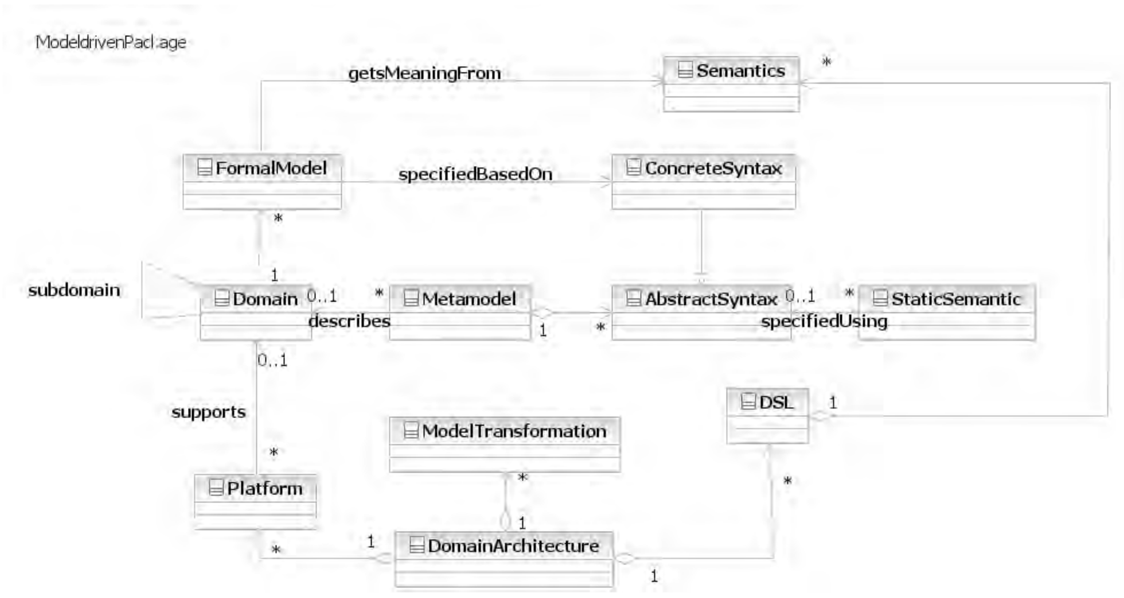


Figure 7.34: SystemEngineering Subontology: Modeldriven Package (adapted from [SV06])

ASysEngineeringSubontology PACKAGES

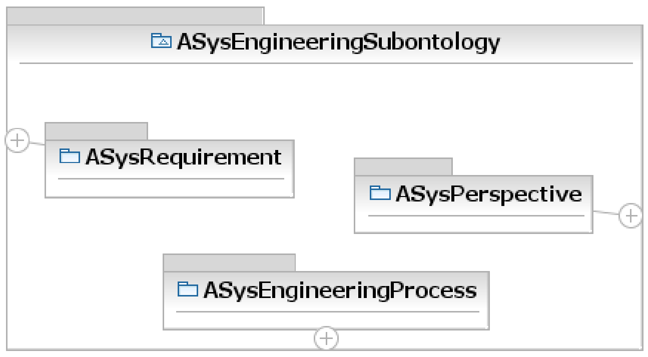


Figure 7.35: ASys Engineering Subontology Packages

- Purpose and specification: The ASys Requirement package gathers the concepts to define an autonomous system's requirements, either as new concepts or as specialisation of the System Requirement Package which defined requirements at a higher level of abstraction. Different documents describing generic autonomous system's characteristics, as well as ASys project research specific concepts have been used as sources for the package elements.

- Conceptualisation: To carry out the ASys engineering process, different characterisation have to be made in advance [SR08]: ProductCharacterisation, ProcessCharacterisation and SystemCharacterisation. The two latter comprise several concepts which have been further classified in two different packages (Fig. 7.40).

  Of special relevance for the SystemCharacterisation concepts are those related to the specific requirements of an autonomous system or Self–X properties [SLB+05], [SLB07] and [HSL08]: self-configuration, self-healing, self-optimisation and self-awareness among others.

- Formalisation: the former concepts have been formalised as UML class diagrams, for the Process Characterisation elements (Fig. 7.37) and for the System Characterisation concepts (Fig. 7.38). The definition of the ontological elements in the ASys Requirement package is given in Annex Sec. 11.2.2.1.

ASys Perspective Package

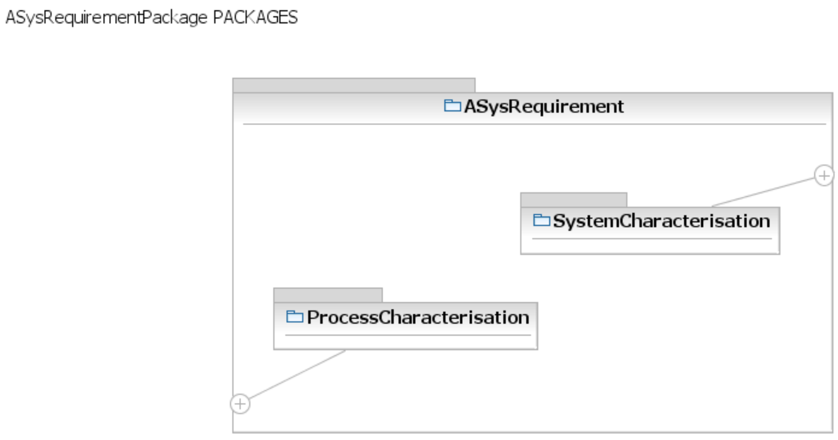


Figure 7.36: ASys Requirement Package subpackages

- Purpose and specification: this package provides the concepts and other elements to define the different perspectives usually considered in an autonomous system's characterisation. The different concepts and relationships considered in this package have been extracted from [RJB04], [MWM07], [vL08], [FMS08] merged and aligned as necessary to cater for the ASys research project ideas.
- Conceptualisation: Autonomous systems are usually too complex to be addressed as a whole, hence it is common to divide the system considering different viewpoints, where a viewpoint describes a perspective of interest that is used to specify a view of a system as consider in the System Perspective Package at an upper level of abstraction. For the ASys research, the following viewpoints have been considered: RequirementViewpoint (having ASysRequirement as concern), StructuralViewpoint (ASysStructure as concern), FunctionalViewpoint (ASysFunction as concern), BehaviouralViewpoint (ASysBehaviour as concern), and PerformanceViewpoint (ASysPerformance as concern). Different Views will conform to each one of the considered viewpoints: a static view describing the autonomous system's components in the form of different models, will conform to the StructuralViewpoint; a use case view describing actors, roles, and use cases will conform to the BehaviourViewpoint, etc. Due to the high variability of possible SystemStakeholders, Views and EngineeringModels for different autonomous systems' applications, these concepts have not been conceptualised as part of this package.

  RequirementViewpoint addresses the autonomous system's analysis and design which focuses on specifying the autonomous system's requirements. StructuralViewpoint considers a static description of the autonomous system, such as the subsystems and elements that an autonomous system consists of. The StructuralViewpoint addresses both the *as–is* (current description of the autonomous system) and the *to–be* (technical description of an autonomous system to be built). FunctionalViewpoint considers the autonomous system under the concern of its function,

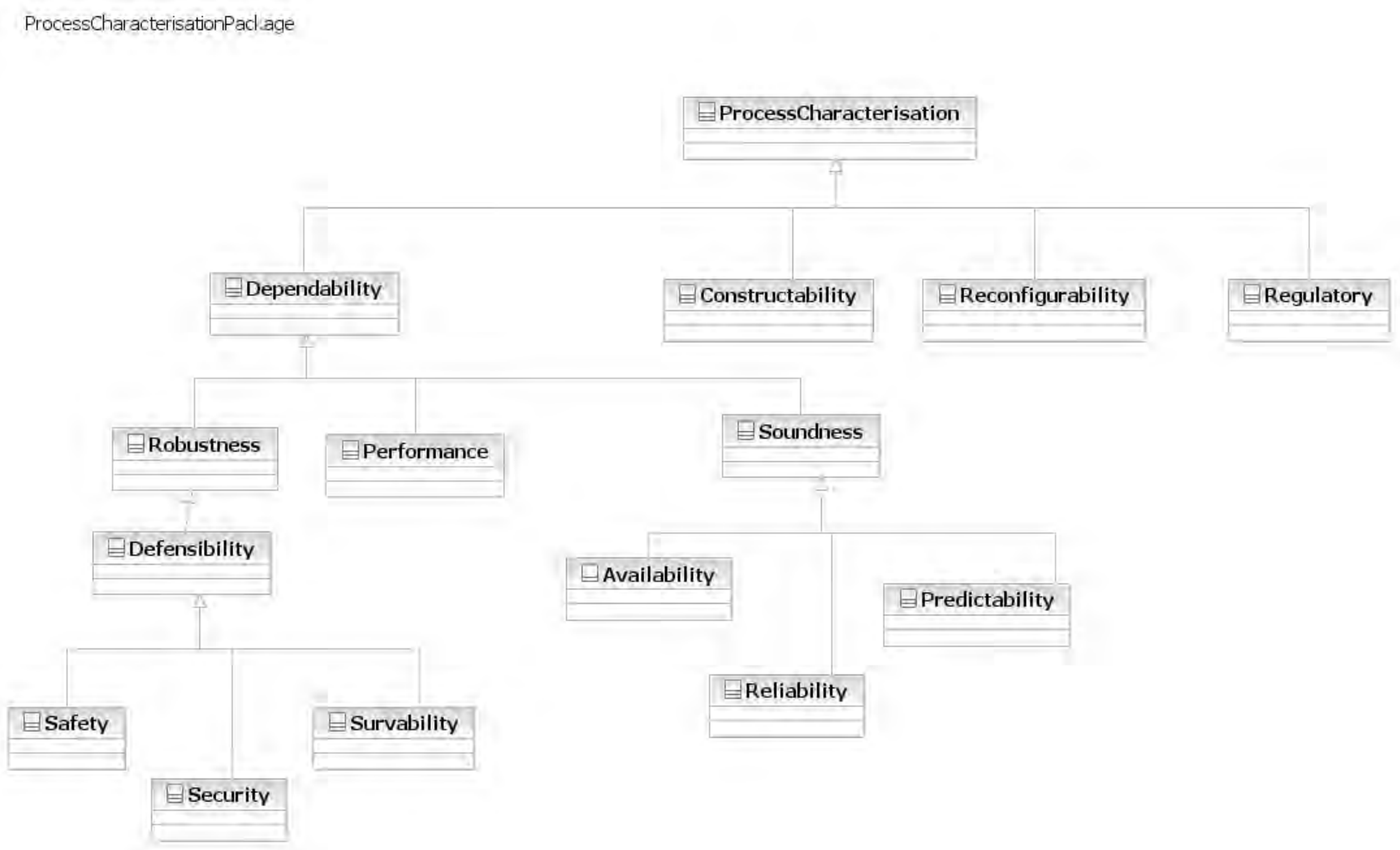


Figure 7.37: ASys Requirement Package: Process Characterisation concepts

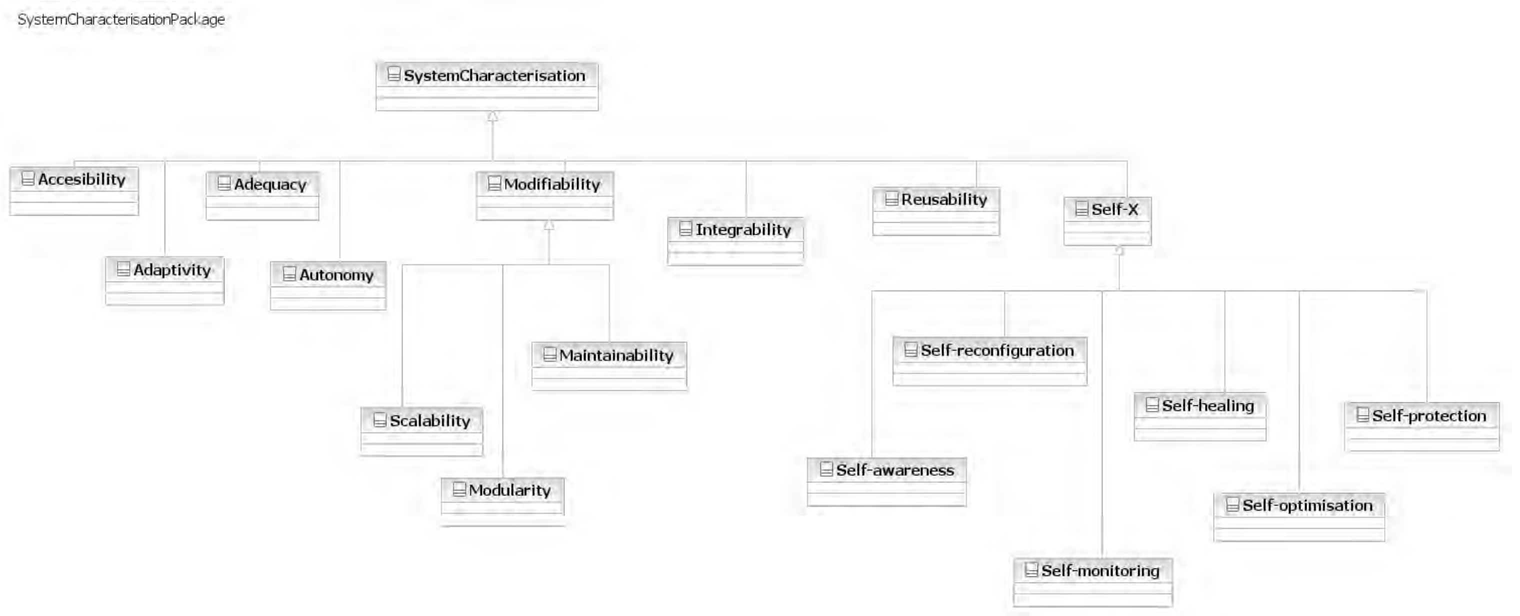


Figure 7.38: ASys Requirement Package: System Characterisation concepts

understood as a role played by a behavior specified in a context [Miz08], or in other words, the desired behaviour of the system [MWM07]. BehaviouralViewpoint regards the autonomous system from a dynamic or behavioural aspect (either observed in an as–is system or predicted, in a to–be system) where ASysBehaviour can be described as a series of changes to snapshots of the system [RJB04]. The PerformanceViewpoint is concerned with the evaluation of an autonomous systems under different performance measures [FMS08].

- Formalisation: the different concepts have been formalised in an UML diagram (Fig. 7.39), where the Viewpoint concept is imported from the Perspective Package. The different Concerns (ASysRequirement, ASysStructure, ASysFunction, ASysBehaviour, and ASysPerformance) have not been included in the diagram for the sake of simplicity. The definitions of all the concepts is provided in the Annex in Section 11.2.2.2.

ASys Engineering Process Package

- Purpose and specification: this package contains the ontological elements to specialise engineering process terms to the autonomous system's domain for an autonomous system's generic engineering process. Several sources have been used to define and organise this generic and re–usable engineering process of an autonomous system, taking into account the aspects of autonomy, integration, real–time features, and UML and SysML oriented software deployment which are required within the ASys research programme [SMP99], [Dou04], [Seg05], [SR08], [Est08].

- Conceptualisation:

  The different concepts defined in the Engineering Process Package can be specialised to describe the engineering process of an autonomous system. In this sense, the concept Phase can be conceptualised as: ASysAnalysis, ASysDesign, ASysImplementation, and ASysEvaluation. The ontological elements to conceptualise each one of the different phases have been organised into different packages (Figure 7.40).

  It has to be mentioned that although the aim of this package is to address the entire engineering process, the conceptualisation tasks have focused on those concepts belonging to the ASysAnalysis phase, as the OASys ontological elements are intended to be used in a methodology as analysis elements to describe the autonomous systems at this stage of the ASys research. Later developments within the ASys research project will allow to modify or to conceptualise in greater detail the additional phases suggested here as part of the generic engineering process.

  ASysRequirement focus on eliciting the autonomous system requirements from users and designers. ASysAnalysis denotes a phase which will obtain a high level description of the system from different Viewpoints defined in the ASys Perspective Package (requirements, structural, behavioural, functional, etc). ASysDesign refers

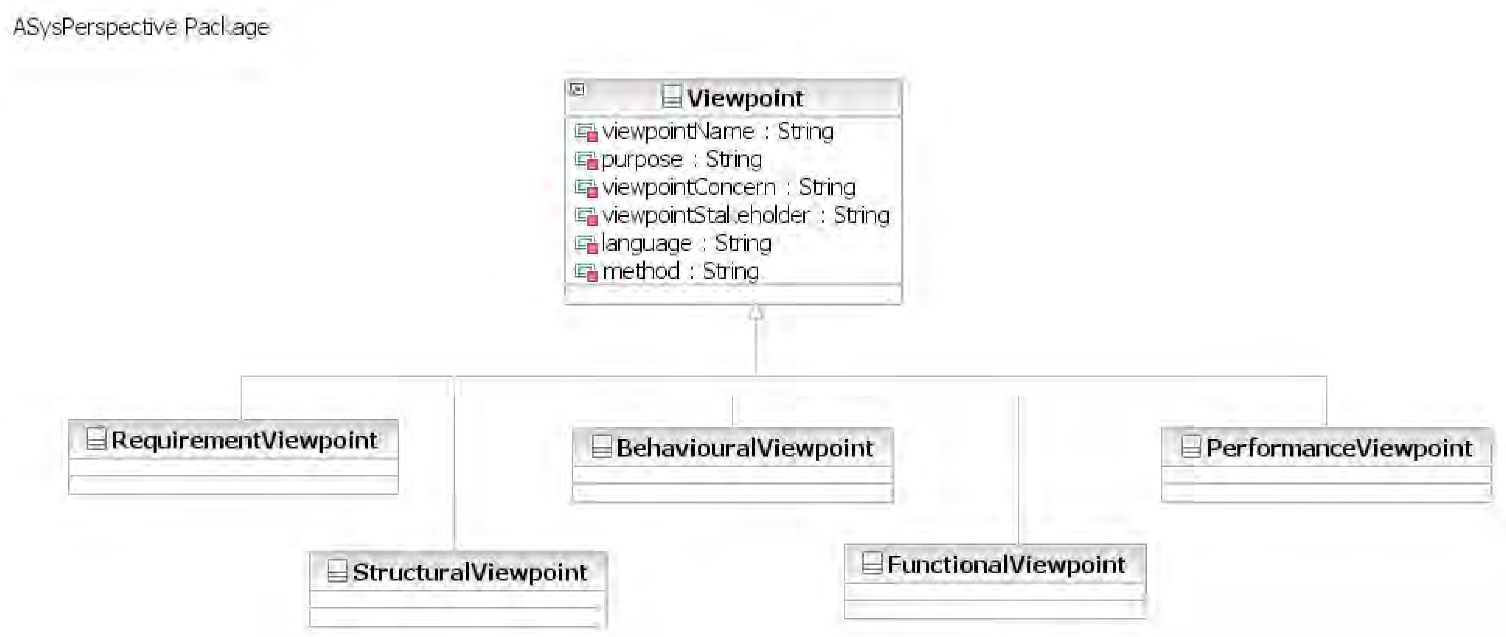


Figure 7.39: ASys Engineering Subontology: ASysPerspective Pacakge

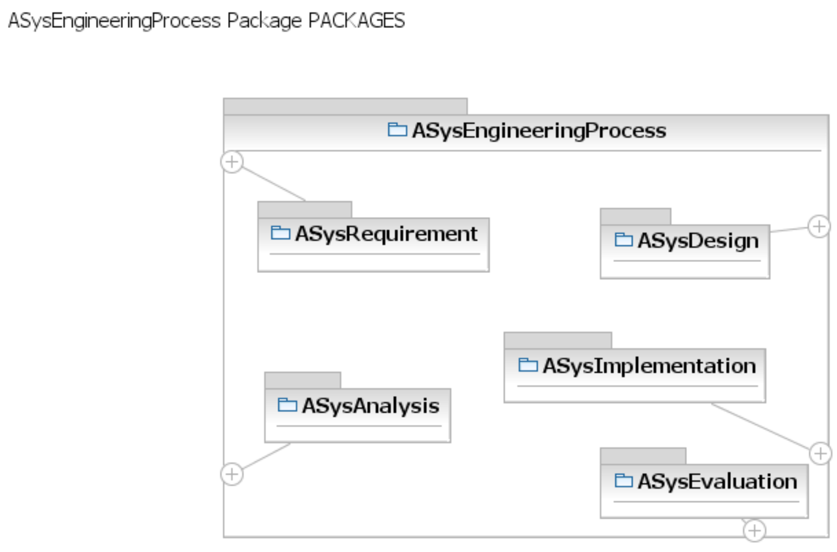


Figure 7.40: ASys Engineering Process Package subpackages

to a phase where a more detailed description will be obtained. ASysImplementation regards a phase which will obtain the description of a system from a physical or deployment consideration. ASysEvaluation refers to a phase where the assessment of the deployed system will be made.

The former phases in this layer inherit all the features and relationships of the concept Phase defined in the System Domain layer. Hence, each Phase might consist of different Tasks which are involved in its global achievement. The ASys-Requirement phasesconsists of the SystemUseCase task to address the definition and detail of the system and subsystems use cases, through the UseCaseModelling and UseCaseDetailing subtasks. Model kinds that could consist of different models and descriptions are obtained as workproducts, in the form of a UseCaseModel and a UseCaseSpecification. The RequirementCharacterisation task refers to the non–functional and functional requirement specification, in the form of Process-Characterisation and SystemCharacterisation workproducts.

The ASysAnalysis phase comprises the tasks of StructuralAnalysis, Behavioural-Analysis and FunctionalAnalysis to differentiate different kinds of analysis depending on the viewpoint selected. StructuralAnalysis refers to the analysis of the system in terms of its structure, topology (SystemModelling subtask) and knowledge (KnowledgeModelling subtask). As a result, a StructuralModel to define the structure and topology of the system, and a KnowledgeModel to represent the different types of knowledge in the autonomous system.

BehaviouralAnalysis is the task related to analysing the system in terms of its behaviour, obtaining a BehaviouralModel kind which consists of different BehaviourModels. In a similar way, the FunctionalAnalysis task refers to the functional modelling process in the autonomous system, obtaining a FunctionalModel kind comprising an Actor, Responsibility and Operation Models.

- Formalisation: the different concepts and relations of each subpackage conceptualised in this package have been formalised in an UML diagram: ASys Requirement (Fig. 7.41) and ASys Analysis (Fig. 7.42). Their definitions are given in the Annex Section 11.2.2.3.

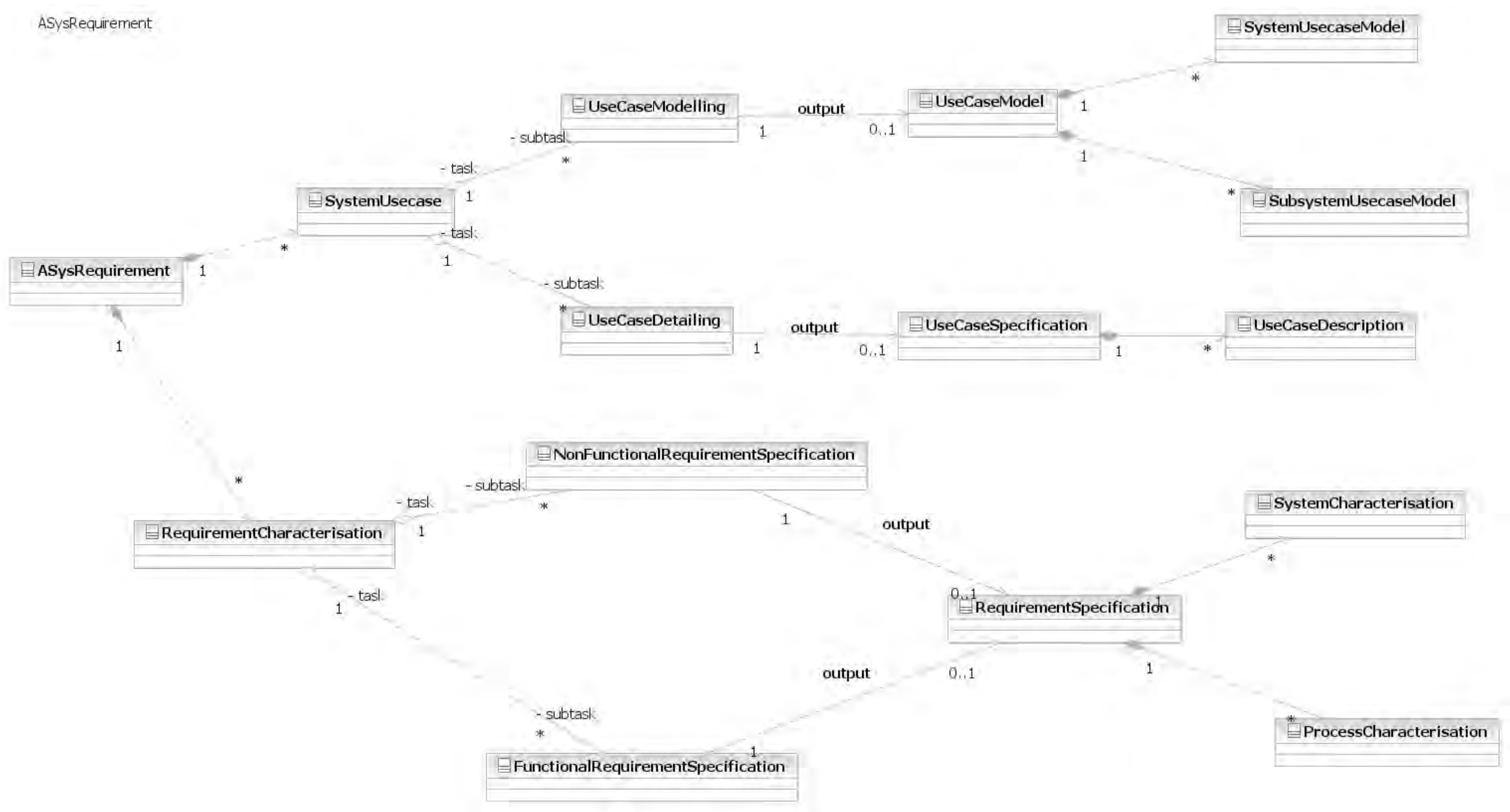


Figure 7.41: ASys Engineering Subontology: the ASys Requirement Phase in the ASysEngineeringProcess Package

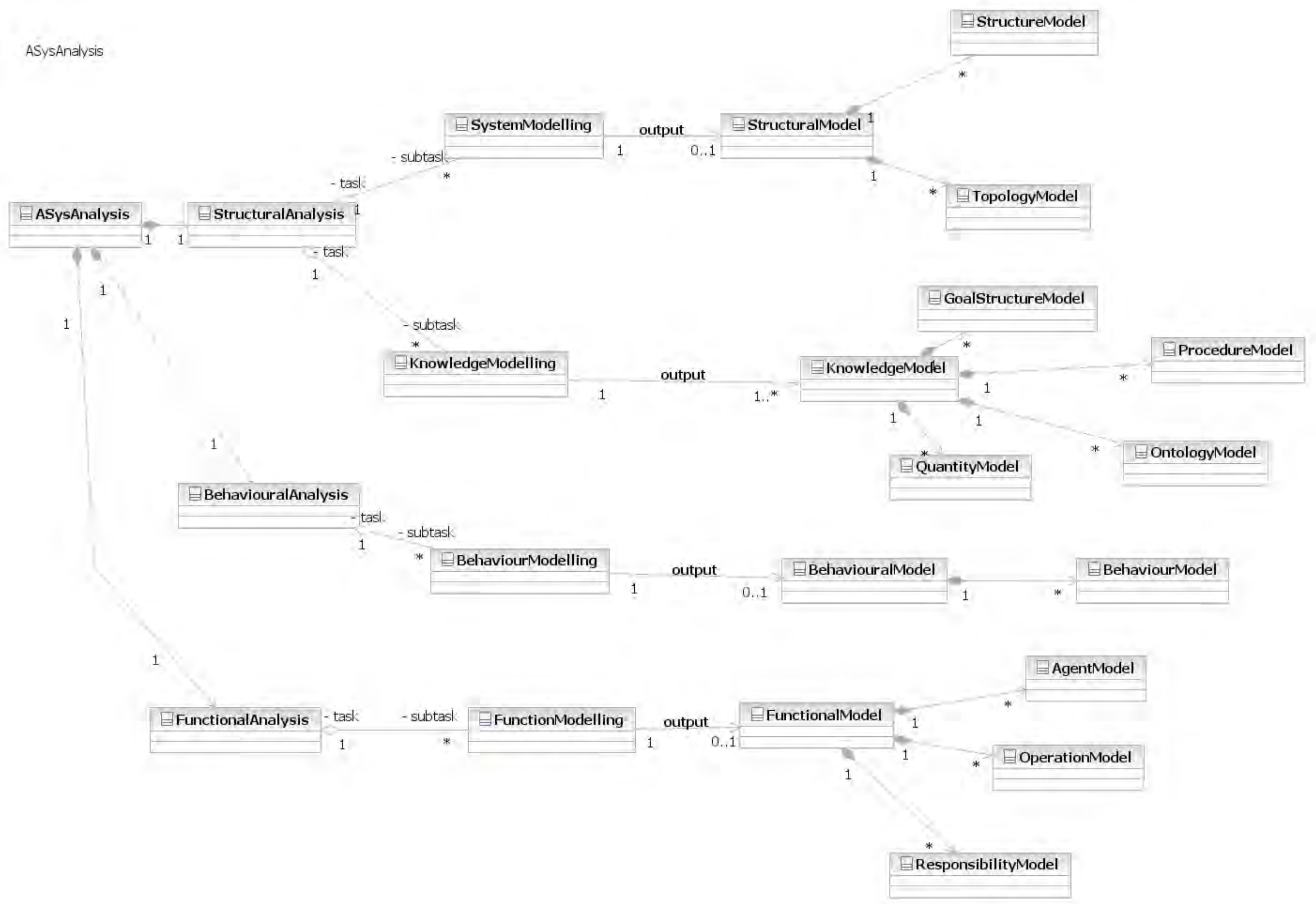


Figure 7.42: ASys Engineering Subontology: the ASys Analysis Phase in the ASysEngineeringProcess Package

Chapter 8

OASys–BASED ENGINEERING METHODOLOGY

8.1 Introduction

This chapter describes the proposal of an OASys–based Engineering Methodology to be used for the analysis and later development of autonomous systems. The different sections state the purpose, scope, description and related elements of the methodology.

8.2 Purpose and Scope

A methodology is understood for the scope of this research as a way of doing things in a particular discipline (as opposed to the meaning of analysis of methods used in a discipline) [HSGP08]. Hence, the OASys–based Engineering Methodology aims to provide support for ontology-based autonomous systems development based on OASys ontological elements.

The aim of the proposed OASys–based methodology is *to provide some guidelines and exemplify the application of the OASys ontological elements to an autonomous system generic engineering process*. Therefore, the purpose of the methodology is neither to provide a detailed autonomous system engineering process (which will be addressed at further stages of the ASys research programme [SR08]) nor to describe a fully developed ontology–based methodology for autonomous system development (which lies out of the scope of this research).

The methodological elements of the OASys–based Engineering Methodology are:

- To address an autonomous system generic engineering process that contain phases and tasks to carry out the autonomous system development.

- To suggest the techniques and guidances to assist the former tasks and subtasks.
- To describe the work products to be obtained as a class of models (this term refers to the fact that the methodology proposes a model kind, which might consist of different diagrams and specifications, not the model itself to be obtained in further development tasks).
- To specify the related OASys elements to be used in the work products.

8.3 Metamodelling: OASys Concepts Usage

Instantiation, by creating or identifying an entity that conforms to the definition [GPHS06], is the usual mechanism for metamodelling within the model–driven paradigm. However, this approach does not consider the ontological foundations involved in modelling and related languages. To overcome this situation, it is required to distinguish two distinct forms of metamodelling or types of instance-of relations [AK03]:

Linguistic metamodelling: relates user concepts to their domain types across metamodelling levels. Concepts are generally instantiated through linguistic instance–of relationships. An *instance-of* relation is used to express that the concept has been made from a specification concept [AZW06], as usually considered in the MOF based metamodelling [Obj06a]. This kind of instantiation is used in OASys to describe a specific ASys or application's objects (Fig. 8.1).

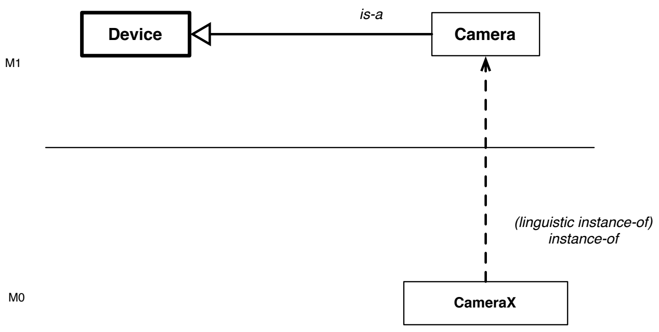


Figure 8.1: Linguistic instantiation of Camera concept

Ontological metamodelling: relates user concepts to their domain types within a level. Concepts are specialised through ontological instance–of relationships. An *is–a* relation is used to express its similarity to an ontological element of an higher level ontology [AZW06]. This approach of specialisation has been used for example in OASys to obtain the ontological elements of the ASysPerspective package as specialisation of those in the Perspective Package at a higher level of abstraction (Fig. 8.2).

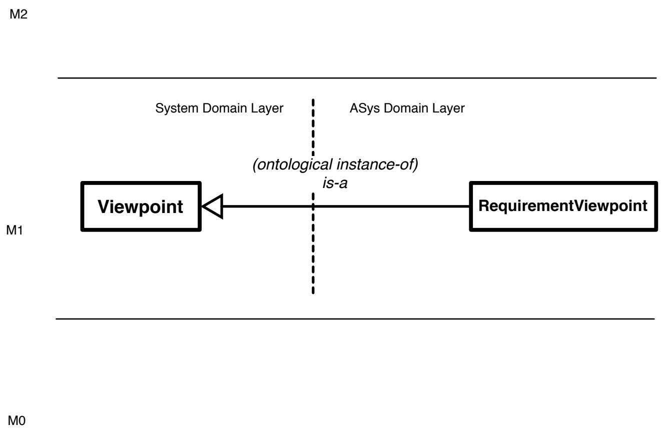


Figure 8.2: Ontological instantiation of Viewpoint concept

The existence of the two different metamodelling approaches has led to some metamodelling problems [HSGP08], when trying to satisfy both an ontological instance-of and the traditional linguistic instance-of metamodelling approach usually followed in the OMG Meta Object Facility (MOF) four layered metamodel architecture with (linguistic) metalevels [Obj06a]. The top level is the M3 layer that provides a MOF meta-meta model, used by MOF to build metamodels. The M2 layer provides metamodels to describe elements in the next layer, such as the UML metamodel, the model that describes the UML itself. The M1 layer contains models, such as those written in UML. The last layer is the M0 layer or data layer, used to describe real-world objects. Ontological instance-of (is-a) can take place within a layer, whereas linguistic instance-of (instance-of) is only allowed between layers. It might then happen that an alleged metamodel at level M1 is closer to an ontological model, i.e., a model that uses domain concepts than to an actual metamodel at level M2. Moreover, stereotypes can be used without having being defined at the corresponding M2 level.

To overcome this situation, and for metamodelling purposes in the OASys–based Engineering Methodology, OASys will play the role of an ontological model expressed with UML modelling constructs at the M1 level, inheriting hence its features from the UML Metamodel itself. Within the M1 level, the elements to consider in the different engineering analysis models proposed in the methodology will be obtained as specialisation through is–as or ontological instance–of relations (Fig. 8.3).

The relation between entities in the model receives the same name as the original relation in the reference package, known as construct overloading [Gui05]. However the specialised relation in the model expresses a different range and domain, and thus with different semantics from the original one.

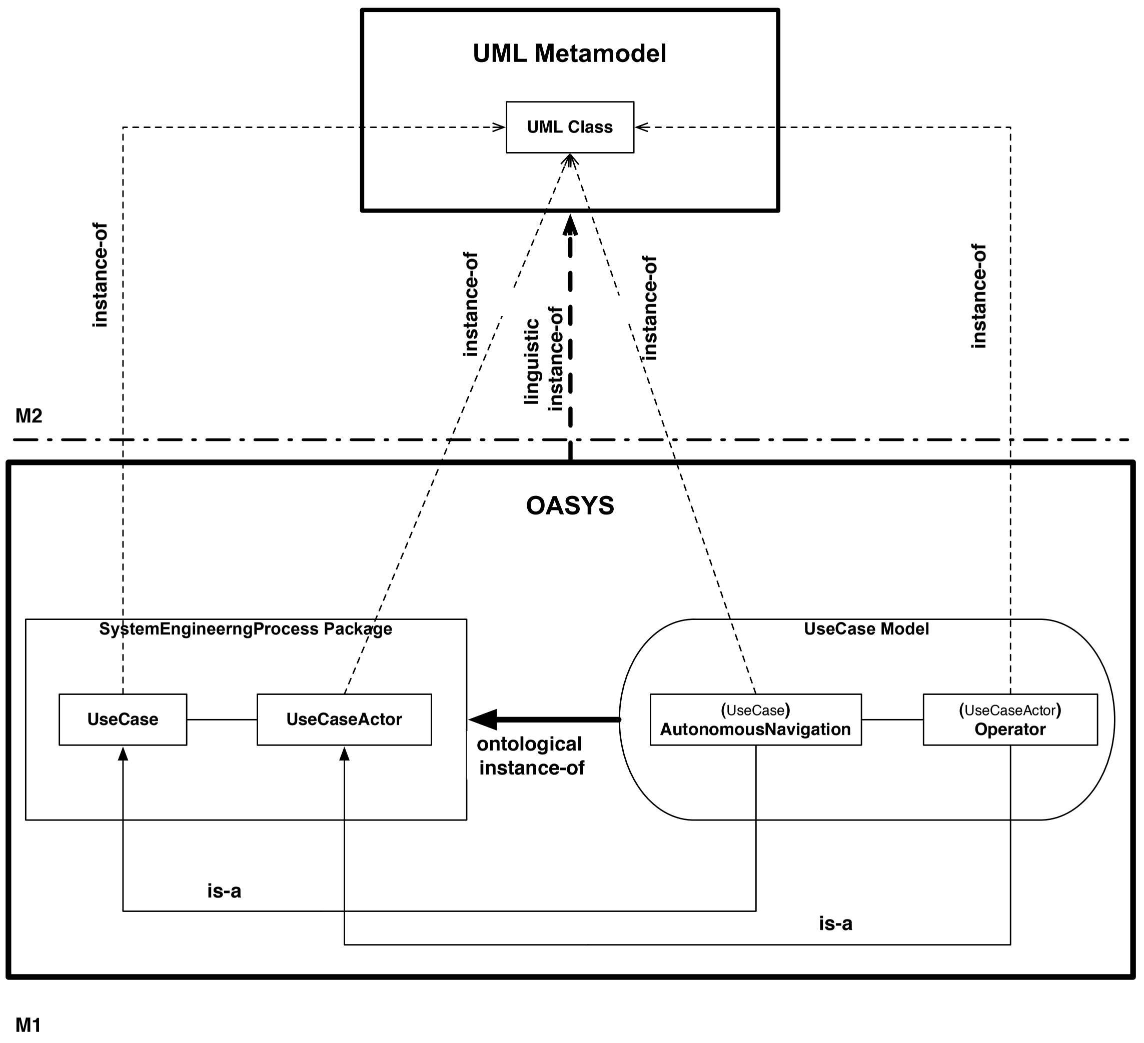


Figure 8.3: OASys as UML Model showing ontological concepts instantiation

8.4 Ontologies, Subontologies and Packages Interrelationships

The different ontologies, subontologies and packages in OASys show interrelationships among their ontological constructs. The relationship among OASys components appears both during the conceptualisation process and during its usage in the methodology. During the conceptualisation process of each ontology, subontology and package, the definition of an element can be based on the existence of already defined concepts. For example, the GoalLifecycle concept definition in the Thought Package is based on the concept Goal definition in the Knowledge Package. A GoalLifecycle concept cannot be defined without having a precise definition of the goal concept. For usage purposes, a model might need to relate both concepts through an additional relationship. For example the Thought Package imports some concepts such as Goal, Algorithm or GoalFunction already defined in the Knowledge Package as shown in Fig. 8.4.

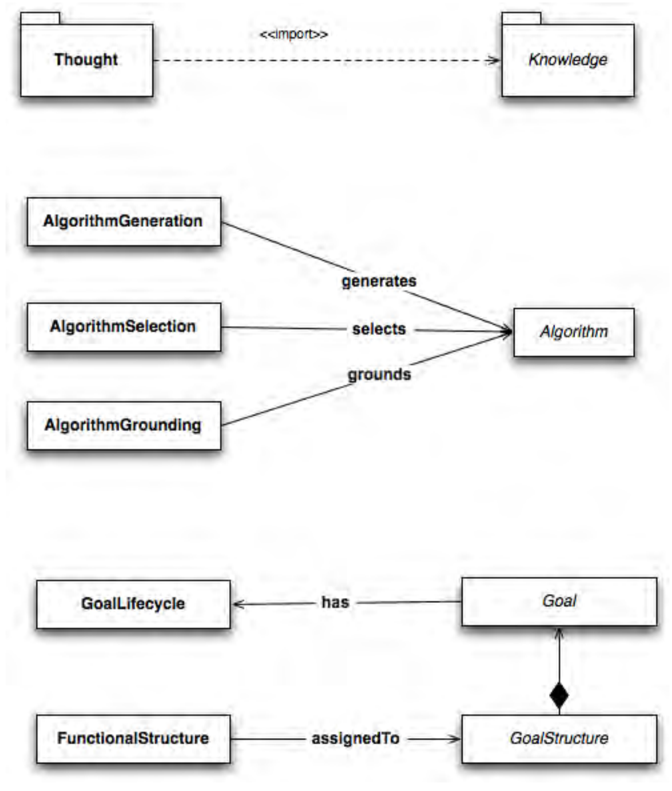


Figure 8.4: Thought package usage of Knowledge package

8.5 Methodology Description

The OASys–based Engineering Methodology is a proposal of a ontology-based autonomous systems generic development based on OASys ontological elements. A methodology can be defined as a composition or collection of related processes and methods [Est08]: The process defines what is to be done. The method or technique defines the how of each task or activity. Tools can be used to enhance what and how the tasks are to be made.

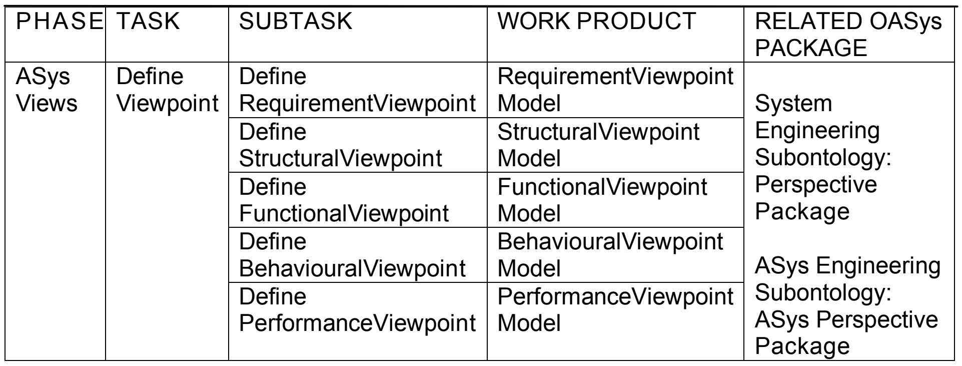
It has to be pointed out that the scope of the suggested methodology is limited to the Analysis phase of the generic autonomous system engineering process considered when conceptualising the ASysEngineeeringProcess package, as part of the ASys Engineering Subontology. Traditional phases of Design, Implementation or Deployment, and Evaluation are not considered at this stage of development of the proposed methodology and the ASys research programme. The reason to focus on the analysis phase of the engineering process lies on the use of ontologies as analysis models in software–oriented processes, since ontologies increase process reliability, offer a common vocabulary, and increase the reusability [AZW06].

The methodology focuses on the description on how to carry out the generic ASys engineering process, in terms of phases, tasks, work products, having as guideline the ontological elements in the System Engineering and ASys Engineering subontologies. Being OASys–based, the methodology considers the ontological elements required in the different tasks, by specifying the OASys packages to be used.

The methodology consists of two main phases, ASys Requirement to identify the autonomous system's requirements, and ASys Analysis to consider the autonomous system's analysis of its structure, behaviour and function. There might be an initial ASys Views phase, to identify those engineering views of interest in the autonomous system development.

ASys Views: this phase identifies the relevant views to consider for a particular autonomous system development, considering the Perspective Package elements, specialised through the ASysPerspective Package concepts (Figure 8.5). Views to consider during the characterisation of an autonomous system range from a RequirementViewpoint, StructuralViewpoint, BehaviouralViewpoint, FunctionalViewpoint and PerformanceViewpoint to consider respectively, the requirements, structure, behaviour, function, and performance concerns for an autonomous system. A ViewpointModel can be obtained for each of the perspectives considered, stating the different stakeholders, concerns, views and engineering models.

ASys Requirement: this phase identifies and elicits stakeholders' requirements for the system under analysis under the RequirementViewpoint. System and subsystems' requirements will be specified during this phase, to characterise the process and the system in terms of the Requirement and ASys Requirement concepts. Traditional requirements engineering techniques can be used to specify the concrete requirements.



PHASE	TASK	SUBTASK	WORK PRODUCT	RELATED OASys PACKAGE
ASys Views	Define Viewpoint	Define RequirementViewpoint	RequirementViewpoint Model	System Engineering Subontology: Perspective Package ASys Engineering Subontology: ASys Perspective Package
		Define StructuralViewpoint	StructuralViewpoint Model	
		Define FunctionalViewpoint	FunctionalViewpoint Model	
		Define BehaviouralViewpoint	BehaviouralViewpoint Model	
		Define PerformanceViewpoint	PerformanceViewpoint Model	

Figure 8.5: OASys–based Methodology: ASys Views phase

The different tasks, subtasks and workproducts concepts have been defined in the ASysEngineeringProcess Package, and they are detailed in Figure 8.6.

The *SystemUseCase* task obtains the system and subsystem's use cases models and specification as workproducts of two different subtasks: the UseCaseModelling and the UseCaseDetailing.

- The UseCaseModelling subtask (Fig. 8.7) obtains an UseCaseModel, consisting of the system and subsystem's use case models. It consists of different activities, such as to define the Subject, choose the UseCase under analysis, identify the UseCaseActors, and establish existing relationships having as conceptual support the concepts and relationships in the Requirement Package of the System Engineering Subontology. The UseCaseModel is an engineering model which could be later transformed into an UML or SysML Use Case Diagram.

- The UseCaseDetailing subtask (Fig. 8.8) produces an Use Case Specification. The Use Case Specification is a record of the use case, which details in a textual or tabular form the main data related to the use case, such as its description, flow of events, subject, actors, etc. The subtask consists of different activities to detail a specific UseCase.

The *Requirement Characterisation* task defines the autonomous system's requirements (Fig. 8.9), based on the Requirement and ASys Requirement Packages. The Requirement Package elements allow to classify the defined requirements as related to the physical, functional, performance or design aspects. The ASys Requirement Package ontological elements handle autonomous system's requirements in terms of process and system characterisation. Requirements can be described as a text that reflects a condition, a criterion and a threshold as proposed in [FCF$^+$09], customised depending on it being an availability, performance, reliability, safety,

PHAS E	TASK	SUBTASK	WORK PRODUCT		RELATED OASys PACKAGE
ASys Requirement	System UseCase	UseCase Modelling	UseCase Model	System UseCase Model Subsystem UseCase Model	System Engineering Subontology: Requirement Package ASys Engineering Subontology: ASys Requirement Package
		Use Case Detailing	UseCase Specification	UseCase Description	
	Requirement Characterisation	Non-functional Requirement Functional Requirement	Requirement Specification	Process Characterisation System Characterisation	

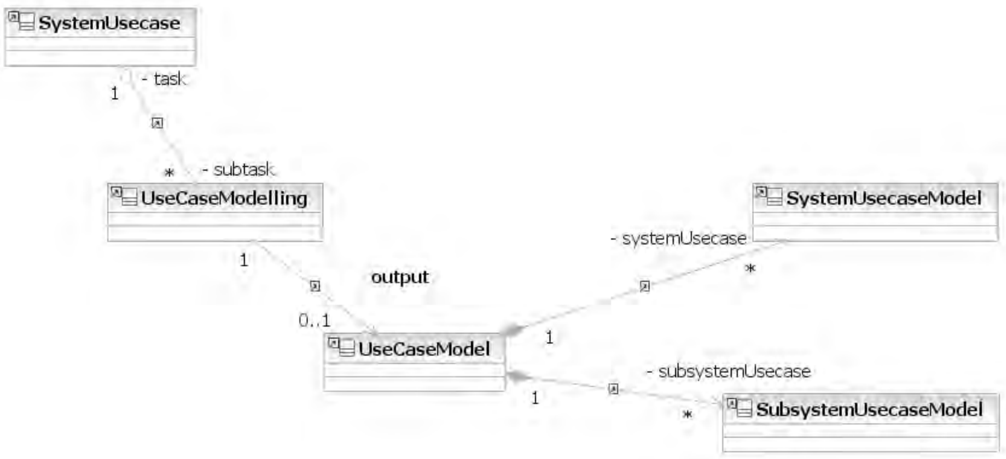
Figure 8.6: OASys–based Methodology: ASys Requirements phase

security, stability or another kind of requirement. Other requirements, such self–x concepts, might serve for common vocabulary purposes among the autonomous system's developers, to describe desired features without being possible to reify them. The OASys-based characterisations can be transformed or mapped into SysML Requirements Diagrams and Tables.

ASys Analysis: the purpose of this phase is to describe the ASys from different viewpoints, through different tasks such as Structural Analysis, Behavioural Analysis and Functional Analysis, as shown in Figure 8.10:

- *Structural Analysis* considers the system from a StructuralViewpoint, consisting of different modelling subtasks to analyse the system's subsystems and elements, the quantities, goals, algorithms, and ontologies already existing (*as-is* ASys) or to be considered (*to-be* ASys) (Fig. 8.11). The main work products obtained are the Structural Model and the Knowledge Model. Both models could be later transformed into UML Class Diagrams or SysML based diagrams.

  The System Modelling Subtask (Fig. 8.12) focuses on modelling the structural elements in the system and their relationships, obtaining a Structural Model. The Structural Model consists of a Structure Model which defines the mereological relations among the subsystems and elements in the autonomous system, by using and refining the more abstract levels concepts in the Gen-



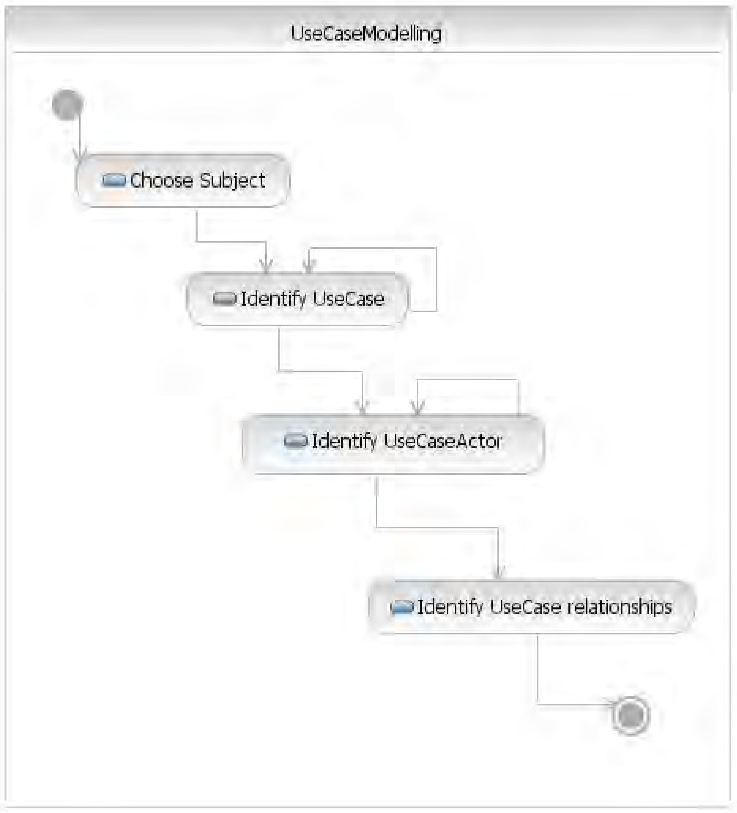
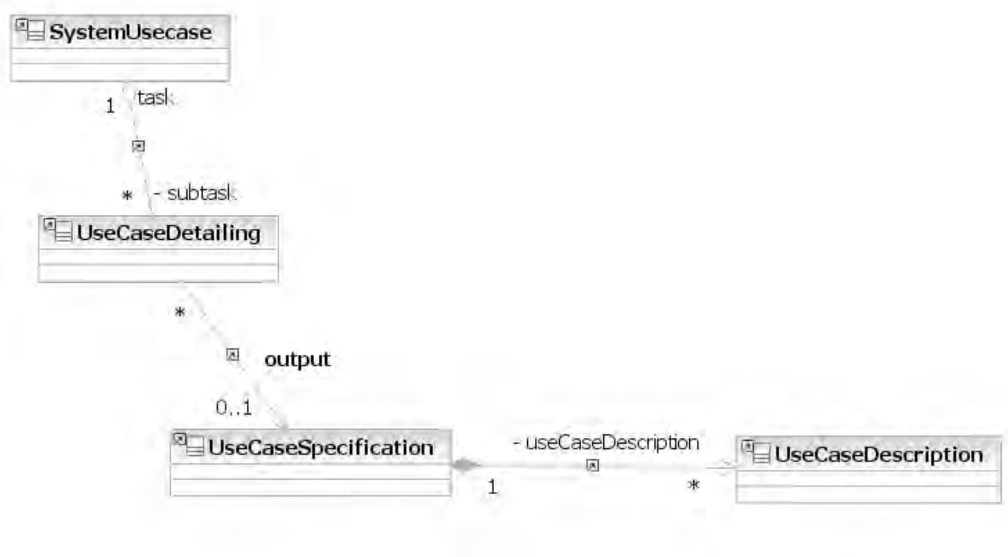


Figure 8.7: UseCaseModelling subtask



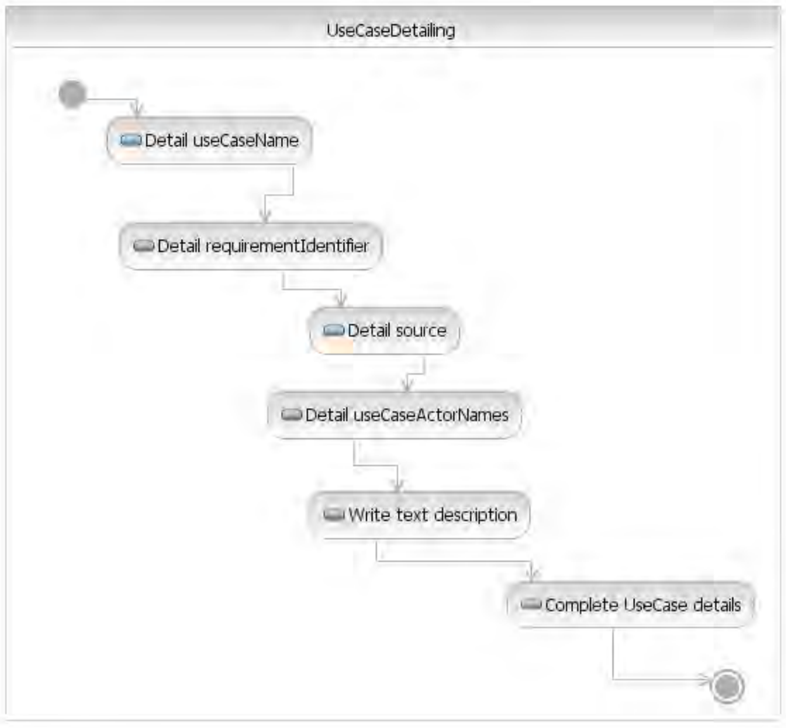


Figure 8.8: UseCaseDetailing subtask

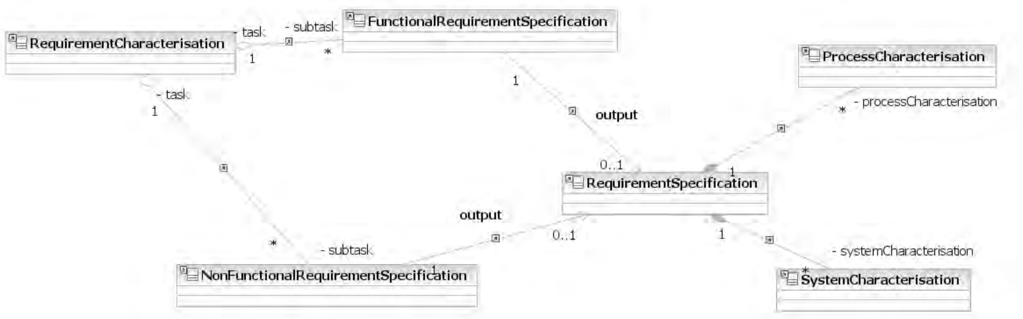
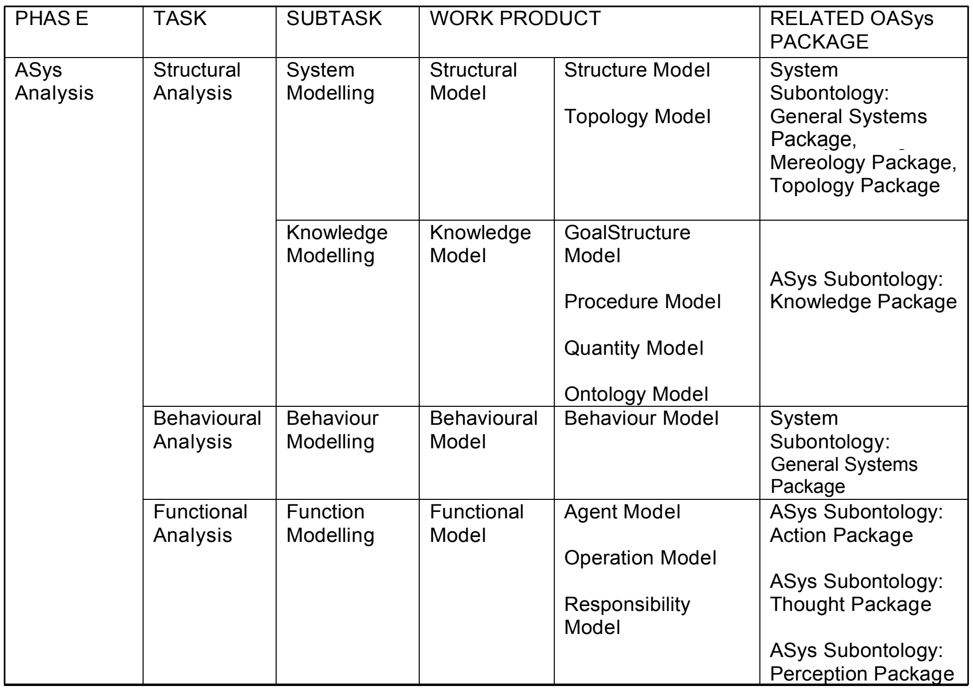


Figure 8.9: Requirement Characterisation task



PHAS E	TASK	SUBTASK	WORK PRODUCT		RELATED OASys PACKAGE
ASys Analysis	Structural Analysis	System Modelling	Structural Model	Structure Model Topology Model	System Subontology: General Systems Package, Mereology Package, Topology Package
		Knowledge Modelling	Knowledge Model	GoalStructure Model Procedure Model Quantity Model Ontology Model	ASys Subontology: Knowledge Package
	Behavioural Analysis	Behaviour Modelling	Behavioural Model	Behaviour Model	System Subontology: General Systems Package
	Functional Analysis	Function Modelling	Functional Model	Agent Model Operation Model Responsibility Model	ASys Subontology: Action Package ASys Subontology: Thought Package ASys Subontology: Perception Package

Figure 8.10: OASys–based Methodology: ASys Analysis phase

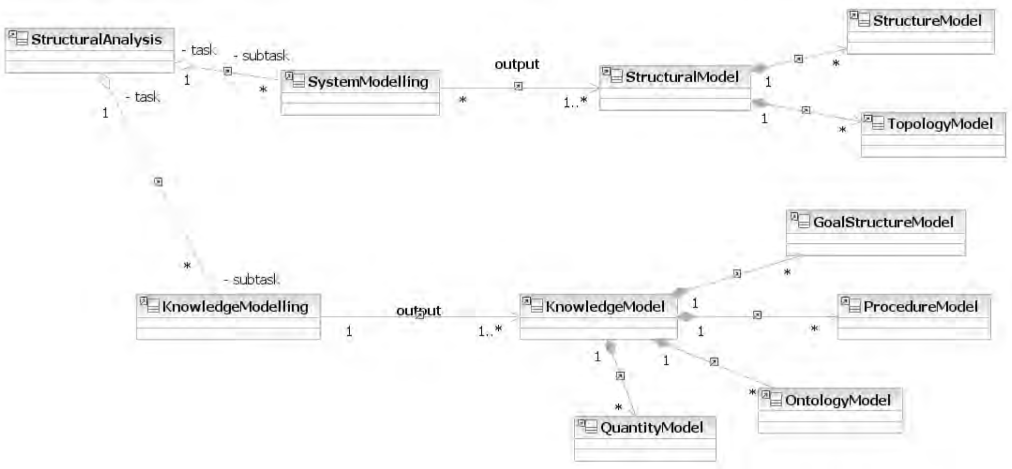


Figure 8.11: StructuralAnalysis task

eral Systems, and Mereology packages, related to system, subsystems, and mereological (part-whole) relations. The Structure Model can consist of one or different models, where the main system and its subsystems are modelled. For simple systems, one model describing the overall structure will suffice. For more structural complex systems, several models could be of help, by modelling each considered subsystem in its own Structure Model. The Structure Model can formalise both the hardware and the software structure in the autonomous system. The ontology–based Structural Models could be later transformed into UML Class Diagrams, or more specifically by means of SysML Block Definition Diagrams to show the structure of the system in subsystems and elements.

Once the system has been characterised into subsystems and elements, it is necessary to describe the topological relationships among them, using the concepts provided by the Topology Package. As part of the Structural Model, a Topology Model is obtained by defining the physical or information links among the components in the autonomous system. Topological relationships can be expressed later using a SysML Internal Block Diagram.

The properties and role of the different parts of the system can be characterised in a Device Model, where each one is further detailed using the Device package ontological elements. The Device Model formalises whether a particular element is a computational or a physical device. Later on, the specific function (sensor, actuator, etc) within the autonomous system will be detailed in the design phase.

The Knowledge Modelling Subtask focuses on obtaining a range of Knowledge Models, which considers the different knowledge types for an autonomous sys-

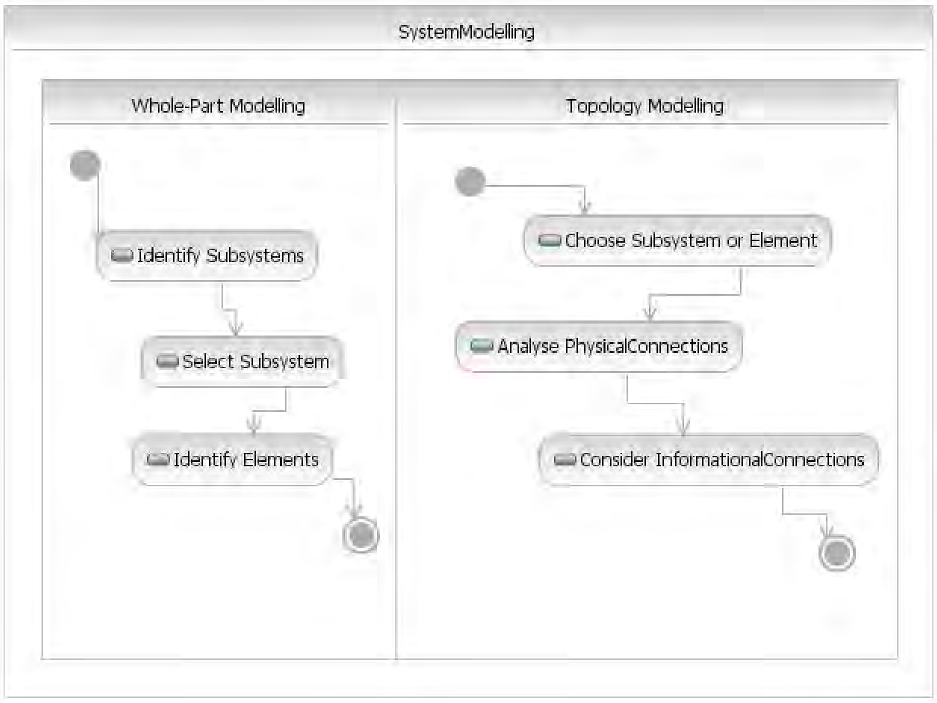


Figure 8.12: System Modelling Subtask

tem, specialising the Knowledge Package concepts into different models (Fig. 8.13). It can consist, depending on the application, of different activities to obtain the models. As part of the Knowledge Modelling, it is specially important the analysis of how the autonomous system goals captures its mission, which is expressed as a general or higher level goal which is in turn and decomposed into intermediate and lower level goals in the form of structure or hierarchy, represented as a GoalStructure Model. Additional models can be obtained during this subtask: the Quantity Model is obtained by identifying the quantities or variables that characterise the autonomous system; the Procedure Model is obtained by analysing possible Algorithms to address the GoalFunctions of the Goals previously identified; the Ontology Model purpose is to consider the domain for the specific ASys, to cater for ontological elements describing the concept definitions and domain properties of the specific application under analysis. The different Knowledge Models can be later be detailed using UML or SysML diagrams.

- *Behavioural Analysis* is a Task (Fig. 8.14) which analyses the system from a Behavioural Viewpoint, consisting of a Behaviour Modelling subtask. The task obtains as workproduct a Behavioural Model, consisting of one or more Behaviour Models, which could be formalised based on UML and SysML Activity, Sequence and State Machine diagrams.

  Behaviour Modelling captures current behaviours in an *as–is* system, or desired ones in a *to–be* system, which will be modelled in a Behaviour Model which uses the General Systems Package concepts related to system's behaviour.

- *Functional Analysis* analyses the system from a FunctionalViewpoint (Fig. 8.15), obtaining a Functional Model which consist of the Agent Model, the Operation Model and the Responsibility Model, which can be formalised as UML and SysML Activity, Collaboration, Sequence and State Machine diagrams. The Functional Model specialises the concepts from the Thought, Perception and Action packages.

  The Actor Model is obtained by identifying the different Actors in the autonomous system, using the corresponding part in the Action Package. The Operation Model captures the different Operations performed by the autonomous system, as specialisation of the Operation taxonomy in the Action Package. As part of the Operations carried out by the autonomous system: the Perception ones as defined in the Perception Package; the ConceptualOperations which include the GoalLifeCyclePhases and GoalReconfigurationPhases to be specialised from the Thougth Package; and the GroundedOperation referrring to the FunctionalDecompositionPhases in the Thought Package. The Responsibility Model captures the distribution of responsibilities among the Actors, defining their capabilities.

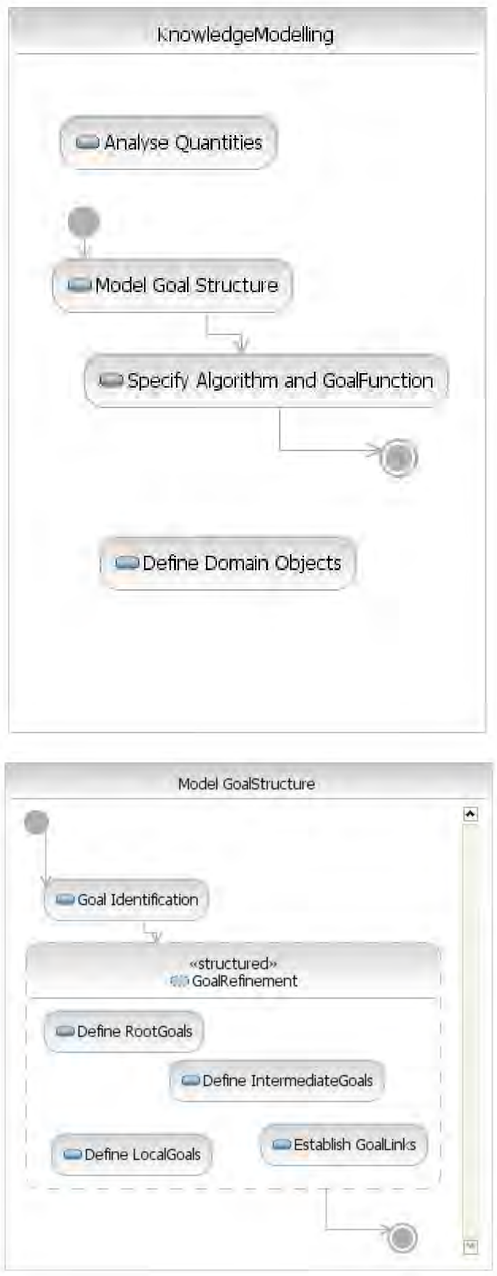


Figure 8.13: Knowledge Modelling Subtask

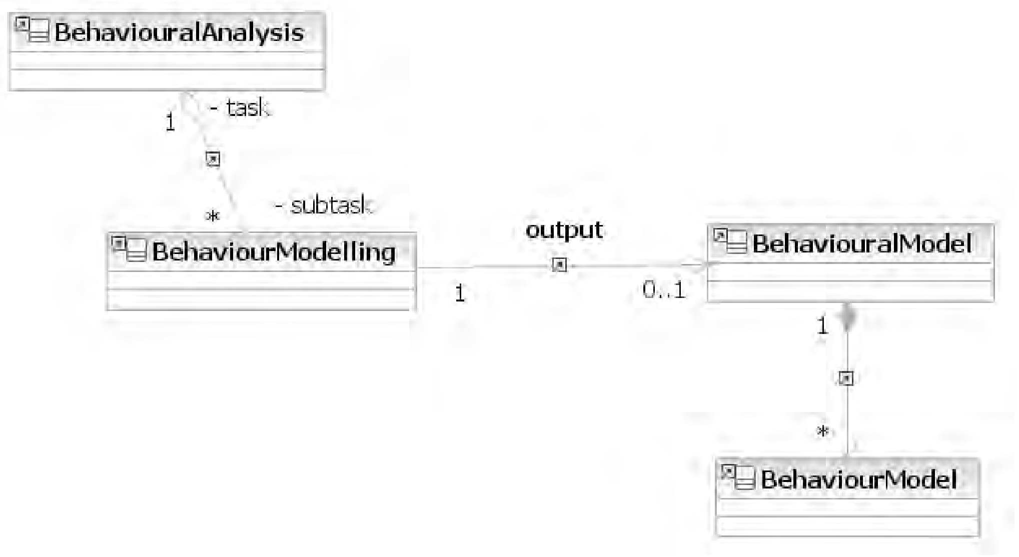


Figure 8.14: BehaviouralAnalysis task

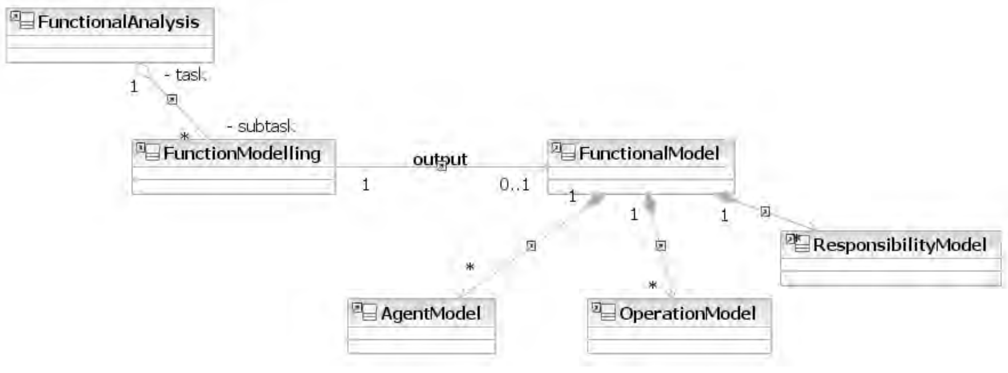


Figure 8.15: FunctionalAnalysis task

Chapter 9

APPLICATION

9.1 Introduction

This chapter shows how the ontology OASys has been used to characterise two testbeds considered within the research project ASys: the Robot Control Testbed and the Process Control Testbed. These exemplary systems have been chosen to cover a different range of applications, which share the commonality of existing or foreseen autonomy. Later, the ontology and the ontology–based engineering methodology have been applied to them.

9.2 Robot Control Testbed

The Robot Control Testbed (RCT) is a robotic–based application under development in the research group ASLab. The research aim is to provide a mobile robot with the necessary cognitive capabilities and an intelligent control system, as to perform complex tasks. The current aplication's software and hardware description, specification, and use cases and additional information can be found respectively in [CH08], [HH08], and [San07]. The following sections summarise the key aspects of it.

9.2.1 RCT Structural Description

The current robotic–based application consists of different interrelated systems to provide the desired capabilities. Three different subsystems, composed of several elements, can be considered: the base platform, the onboard system, and the supporting system.

1. Base Platform
   The base platform of the RCT is a mobile robot Pioneer 2–AT8 which has been designed by ActivMedia Robotics (Fig. 9.1). It is a robust platform which includes all the necessary elements to implement a control and navigating system, specially designed for outdoor applications. Additional systems and elements can be attached to this platform. The ActivMedia Pioneer 2–AT8 is used to build a

complete, complex mobile robotic system named Higgs.

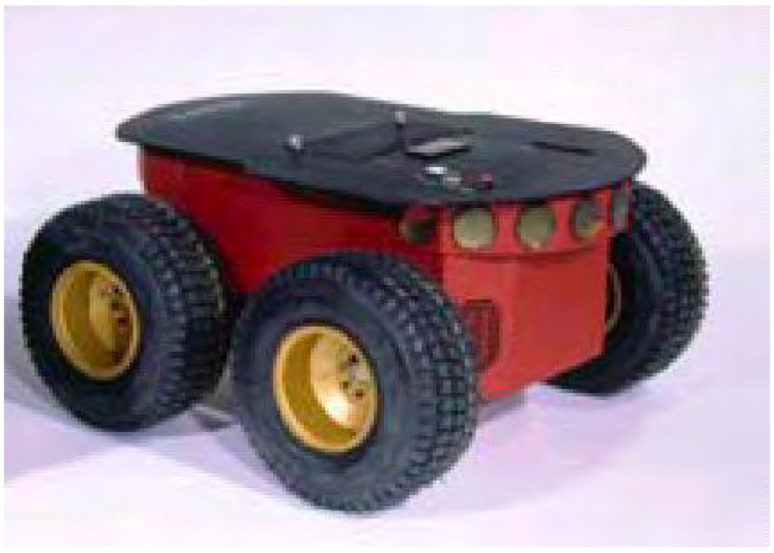


Figure 9.1: Pioneer 2-AT8 (Higgs base platform)

It is a small sized mobile robot, with a support structure made of aluminium. Its total weight is of 15 kg, being capable of carrying up to 40 kg. From a hardware viewpoint, the mobile robot consists of different elements:

Robot Panel: it is the superior platform of the robot, designed for a later assembly of new elements such as cameras or laser systems.

Robot Body: it is a box–shaped element made of aluminium. It contains the batteries, the actuators, the electronic circuits and the rest of elements. It also allows to attach additional elements such as an onboard PC, a modem or additional sensors.

Control Panel: it is the access panel to the robot's microcontroller, placed in the robot panel. It consists of several control buttons, robot status leds (robot switch on, microcontroller status, battery charge) and a serial port RS–232 to be used as an input and output communication link with a external PC.

Sensors: the mobile robot is provided with two arrays of eight ultrasonic sensors each, which allow the detection and location of objects in the mobile robot environment. The arrays are placed at the front and at the rear part of the robot.

Actuators: the robot contains 4 Pittman motors GM9236E204. Each one includes an optical encoder to determine the robot's speed and position.

Microcontroller: it is a Hitachi H8S microcontroller board consisting of different elements (memories, serial ports, inputs, outputs, 8 bit bus) which carries out different operations such as trigger and register the sensors' signal, control the actuators, and some other low–level operations.

Bumpers: they are additional elements attached to the platform, 5 at the front and 5 at the rear.

Power: there are three batteries 12 VDC 7 A-h, located at the rear part of the robot. They provide 252 W-h, which assure several hours of autonomy movement to the robot. Their status can be checked in the corresponding led of the control panel.

In addition to the hardware, the mobile robot has different software elements, either provided by the manufacturer or developed by ASLab team members.

AROS (ActivMedia Robotics Operating System): it is the operating system, consisting in server processes running on the Hitachi microcontroller in Higgs. It is a low–level software in charge of regulating the motors' speed, sonars' signal, encoders' signal and other low–level tasks. This software will also communicate the obtained information to other client software applications through the RS-232 serial interface.

ARIA: it is an applications–programming interface (API) based on C++ to control the robot. It acts as the client in the client–server topology. It allows to program high–level software applications, such as intelligent behaviour (obstacle avoidance, object recognition, wandering, exploration, etc). The robot control is based on direct commands, movement commands or abstract–level actions.

ASLab team software: there are a set of modules developed by the ASLab team members to extend or to add capabilities of Higgs. The modules developed are: communication [Par04], synthetic emotions [Con05], SOAR integration [Sán05], voice [Pan05], RT–CORBA control server [Her05], RT–CORBA robot status register, Java based CORBA client, CORBA based remote operation [Con07], and surface recognition [God07].

2. Onboard Systems

On the base platform, different devices have been attached to expand the original range of functionalities of the mobile robot (Fig. 9.2).

- Onboard Computer: it is a computer attached to the base platform whose mission is to facilitate the communication with the microcontroller, the control of the robot, the execution of complex navigation operations, and additional communication operations. Taking into consideration several requirements and operational constraints defined in [CH08], a GENE–6330 was chosen. Related to software, the onboard computer uses Linux as operating system, as well as having a real–time software RTAI and RT–CORBA server.
- Laser: it is a laser scanner for mobile robotics applications, placed at the front part of the robot panel.
- Wrist: it is a 2 DOF high precision and torque actuator for orientation of the camera.

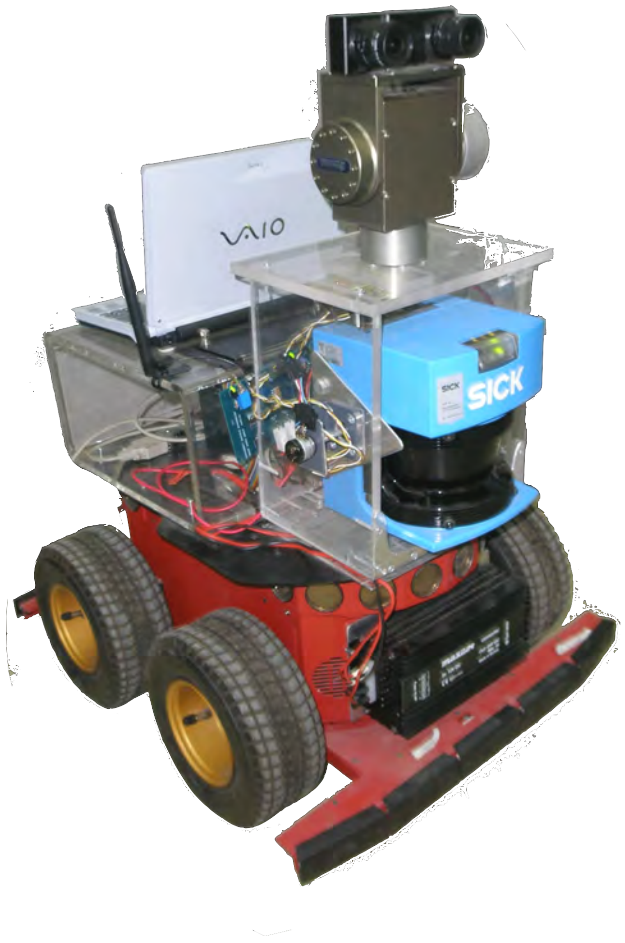


Figure 9.2: Pioneer 2-AT8 with onboard systems (Higgs)

- Camera: it is a stereo camera that will be used as the robot's vision system, attached to the wrist.
- Radio: it is a radio system for DGPS.
- Laptop: it is a lightweight highly functional laptop.
- GPS: it is a high performance GPS system.

3. Supporting Systems

   Along with the base platform and the onboard systems, some additional supporting systems are included to complete its functioning.

   (a) Wireless Network: it is an additional subsystem included in the testbed to allow the communication with the wireless local network in the ASLab laboratory. It consists of a wireless card (placed in the onboard computer), an antenna necessary as the wireless card is placed inside the aluminium robot body (attached to the robot panel), and an access point (connected to the laboratory LAN).
   (b) Remote Control: it is a portable device to remote control the robot.
   (c) Server: it is an additional computer to run additional software.
   (d) User Terminal: it is a client computer to control the robot remotely.

9.2.2 RCT Functional Description

The functional description of the Robot Control Testbed has been made through the specification of use cases and requirements. *FunctionalRequirement* has been defined in the ontology as a requirement that specifies an operation or behaviour a system must perform.

For the RCT, the functional requirements considered can be classified into primary and secondary ones. A primary functional requirement refers to a main capability to be fulfilled by the system. In the case of the RCT, two primary functional requirements have been considered: navigation, and survival.

A secondary functional requirement refers to an additional capability desired in the system. For the RCT, these secondary functional requirements are: to explore the environment, to avoid an obstacle while moving or exploring, reactive movement, the identification of an object in the environment, and search. Some additional functional requirements expand or detail the former ones, such as autonomous navigation as an extended case of the navigation one.

9.3 OASys–based Engineering Methodology Applied to RCT

The OASys–based Engineering Methodology described in Sec. 8.5 has been applied to the Robot Control Testbed for its conceptual analysis. The following sections provide a description on how the different tasks have been accomplished, with examples of the different workproducts obtained. The terms in italic that appear in the phases, tasks, subtasks and models descriptions are those defined in the Packages of OASys.

9.3.1 RCT ASys Views Phase

The purpose of this *Phase* is to identify the different *View*s of interest for the RCT considering the Perspective Package elements, specialised through the ASysPerspective Package concepts.

For example the *RequirementViewpoint* specialises the concept of *Viewpoint* to address the analysis of the RCT from a *ASysRequirement Concern*, to establish the different requirements to be defined in the system. Considering this viewpoint, a RCTRequirementViewpointModel can be instantiated (Fig. 9.3). The different concepts, relationships and attributes where chosen from the Perspective and the ASysPerspective Package. The original concepts have been ontologically instantiated into RCT related ones. For example, the concept RCTRequirement is an *ASysRequirement* concern, which in turn is a*Concern* from the Perspective Package. In a similar way, RCTPerformanceViewpoint is a *Viewpoint*, as defined in the Perspective Package.

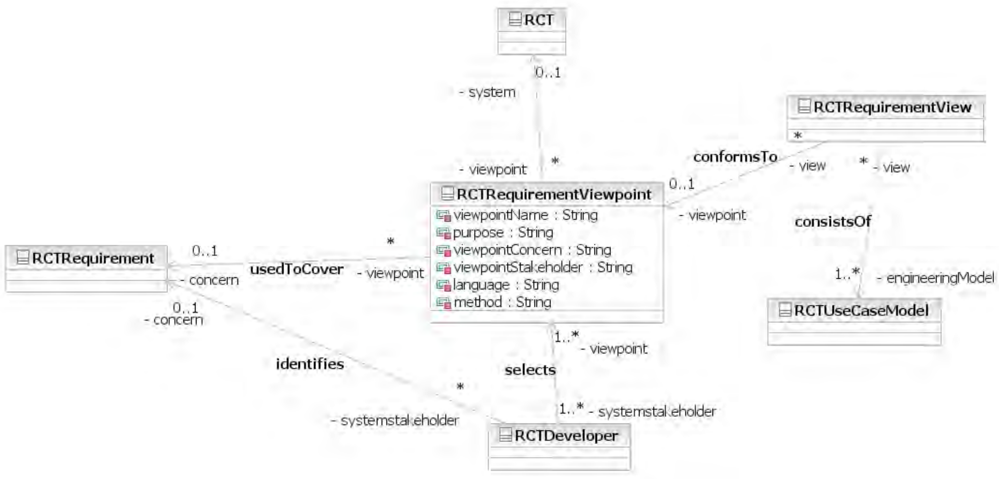


Figure 9.3: RCT RequirementViewpoint Model

The *StructuralViewpoint* pays attention to the autonomous system structure, as considering the *ASysStructure Concern.* A RCTStructuralViewpointModel can be obtained (Fig. 9.3) by instantiating the different concepts, relationships and attributes in the Perspective and the ASysPerspective Packages. In this model, the RCTStructuralView is a *View* that will consist of a RCTStructuralModel, which is the *EngineeringModel* to be obtained as output of the StructuralAnalysis task.

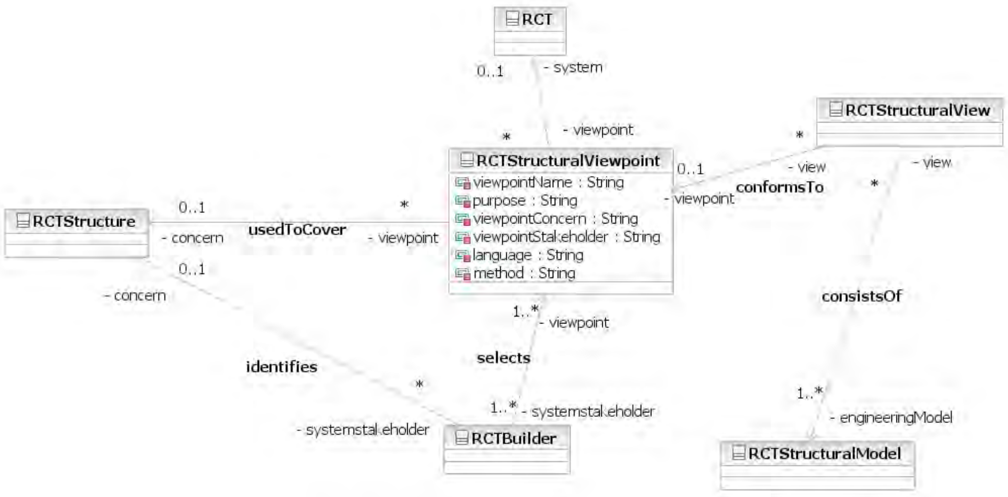


Figure 9.4: RCT StructuralViewpoint Model

Additionally, the *BehaviouralViewpoint* serves to describe the autonomous system's behaviours, i.e., how the system operates under certain conditions. The RCTBehaviouralViewpointModel obtained (Fig. 9.5) instantiates the concepts and relationships in the Perspective and the ASysPerspective Packages. The different roles played for the concepts in the model are shown in each one of the relationships, which are given the same name as in the original package.

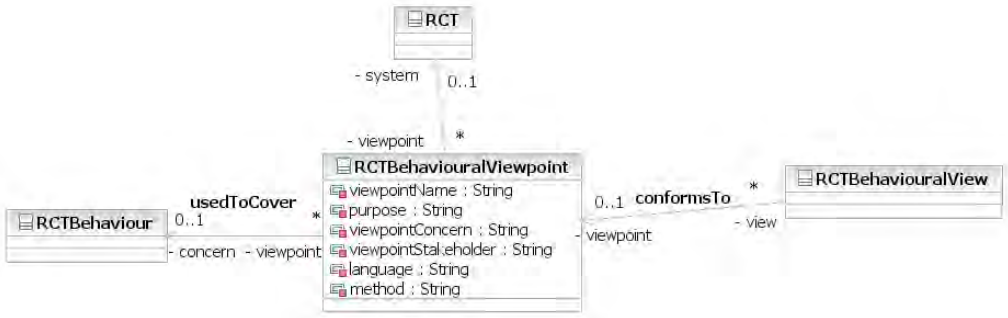


Figure 9.5: RCT BehaviouralViewpoint Model

In turn, the *FunctionalViewpoint* focus on the analysis of an autonomous system's functions, as desired or expected behaviours. The RCTStructuralViewpointModel (Fig. 9.6) instantiates the Perspective and the ASysPerspective Packages ontological elements, focusing on the functional aspects to consider for the RCT. In this model, the RCT-FunctionalView is a *View* formalised as a RCTFunctionalModel, which is the *Engi-*

neeringModel consisting of different function related models, obtained as outputs of the FunctionalModelling subtask.

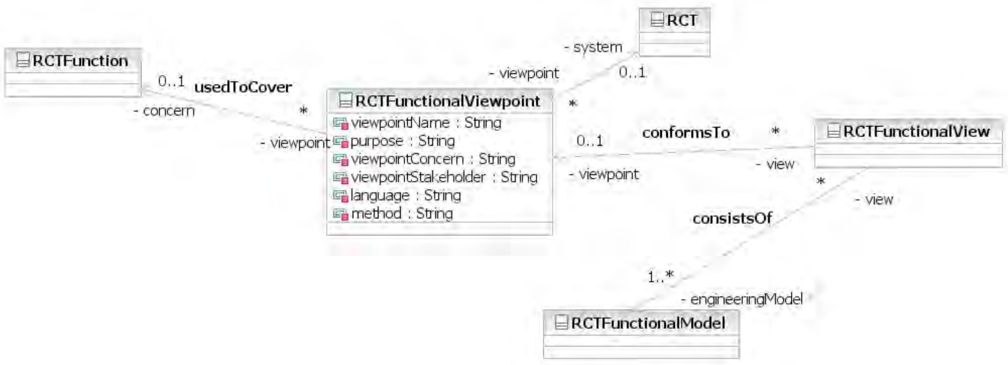


Figure 9.6: RCT FunctionalViewpoint Model

During the RCT development, it will be necessary to take the *PerformanceViewpoint*, to evaluate the performance requirements and benchmarking of the RCT once it is finally implemented. The PerformanceViewpoint addresses the performance aspects of the RCT. The Perspective and the ASysPerspective Package ontological elements were used to build up a PerformanceViewpointModel for the RCT (Fig. 9.7).

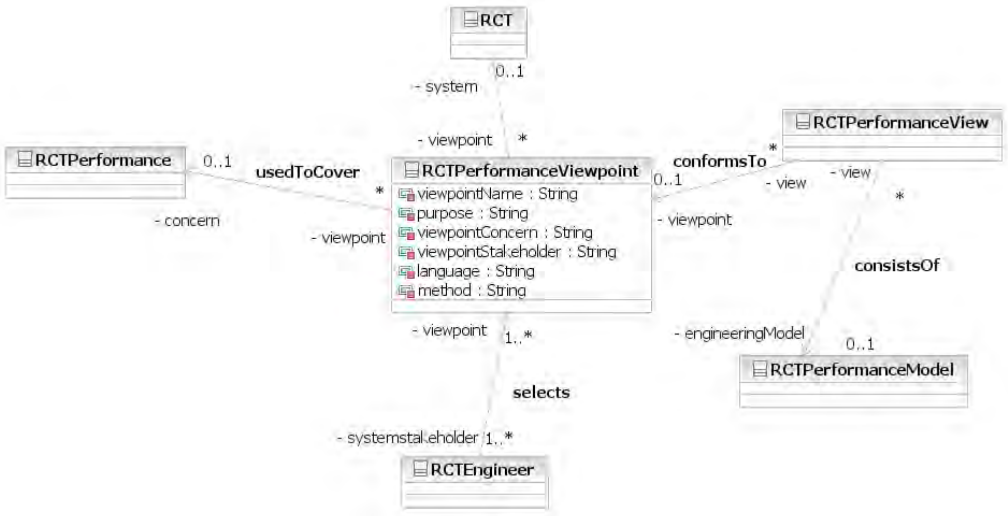


Figure 9.7: RCT PerformanceViewpoint Model

9.3.2 RCT ASys Requirement Phase

The purpose of this phase is to identify and elicit stakeholders' requirements for the RCT considering the RequirementViewpoint. The requirements are to be specified during this phase, by using traditional requirements engineering techniques.

- System UseCase Task

  The first *Task* during the *ASys Requirement Phase* is to analyse the RCT*System UseCase*s, by instantiating the Requirement Package' ontological elements in the SystemEngineering Subontology for the Robot Control Testbed (Fig. 9.8). This *Task* consists of the *Subtask*s of RCT *UseCaseModelling* and RCT *UseCaseDetailing*, as specialised from the ASys Engineering Subontology.

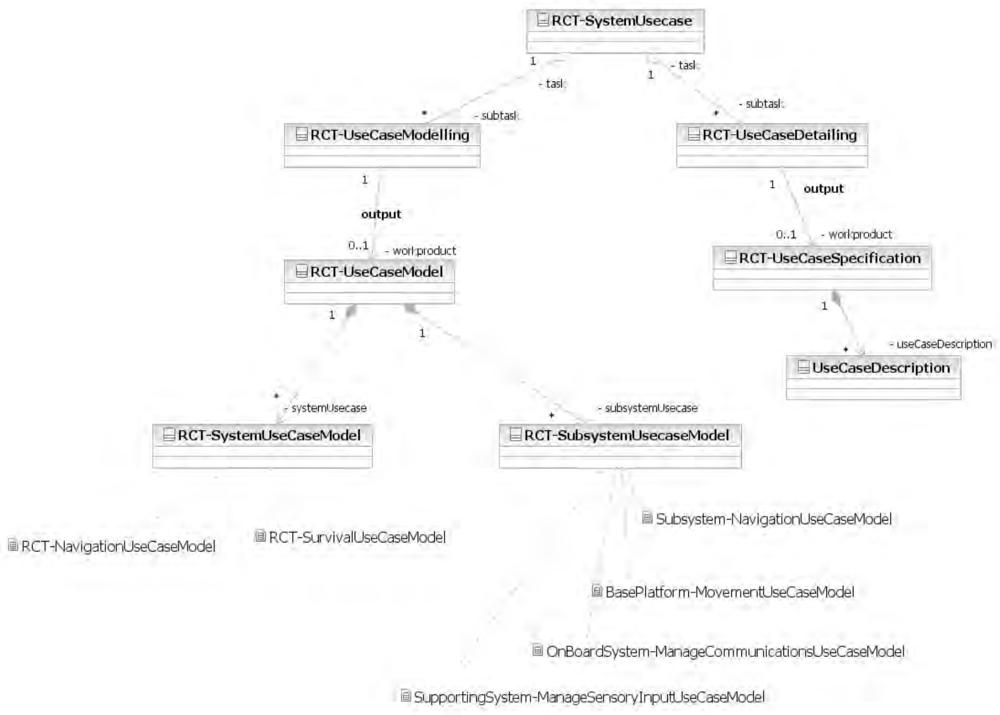


Figure 9.8: RCT UseCase Task

An *UseCase* in OASys has been defined as a mean to capture a requirement of a system, as defined in the Requirement Package. To define the UseCase, the *Subject* as system under consideration, and the different *UseCaseActor*s as objects that interact with the system are also identified, among other aspects. As a result, a *SystemUseCaseModel* is obtained, detailing the previous identified elements. The UML classes in the model are instantiation of the original OASys concepts, this fact being specified by the UML roles names in the shown associations.

When the *Subject* under study is the Robot Control Testbed, an RCT *UseCase-Model* shows the RCT's requirements by means of use cases. A system's requirements can be of different types (physical, functional, performance, interface, design) as defined in the Requirement Package. An initial requirement analysis made by the RCT developers, identified the *FunctionalRequirement*s for the RCT, where this concept has been defined as a "requirement that specifies an operation or behaviour that a system must perform" in OASys. Primary *FunctionalRequirement*s for the RCT are to navigate, as well as to survive.

The navigation requirement is captured by means of the *UseCase* Navigation, which *include*s the secondary *FunctionalRequirement*s of being able to explore the environment, identify elements in the environment, and to avoid obstacles. These requirements are captured in the *Usecase*s of EnvironmentExploration, Identification and ObstacleAvoidance respectively (Fig. 9.9). In turn, the *FunctionalRequirement* of surviving is captured in the Survival *UseCase*, which *include*s the Subsystem-Failure and Recharge *UseCase*s (Fig. 9.10).

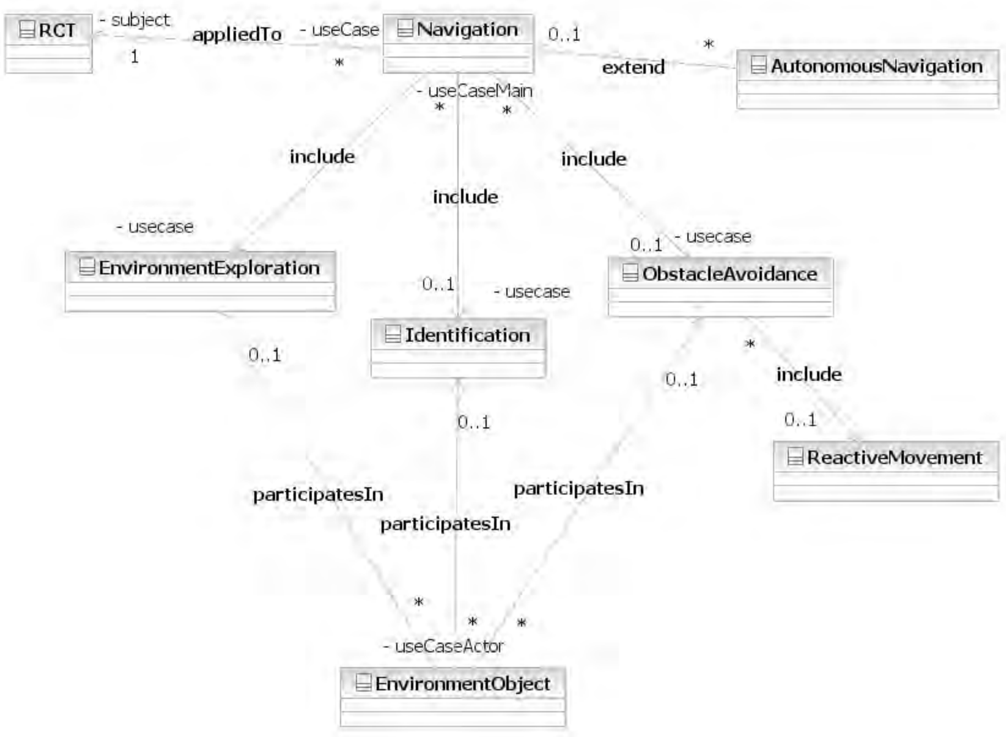


Figure 9.9: RCT Navigation UseCaseModel

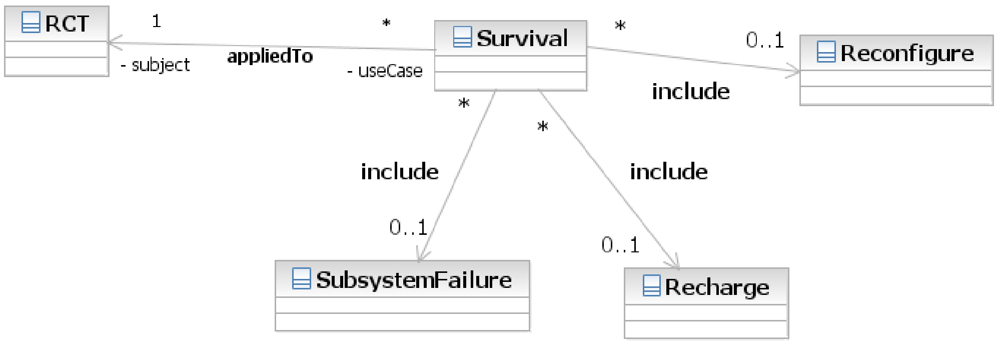


Figure 9.10: RCT Survival UseCaseModel

It might be interesting to detail a particular *UseCase* by paying attention to the *Subsystems* in the *System*, to detail further *Requirements*. *Subsystems* identified in the RCT are the BasePlatform, the OnboardSystem, and the SupportingSystem. Focusing, for example, on the Navigation *UseCase* previously defined, it is possible to identify additional *Requirements* for the *Subsystems*, in the form of a RCT Subsystem *UseCaseModel* (Fig. 9.11).

The *UseCase Modelling Subtask* can also obtain a *Subsystem UseCaseModel*, which gathers the *Requirement* for a particular *Subsystem*. In the case of the RCT, this subsystem usecasemodel was obtained for the BasePlatform (Fig. 9.12 showing the instantiation of the original ontological elements) to detail the *FunctionalRequirement* of movement; for the OnboardSystem to specify the *FunctionalRequirement* of ManageCommunications (Fig. 9.13); and for the SupportingSystem to clarify the Manage SensoryInput requirement (Fig. 9.14).

The next *Subtask* in the *ASysRequirement Phase* is to detail each *UseCaseModel* obtained during a *UseCase Detailing Subtask*. This detail can be achieved by filling a pre–defined ASys UseCase Pattern, with the relevant information in each field as required. This *Subtask* was performed by the RCT developers, having as results different textual tables for the previous *UseCases*. For example, the detail for the Navigation requirement is shown in Fig. 9.15, and the Survival one in Fig. 9.16.

- Requirement Characterisation

  An additional *Task* in this *Phase* is to characterise the autonomous system's requirements, to obtain a *RequirementCharacterisation*, as defined in the ASys Engineering Process Package of the ASys Engineering Subontology. The taxonomies of concepts considered in the ASys Requirement Package can be described specifying a condition, a requirement criterion and a possible threshold. As a result, a *Process Characterisation* and a *System Characterisation* in the form of a textual

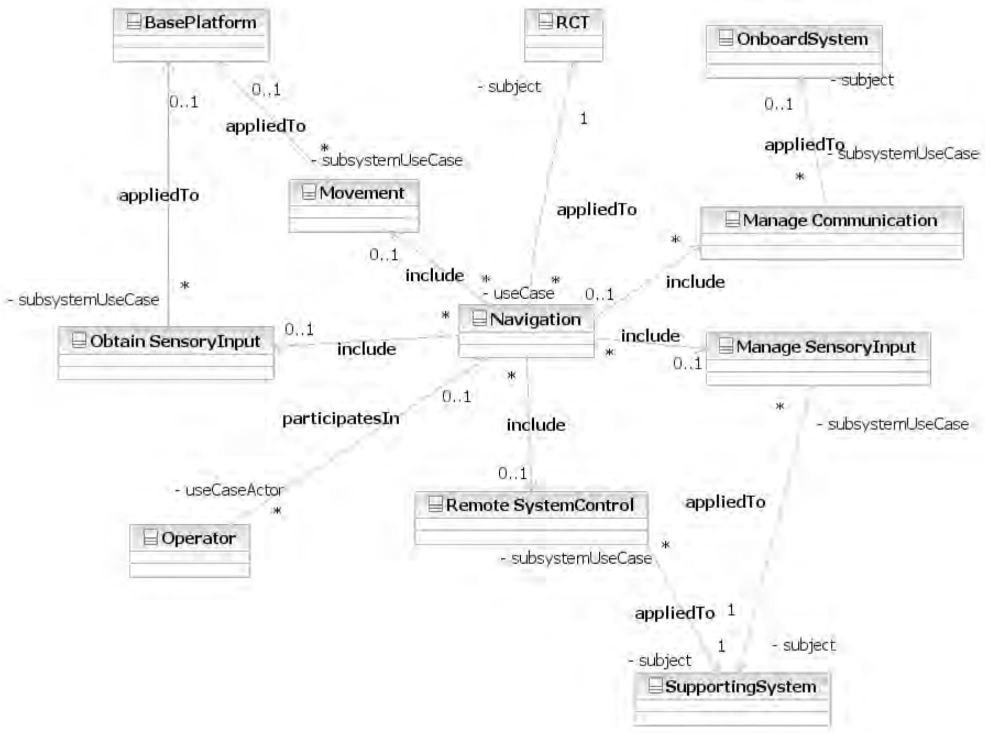


Figure 9.11: RCT Navigation UseCaseModel detailed for Subsystems

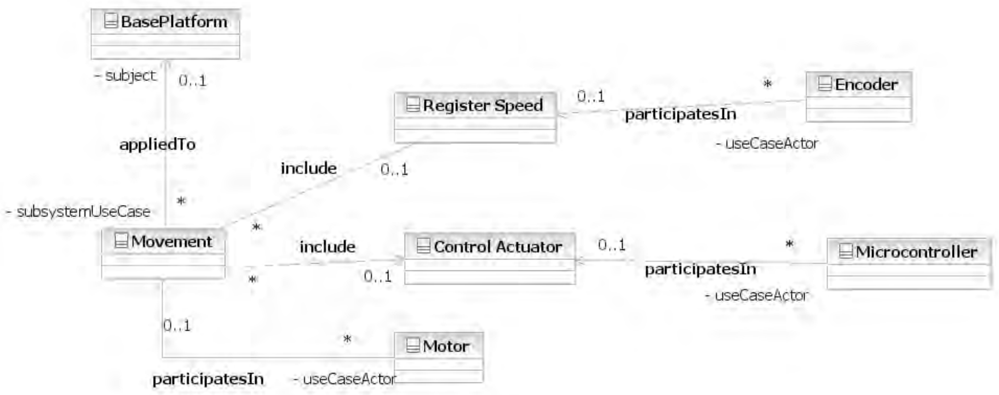


Figure 9.12: BasePlatform Subsystem UseCaseModel

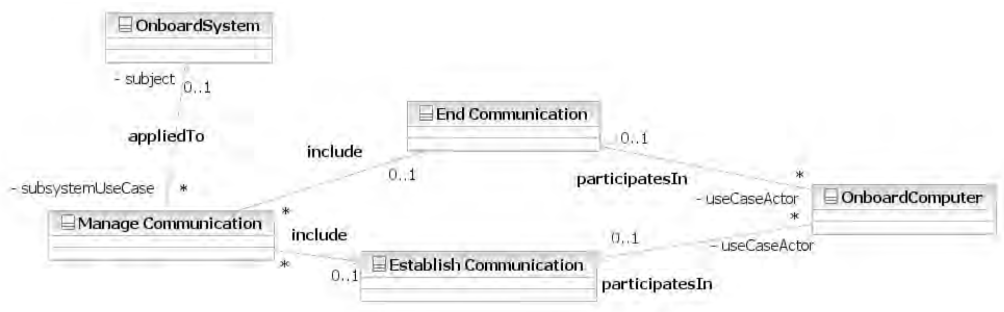


Figure 9.13: OnboardSystem Subsystem UseCaseModel

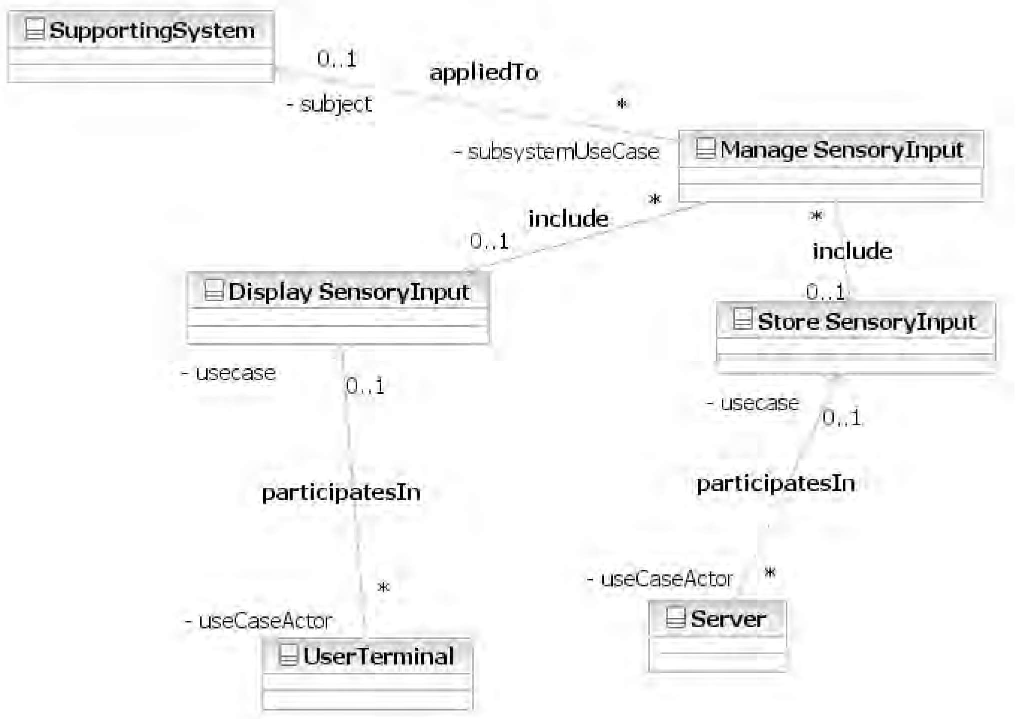
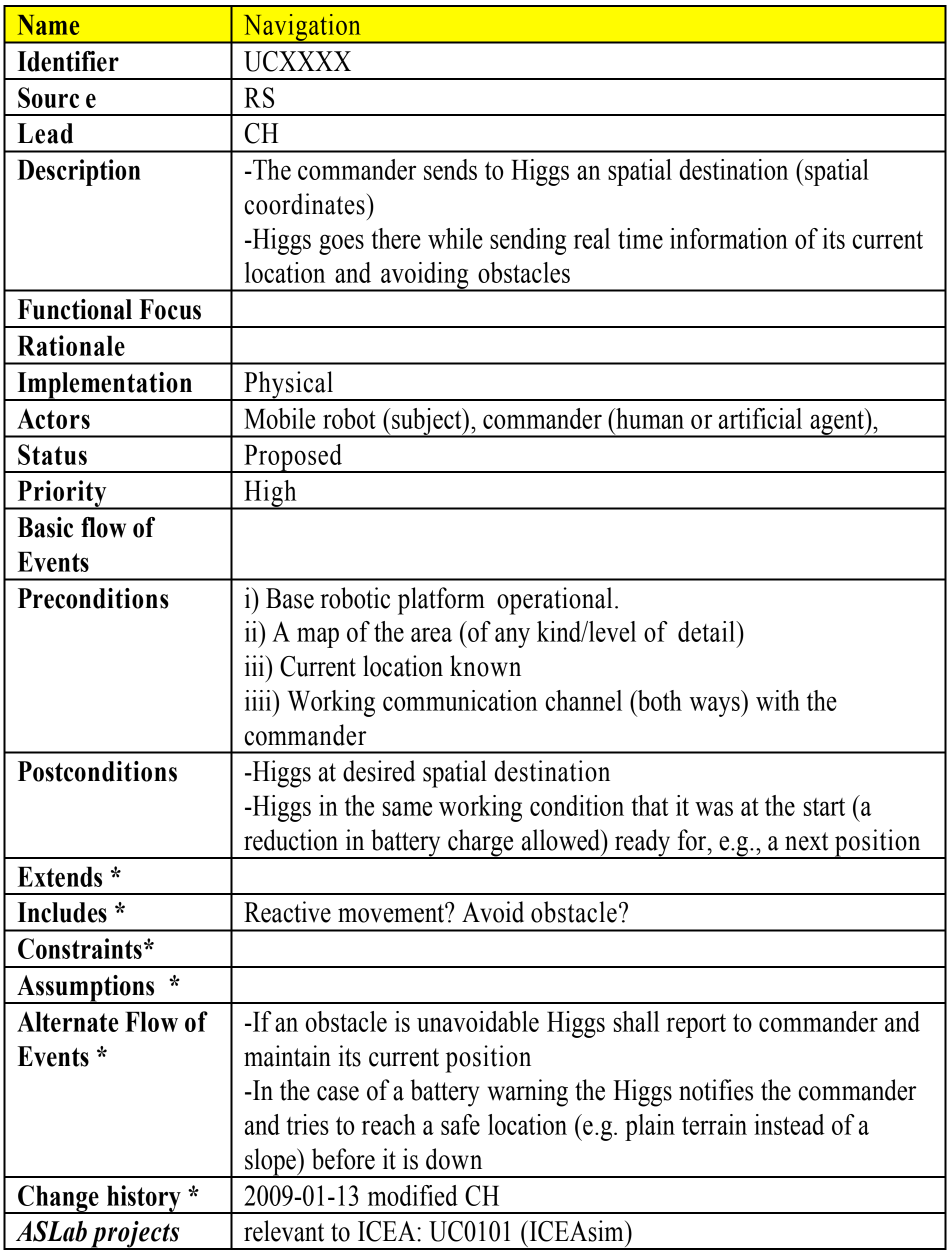


Figure 9.14: SupportingSystem Subsystem UseCaseModel



Name	Navigation
Identifier	UCXXXX
Sourc e	RS
Lead	CH
Description	-The commander sends to Higgs an spatial destination (spatial coordinates) -Higgs goes there while sending real time information of its current location and avoiding obstacles
Functional Focus	
Rationale	
Implementation	Physical
Actors	Mobile robot (subject), commander (human or artificial agent),
Status	Proposed
Priority	High
Basic flow of Events	
Preconditions	i) Base robotic platform operational. ii) A map of the area (of any kind/level of detail) iii) Current location known iiii) Working communication channel (both ways) with the commander
Postconditions	-Higgs at desired spatial destination -Higgs in the same working condition that it was at the start (a reduction in battery charge allowed) ready for, e.g., a next position
Extends *	
Includes *	Reactive movement? Avoid obstacle?
Constraints*	
Assumptions *	
Alternate Flow of Events *	-If an obstacle is unavoidable Higgs shall report to commander and maintain its current position -In the case of a battery warning the Higgs notifies the commander and tries to reach a safe location (e.g. plain terrain instead of a slope) before it is down
Change history *	2009-01-13 modified CH
ASLab projects	relevant to ICEA: UC0101 (ICEAsim)

Figure 9.15: UseCase Detailing Subtask for the Navigation requirement

Name	Survival
Identifier	UCXXXX
Sourc e	CH
Lead	CH
Description	The robot is pursuing a mission that is permanent in time (e.g. patrol) and has to recharge itself appropriatedly so as to keep achieving the objectives of the mission
Functional Focus	robustness
Rationale	
Implementation	Physical
Actors	Mobile robot, human operator, environmental entities
Status	Proposed
Priority	High
Basic flow of Events	-
Preconditions	i) Base robotic platform operational. ii) Location of the charging points
Postconditions	
Extends *	
Includes *	Identification
Constraints*	
Assumptions *	
Alternate Flow of Events *	
Change history *	2009-01-13 created CH
Open issues *	doubtable formulation, is this a UC?
ASLab projects	ICEA: UC0101 (ICEAsim)
Free slots *	

Figure 9.16: UseCase Detailing Subtask for the Survival requirement

description.

9.3.3 RCT ASys Analysis Phase

The ASys Analysis Phase consists of differents *Tasks*, such as *StructuralAnalysis*, *BehaviouralAnalysis*, and *FunctionalAnalysis*. The differents tasks have been carried out for the Robot Control Testbed, as described in following sections.

Structural Analysis

The RCT *StructuralAnalysis Task* (Fig. 9.17) pays attention to the RCT from a *StructuralViewpoint*, i.e., the analysis considering the system's structure in terms of subsystems and elements. As outcome of the task, a RCT*StructuralModel* is obtained, consisting of RCT *StructureModel*s and RCT *Topology Model*s.

- System Modelling

  For the RCT *StructureModel*, the different subsystems part of it were determined according to the structural definition provided. Hence, three different kind of subsystem were identified: the platform, the onboard system attached to it, and the supporting system which are not physically part of the robot but take part into its operation. The platform was further analysed to identify the subsystems or elements part of it. The different onboard systems were identified. Finally, the supporting systems were specified. The overall analysis result was formalised as a *StructureModel* for the RCT (Fig. 9.18).

  The *TopologyModel* allows representing the topological connections between a system's parts, by refining the Topology Package concepts. For the Robot Control Testbed, it was considered important to differentiate the topology from an informational and a physical aspects. The first one refers to the communication connections between parts, such as WiFi, Ethernet, etc. The latter, the physical connections among parts in a traditional way. Hence, two different topology models have been obtained for the RCT system: the RCT Informational Topology Model (Fig. 9.19), and the RCT Physical Topology Model (Fig. 9.20).

  Each one of the subsystems and elements identified in the *StructureModel*, can be further detailed during a design phase to represent their characteristics and roles, by ontologically instantiating the concepts in the PhysicalDevice Package into a *DeviceModel*.

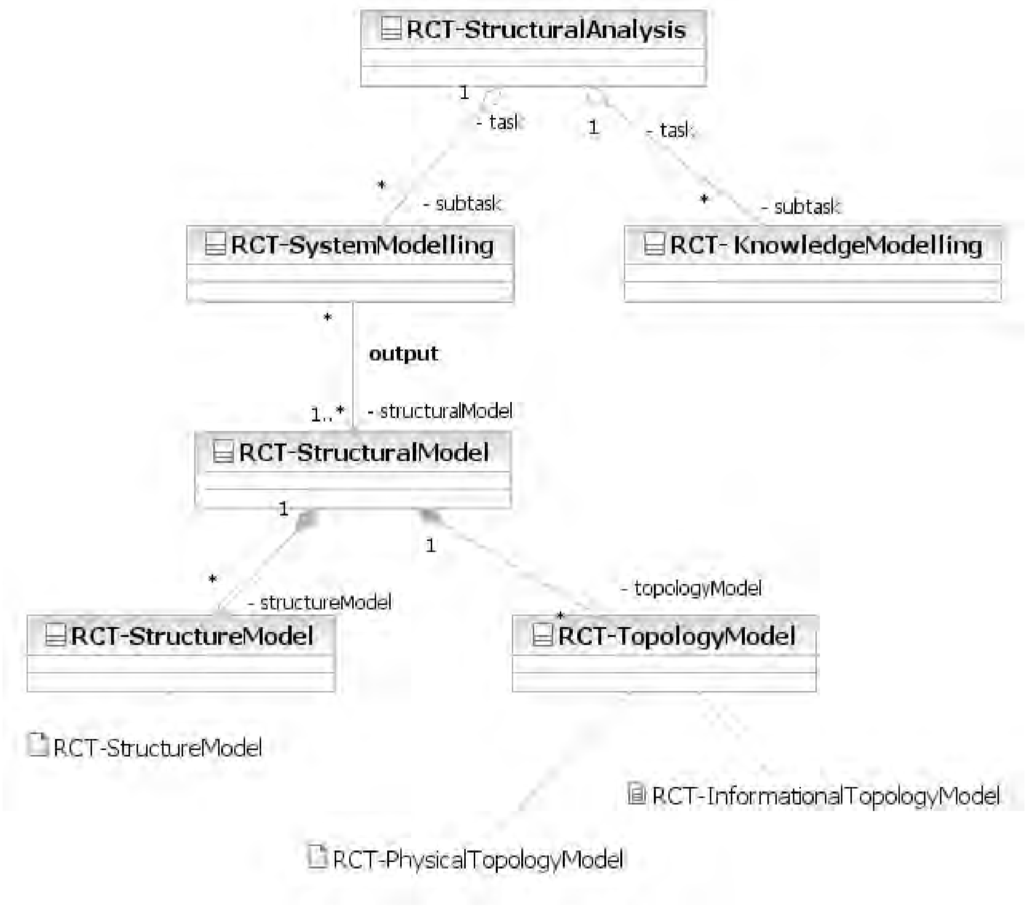


Figure 9.17: RCT Structural Analysis Task

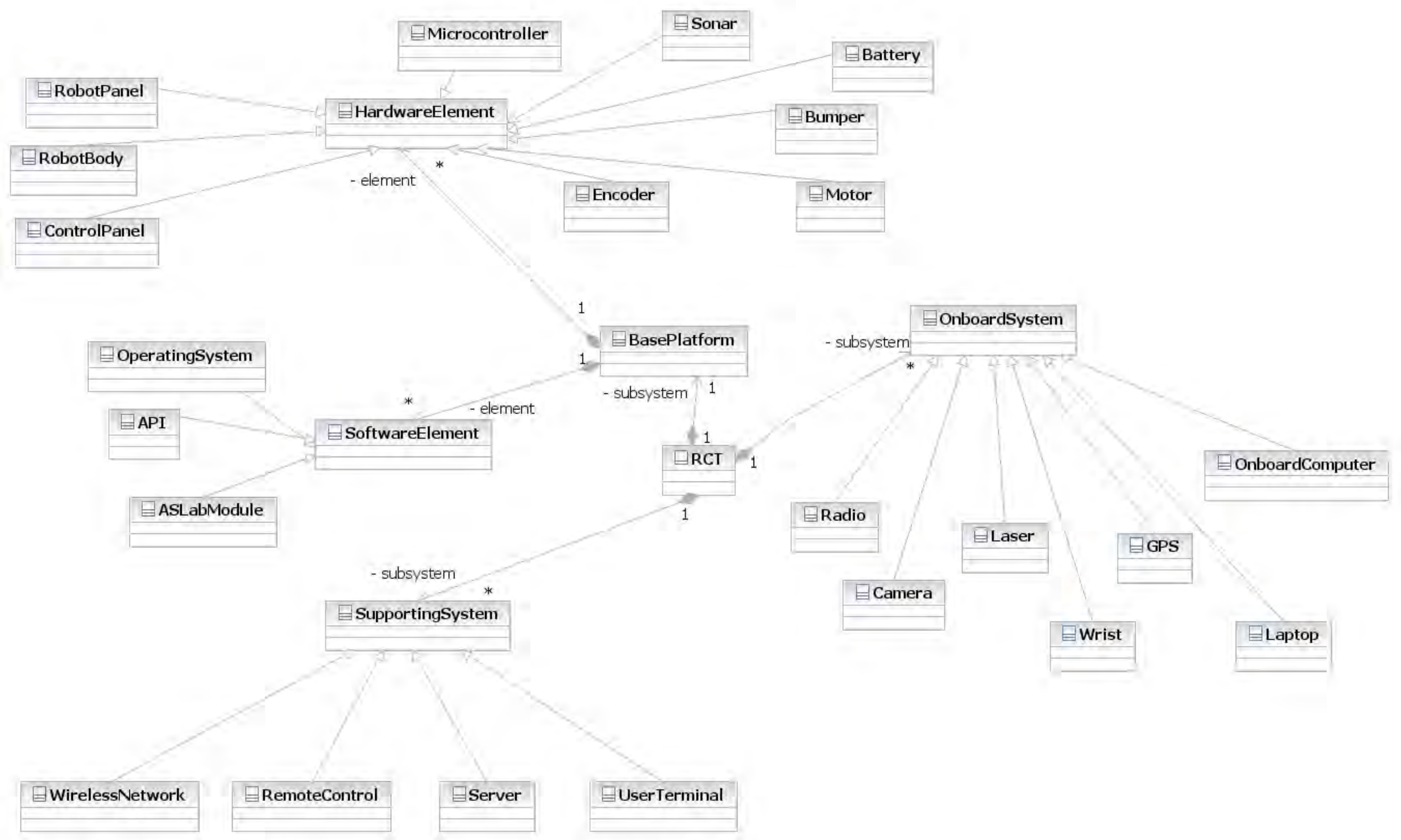


Figure 9.18: RCT Structure Model

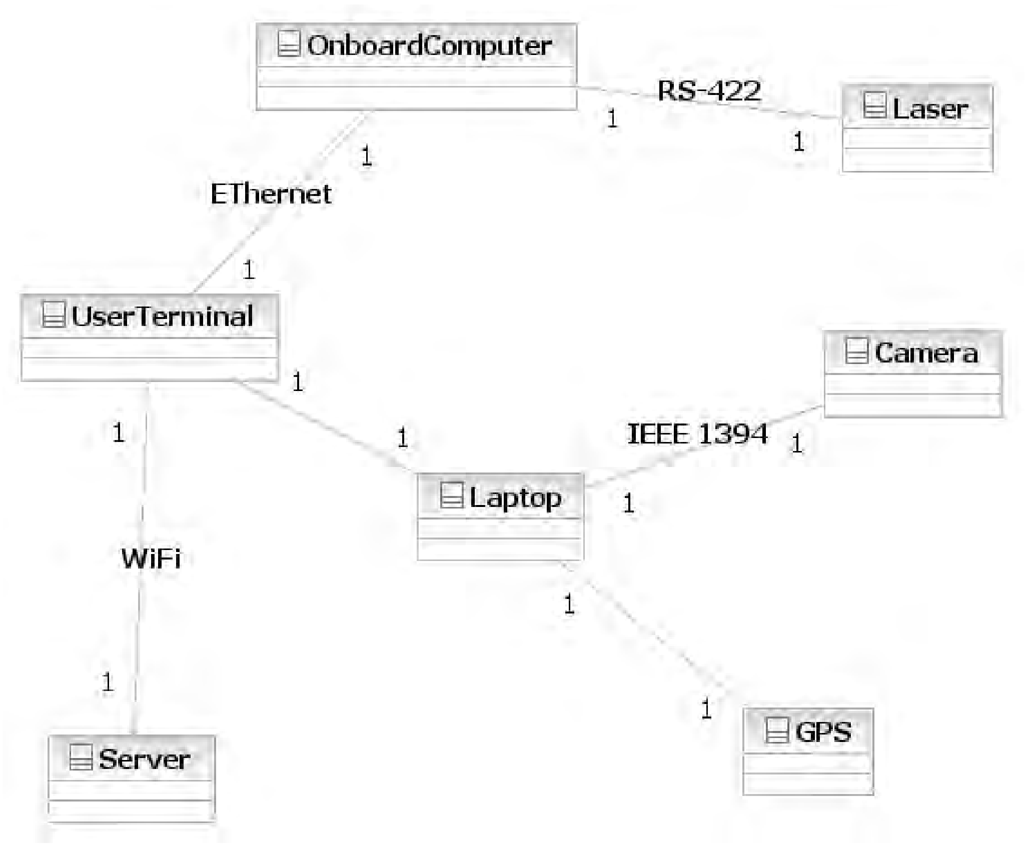


Figure 9.19: RCT Topology Model: informational connections

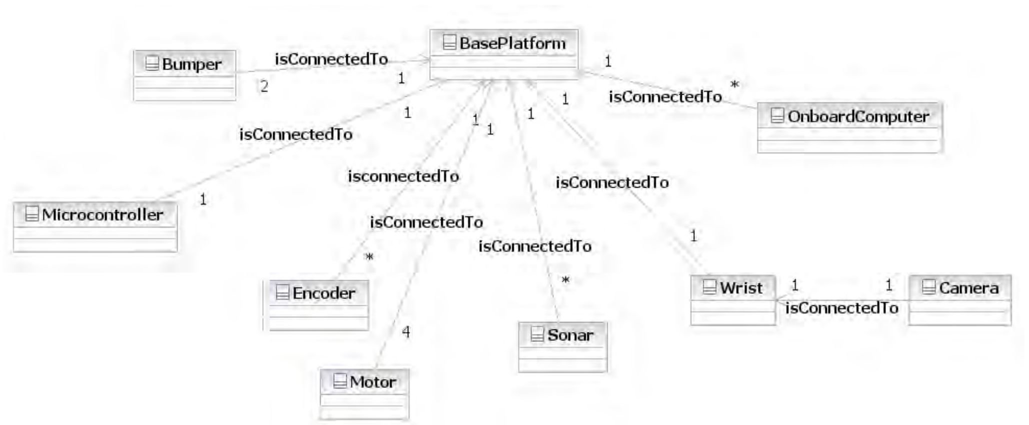


Figure 9.20: RCT Topology Model: physical connections

9.4 Process Control Testbed

The Process Control Testbed (PCT) is the chemical pilot plant designed to test the application of autonomous system ideas to continuous processes. Its main component is a Continuous Stirred Tank Reactor (CSTR), as well as the related instrumentation and control system. The aim is to provide the system with cognitive capabilities to carry out complex tasks such as fault diagnosis, alarm management, and control system reconfiguration. The following sections summarise the detailed description and underlying ideas of the PCT, as described in [dlM09b], [dlM09a], and [Rod07].

9.4.1 PCT Structural Description

The CSTR: it is composed of a stainless steel vessel reactor (Fig. 9.21) provided with a stainless steel jacket for cooling and heating. To improve the mixing process, the reactor has been provided with a stirring unit (Fig. 9.22) powered by a motor.

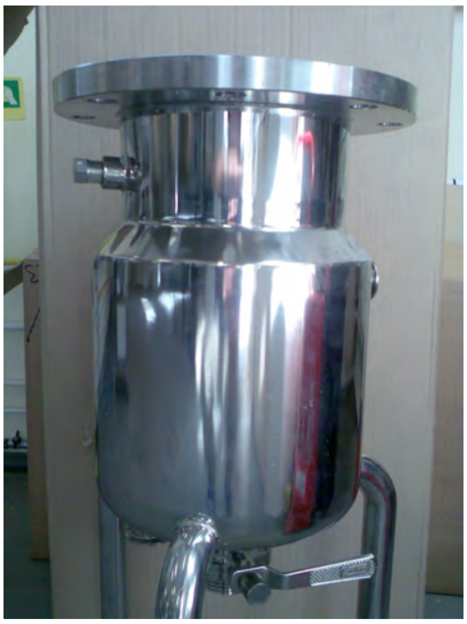


Figure 9.21: CSTR reactor vessel

Hydraulics: it is composed of the different tubing (for the reactants, the products, and the cooling water and steam), the pumps used to feed the reactants to the reactor, and the control and relief (safety) valves.

Storage: two small tanks for reactants storage and one for the product.

Instrumentation: it consists of several sensors (one for pH, one for temperature, and one for pressure). Moreover, there is an oxidation–reduction potential (ORP) sensor,

Figure 9.22: Reactor stirring unit

an electrochemical analyzer, and two electromagnetic flowmeters to test and to control the different elements (reactants, products and steam) in the system.

Control System: the control system main element is a conventional computer, together with National Instrument's data acquisition boards for input/output analog and digital signals (current and voltage).

9.4.2 PCT Functional Description

The system, as depicted in Fig. 9.23 is composed of a CSTR type reactor (R01) where two input streams are fed by their corresponding pumps (P01 and P02), and an outlet stream. The chemical reactions taking place in the reactor should to be cooled with water or heated with steam, depending on the reaction. The stirring unit assures the homogeneous composition in the reactor. There is also a relief valve (SV01).

The P&I diagram in Fig. 9.24 shows the basic control structure for the endothermic reaction (steam heating) case. The control strategies used are:

- Temperature: the temperature in the reactor is controlled by acting upon the steam inlet flow into the jacket.
- Level: it is not necessary to control the reactor's level, as its output is by overflow.

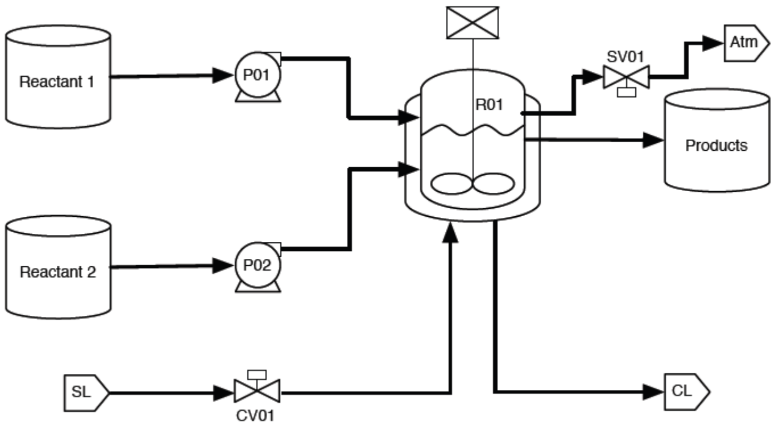


Figure 9.23: PCT Process Diagram from [dlM09b]

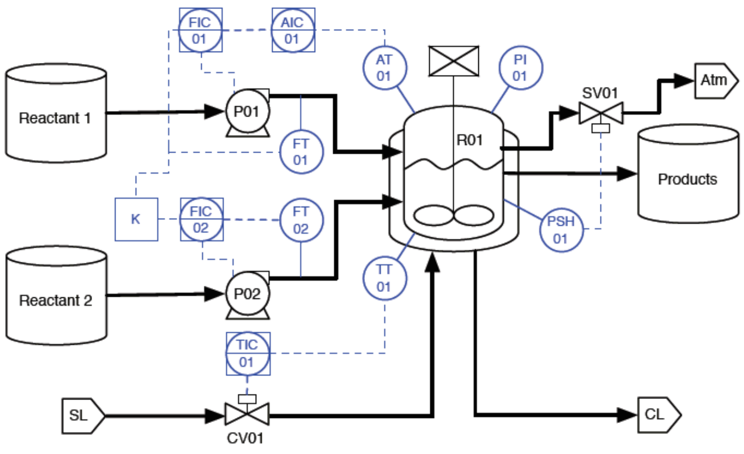


Figure 9.24: PCT Control Diagram from [dlM09b]

- Pressure: it is not controlled but supervised with a pressure sensor.
- Composition: a cascade loop will be used. Composition regulator controls the set point of one of the inlet streams.
- Inlet flows: one of them is controlled by the composition loop, whereas the second one is proportional to it. Products must be fed in a stoichiometric ratio.
- Stirring: there is no need to control the stirring in the reactor.

9.5 OASys–based Engineering Methodology Applied to the PCT

The OASys–based Engineering Methodology has been applied to the Process Control Testbed during its analysis, as a process where the different elements interacting in the testbed have been represented using the ontological elements of OASys. As a result different tasks have been carried out and several work products have been obtained. The terms in italic refer to those ontological elements defined in the different subontologies and packages of OASys.

9.5.1 PCT ASys Views Phase

The objective of this *Phase* is to identify the different views of interest for the Process Control Testbed considering the Perspective Package elements, which are specialised through the ASysPerspective Package concepts.

For example the *RequirementViewpoint* specialises the concept of *Viewpoint* to address the analysis of the PCT from a *PCTRequirement Concern*, to establish the different requirements to be defined in the system. Considering this viewpoint, a PCTRequirementViewpointModel can be instantiated (Fig. 9.25). In this model, the PCTRequirementView is a *View* that will consist of a PCTUseCaseModel, which is an *EngineeringModel* to be obtained as output of the *UseCaseModelling Subtask*.

The *StructuralViewpoint* pays attention to the autonomous system structure, as considering the *ASysStructure* concern. A PCTStructuralViewpointModel can be obtained (Fig. 9.26) by instantiating the different concepts, relationships and attributes in the Perspective and the ASysPerspective Packages. The original concepts have been ontologically instantiated into PCT related ones.

The *FunctionalViewpoint* focus on the analysis of an autonomous system's functions, to be considered in the PCTFunctionalViewpointModel (Fig. 9.27). It instantiates the Perspective and the ASysPerspective Packages ontological elements,

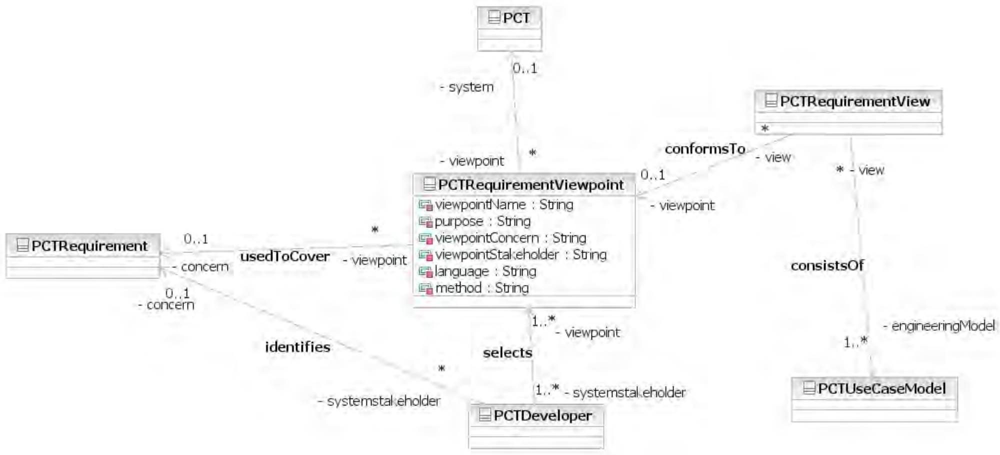


Figure 9.25: PCT RequirementViewpoint Model

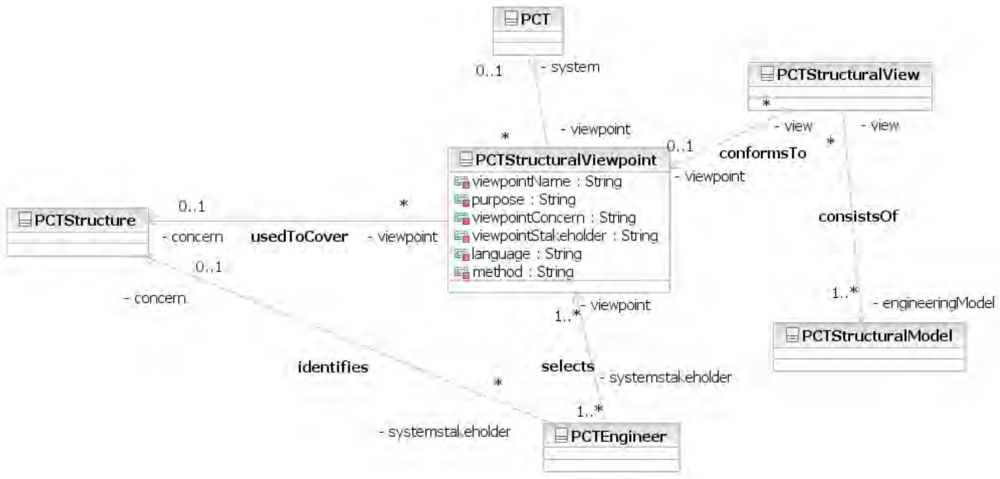


Figure 9.26: PCT StructuralViewpoint Model

focusing on the functional aspects to consider for the PCT. In this model, the PCTFunctionalView is a *View* formalised as a PCTFunctionalModel, which is the *EngineeringModel* consisting of different function related models, obtained as outputs of the *FunctionalModelling Subtask*.

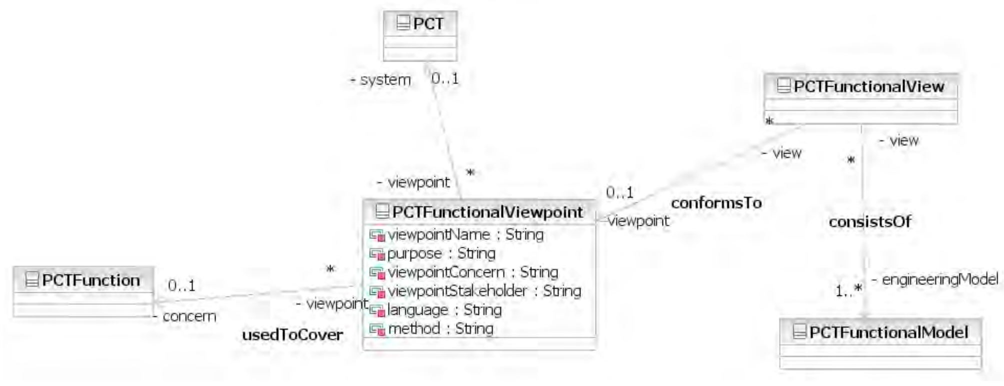


Figure 9.27: PCT FunctionalViewpoint Model

9.5.2 PCT ASys Requirement Phase

The aim of this engineering phase is to identify requirements for the Process Control Testbed, using common requirement engineering techniques to specify them. Within this *Phase*, the *SystemUseCase Task* is dedicated to the development of the system's *UseCases*, in this case the PCT UseCases during the *UseCaseModelling* and the *UseCaseDetailing Subtasks* (Fig. 9.28).

- UseCase Modelling
  The PCT *UseCaseModel* (Fig. 9.29) was obtained as a work product in the PCT *UseCaseModelling Subtask*, considering the ontological elements in the Requirement Package of the SystemEngineering Subontology.

  Some use cases appearing in the PCT *UseCaseModel*, have been further analysed to identify actors and additional requirements. As result, the Start Plant (fig. 9.30), the Stop Plant (Fig. 9.31), and the Operate Plant (Fig. 9.32) Use Case Models were obtained.

- UseCase Detailing
  Each *UseCaseModel* has been further detailed during the *UseCaseDetailing Subtask*, using the ASys UseCase Pattern, by filling the relevant information in each field as required. Figure 9.33, and Figure 9.34 show the results, respectively, of detailing the Feed Reactants and Make Reaction elements that

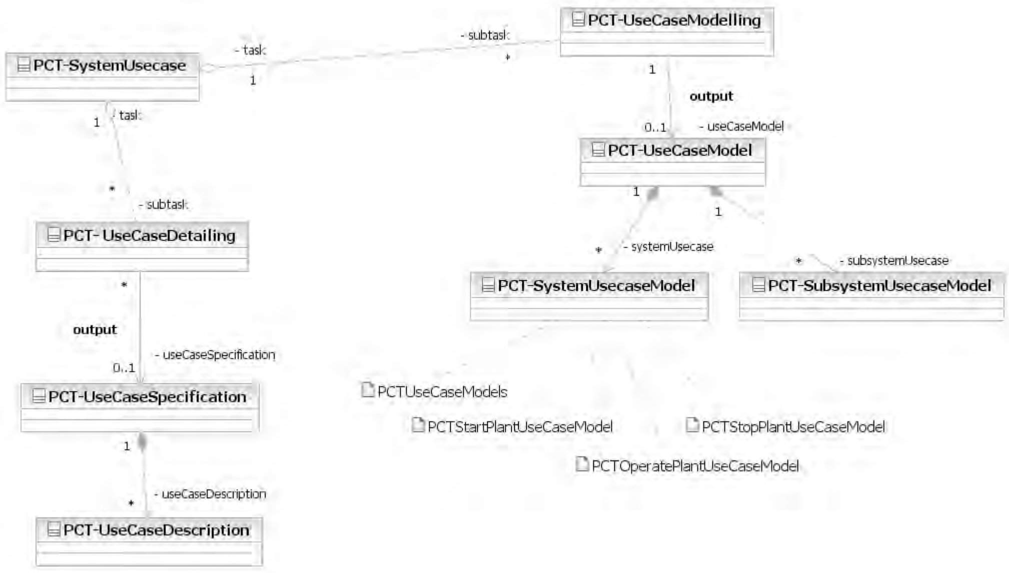


Figure 9.28: PCT UseCase Task

appear in the PCT System *UseCaseModel*.

9.5.3 PCT ASys Analysis Phase

This *Phase* has as objective to analyse the Process Control Testbed from different *Viewpoint*s, to consider its structure, behaviour and function. Here, the emphasis has been made on the *StructuralAnalysis* of the *System* in terms of *Subsystem*s and *Element*s.

Structural Analysis

The PCT *StructuralAnalysis* has as objective the analysis of a system considering its structural aspects, under a *StructureViewpoint* (Fig. 9.35). Different *EngineeringModel*s can be obtained as result of performing the two main *Subtasks*: *SystemModelling* and *KnowledgeModelling*. These *Subtasks* for the Process Control Testbed are described in the next sections.

System Modelling is one of the *Subtask*s considered in the *StructuralAnalysis*, to obtain the PCT *StructuralModel*. The PCT *StructuralModel* is a model kind to describe an autonomous system from a structural viewpoint that conforms, as a matter of fact, to a specific level of detail that could be further refined through domain focalisation. In this case, the PCT *StructuralModel* consists of different

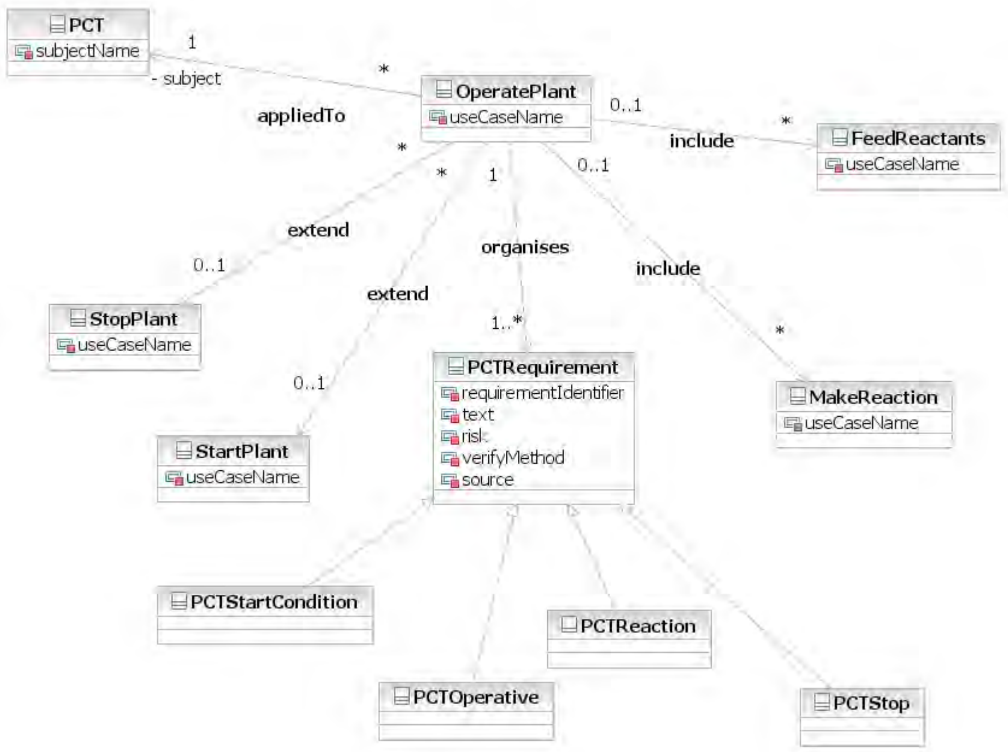


Figure 9.29: PCT System UseCase Model

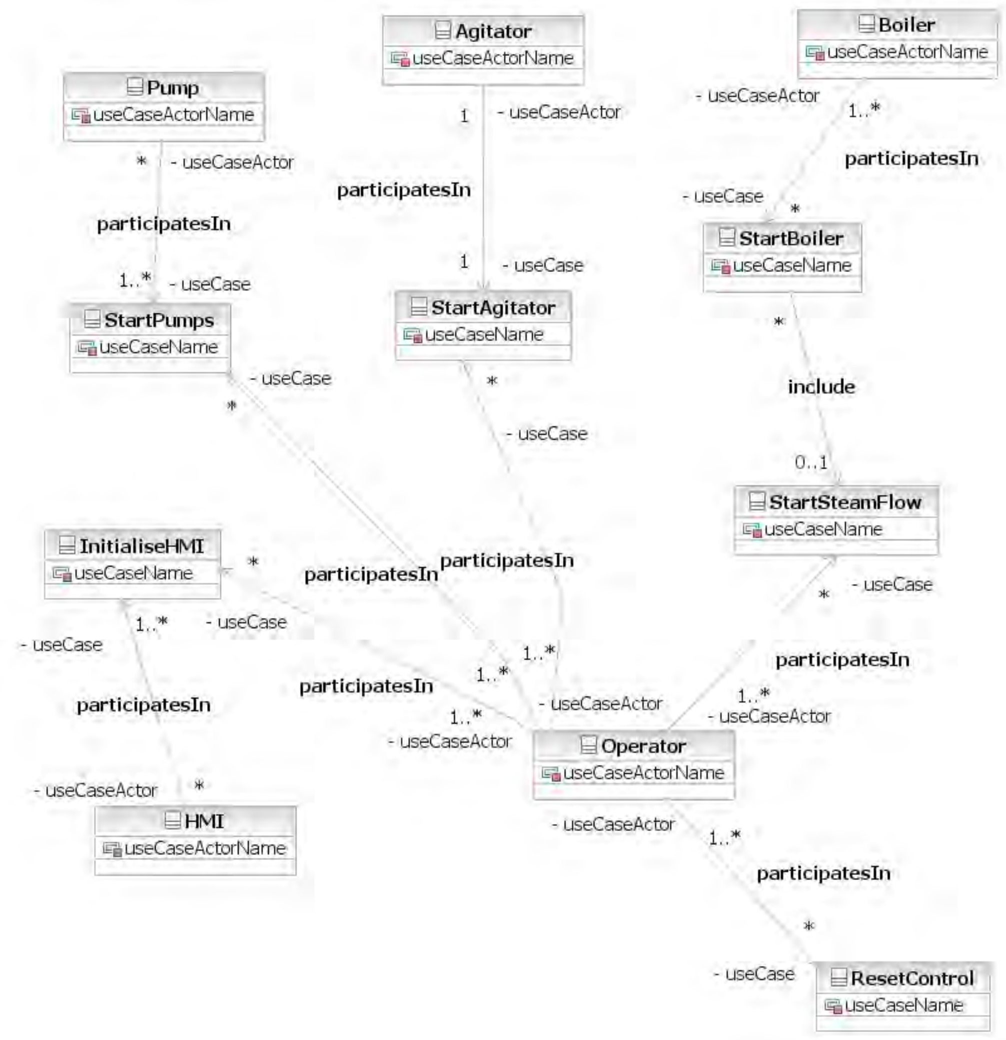


Figure 9.30: PCT Start Plant UseCase Model

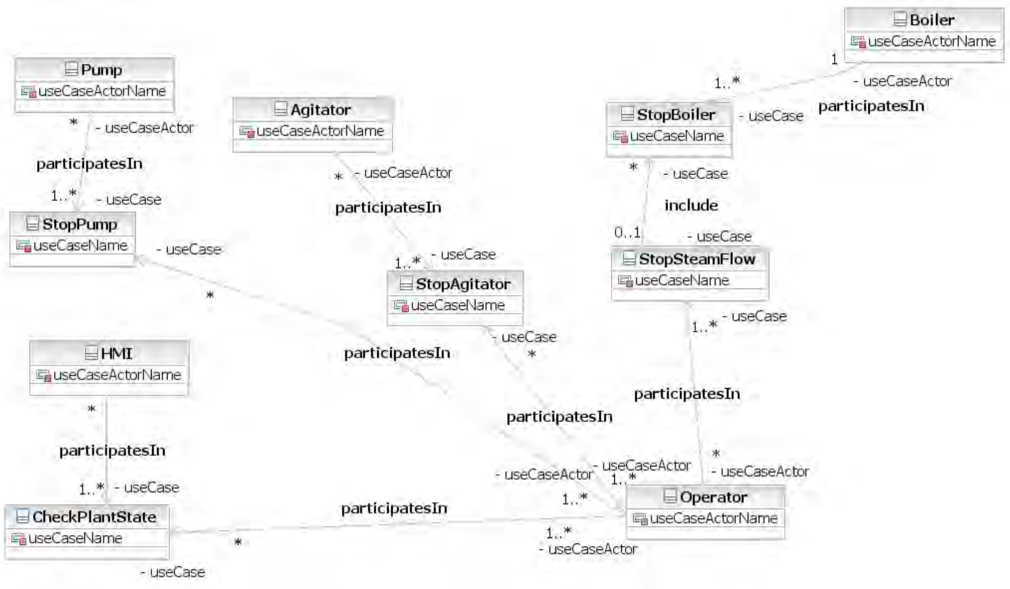


Figure 9.31: PCT Stop Plant UseCase Model

*StructureModel*s to describe the components in the testbed, as well as the *TopologyModel*s to describe topological connections among the components in the system. The StructureModel specialises the ontological elements in the General Systems Theory and Mereology Packages, to describe the structural features of the testbed. The Topology Model refines the concepts in the Topology Package.

The PCT StructureModel shows that in general, a *System* consists of *Subsystems*, i.e., a system that is constituent of another one, which in turn can be decomposed until reaching to its *Elements*, which cannot be further decomposed. The Process Control Testbed consists of the CSTR Reactor, the Hydraulic subsystem, the Control System and the Instrumentation (Fig. 9.36).

In the model, the relationships between the *System* (the PCT), and the *Subsystems* (CSTRreactor, Hydraulics, ControlSystem, Power, Instrumentation and Hydraulics) are expressed by means of the UML composition association. If it is necessary to emphasise the existence of decomposition levels in a system, the different relationships defined in the Mereology Package could be applied. For example, the concept PartofComposite can be used to further specialise a subsystem. The PartOnly concept allows for specifying an object that is part of another, but without parts of its own. The isExclusiveltyPartOf and the isPartOf relationships can specify the existing relation between the parts of the system at the higher and more general level of abstraction. These relationships can be refined when referring to a subsystem or element, as in the case of the isSubsystemOf relationship to detail that the relation is between a system and one of its subsystems. For the PCT

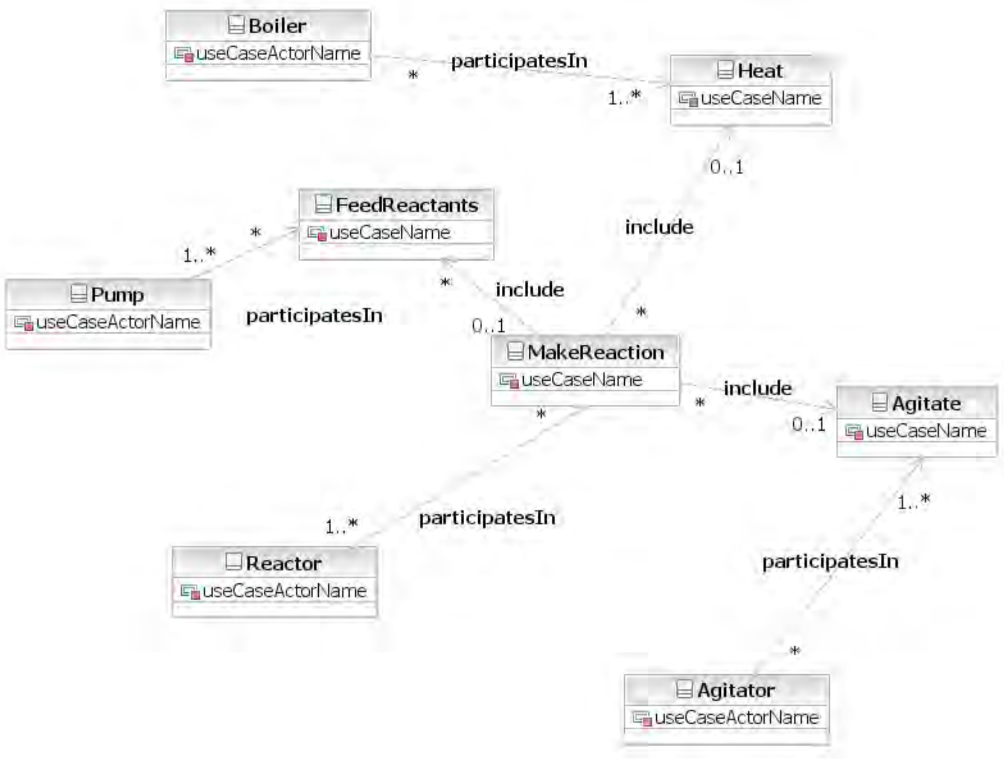


Figure 9.32: PCT Operate Plant UseCase Model

Name	Feed Reactants
Identifier	UC0001
Source	MR
Lead	MR
Description	The process plant needs reactants to produce the products, so a continuous flow of reactants is needed
Functional Focus	Continuous flow of chemical species
Rationale	
Implementation	Physical
Actors	Pumps and pipes
Status	Proposed
Priority	High
Basic flow of Events	- The pump suctions the reactants from the tanks - The reactants flow through the pipes of the reactor
Preconditions	Existence of reactants
Postconditions	Continuous flow of reactants

Figure 9.33: PCT feed reactants Use Case Description

Name	Feed Reactants
Identifier	UC0001
Source	MR
Lead	MR
Description	The process plant needs reactants to produce the products, so a continuous flow of reactants is needed
Functional Focus	Continuous flow of chemical species
Rationale	
Implementation	Physical
Actors	Pumps and pipes
Status	Proposed
Priority	High
Basic flow of Events	- The pump suctions the reactants from the tanks - The reactants flow through the pipes of the reactor
Preconditions	Existence of reactants
Postconditions	Continuous flow of reactants

Figure 9.34: PCT make reaction Use Case Description

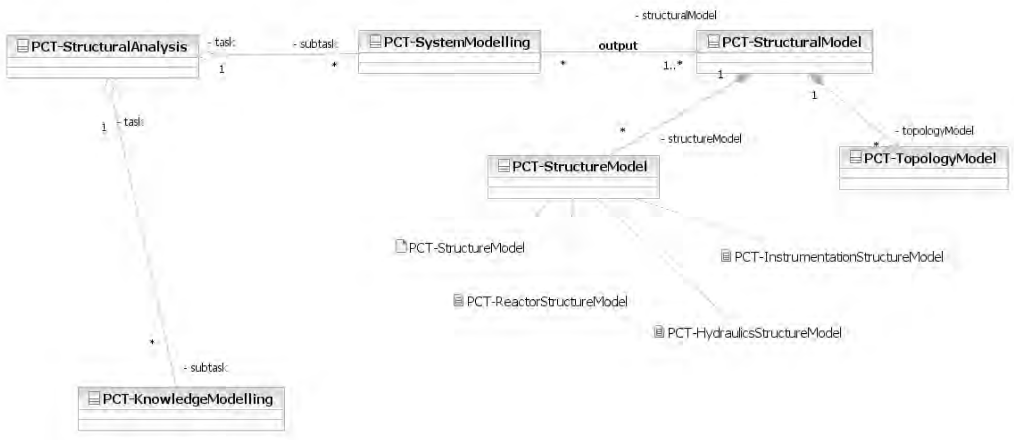


Figure 9.35: PCT Structural Analysis Task

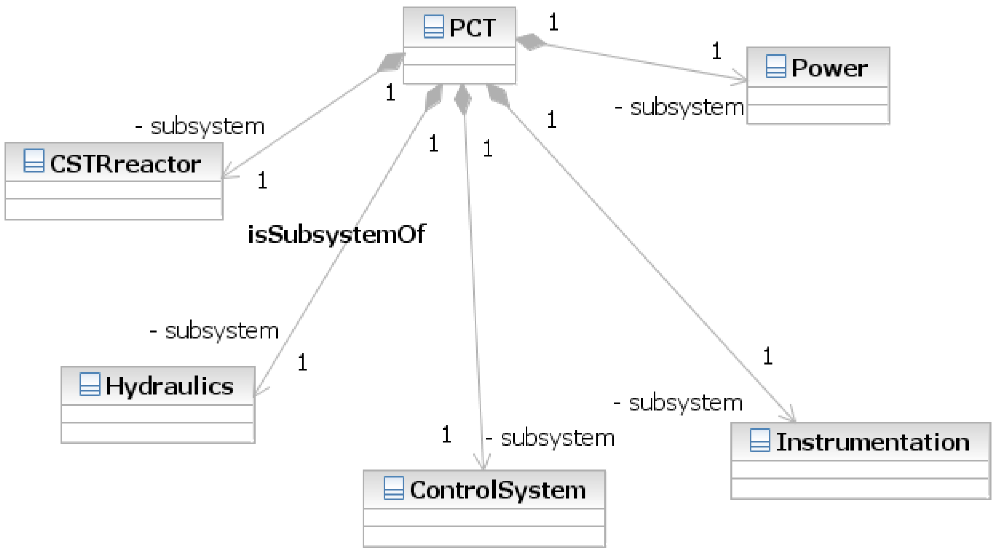


Figure 9.36: PCT Structure Model

Structure Model, the role as subsystems has been specified through the UML association role, to avoid cluttering the diagram.

Additional *StructureModel*s were obtained when a particular *Subsystem* such as the CSTR Reactor (Fig. 9.37), the Hydraulics (Fig. 9.38), the Control System, the Instrumentation (Fig. 9.39) become the system under analysis. The combined use of these ontological concepts provides for the description of the structural relations among the components in the Process Control Testbed.

Once again, to point out that the relationships between the different *Subsystems* and *Elements* are modelled directly using UML composition, aggregation and generalization associations, detailing the classes roles in them. If required, the associations could be changed into directed associations detailed by using the terms in the Mereology Package. However, it has not been done here to avoid cluttering the model.

Another element of the *StructuralModel* is the *TopologyModel*, to detail the connections of the elements previously considered in the *StructureModel*. This model can be obtained by instantiating the concepts defined in the Topology Package.

With this set of Models, the *StructuralAnalysis* of the Process Control Testbed is accomplished. Further analysis will pay attention to its *Behaviour*, its *Functions*

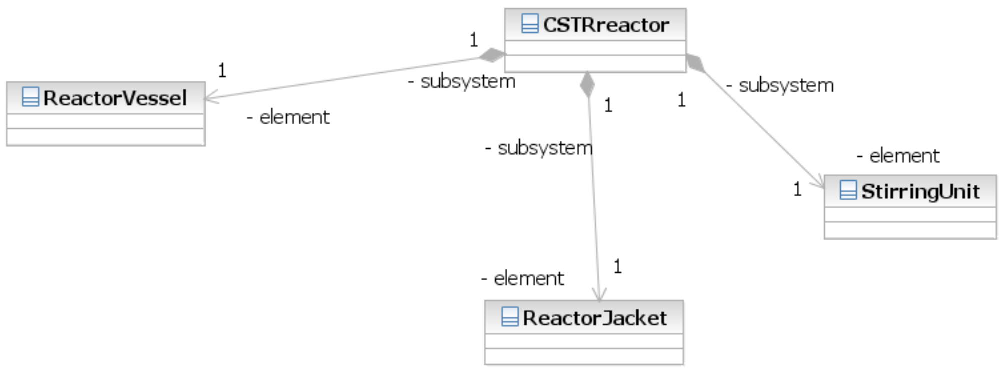


Figure 9.37: CSTR Reactor Structure Model

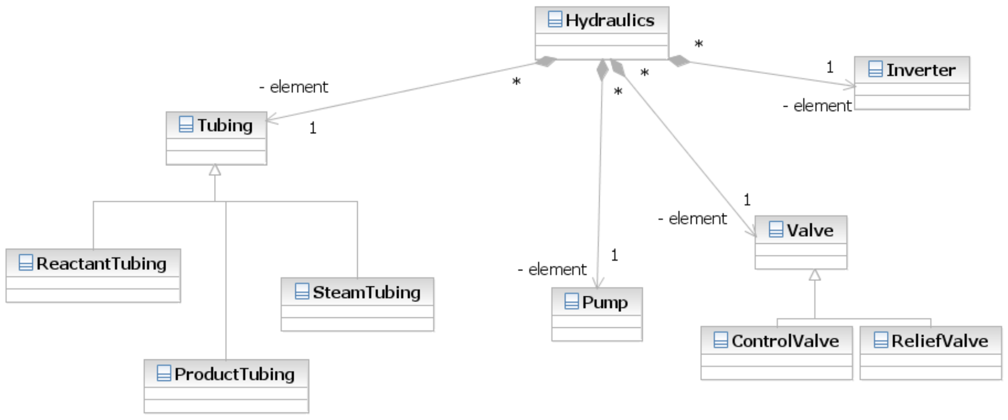


Figure 9.38: Hydraulics Structure Model

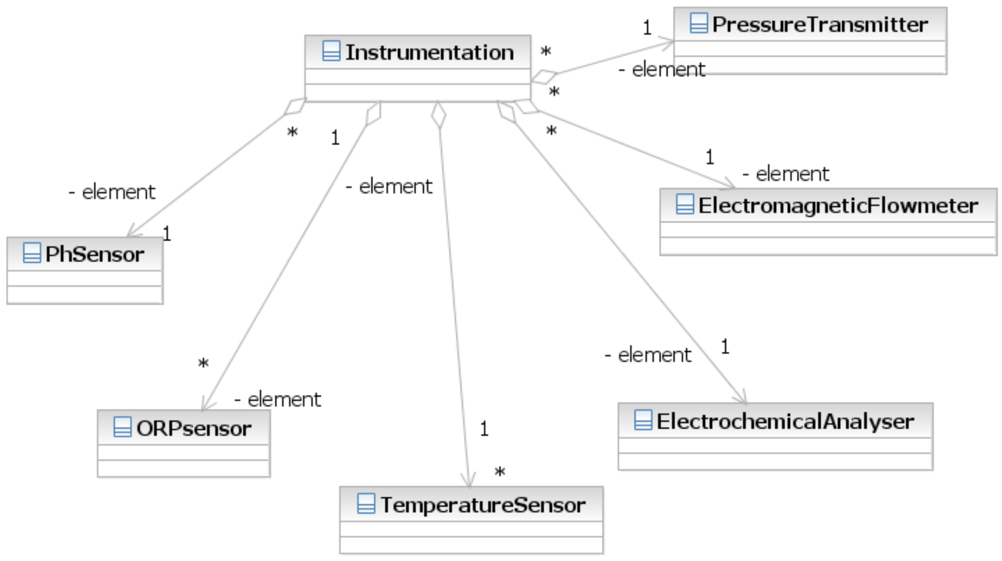


Figure 9.39: Instrumentation Structure Model

to describe the cognitive capabilities in the control system to be developed.

Part IV

Conclusion and Annex

Chapter 10

CONCLUSION AND FURTHER WORK

The research motivation, scope and objectives were described in Ch. 1. The discussion on how the objectives have been finally accomplished by the development of the Ontology for Autonomous Systems (OASys), and its related methodology is given in this chapter. Shortcomings of the developed research to fulfill the initial requirements are also analysed. As a result, a proposal of aspects or research lines identified throughout the research to be further developed is given.

10.1 A Review of the Research

The general purpose of the research was to to capture and exploit the concepts to support the description and the engineering process of any autonomous system, as part of the ASys research programme which addresses the development of a technology for autonomous systems engineering, regardless of its particular application. With this purpose, two different elements have been considered and developed [BASRH10b]:

1. *An Ontology for the domain of Autonomous Systems (OASys)*
   As part of the ASys research programme, OASys was initially conceived with a similar role to those found in the literature as seen in Chapter 4 and Chapter 6: an ontology as software asset at design and development time as a representation–based mechanism in the form of a domain ontology, to be re–used in future applications. Hence, OASys has been used to carry out **an ontological conceptualisation of the domain of autonomous systems**. OASys has initially played a similar provided similar benefits, such as clarifying the structure of knowledge or allowing knowledge sharing and reuse among

different agents. Additional advantages in our case have been a common conceptualisation not only of the autonomous systems domain but also of their engineering process as described by the developers, providing a thorough and common understanding of the domain. Moreover, we have conceptualised not only the entities, actors and resources part of an autonomous system, but more importantly the capabilities we consider essential in being autonomous, such as the perception process, the goal–orientation, and the functional way of achievement and adapting.

OASys has been developed according to a comprehensive set of requirements (Ch. 7) which have been fulfilled by the following features:

Structure: OASys has been organised according to two orthogonal dimensions: one consisting of the different levels of abstraction, the other consisting of the separation between the system analysis and its engineering process.

To address the different levels of abstraction, OASys has adopted a layered structure to separate general knowledge from particular one, therefore considering different levels of abstraction to conceptualise the autonomous systems domain. The more abstract level is the System Domain Layer, which contains the ontological elements about a general system characterisation and engineering. A second layer, the ASys Domain Layer gathers the ontological constructs to describe and charaterise an autonomous' system structure, function and behaviour. The focus lies on autonomous systems as conceptualised in the framework of the ASys project, however being general enough to be re-used in the development of any other autonomous system.

To separate between the system's analysis and its engineering process, OASys has been divided in two ontologies. ASys Ontology (Ch. 7 Sec. 7.4.1) gathers the ontological elements for an autonomous system's analysis. Likewise, ASys Engineering Ontology (Ch. 7 Sec. 7.4.2) collects the concepts and relationships to describe an autonomous system's engineering process.

Design criteria: To assure its coherence and quality, OASys has been developed bearing in mind the following design criteria

- Extendibility: A major issue to be addressed in OASys is the possibility of its extension and modification. To support this feature, concepts within one layer have been classified using two different elements: subontologies and packages. Different subontologies have been considered, to contain subdomain–related concepts. Packages within each subontology were defined to gather related concepts.
- Clarity: Ontologies and glossaries have been reviewed to include their terms in OASys with the aim of using concepts which have existed and

been long used in the scientific community. The available documents were carefully analysed to extract the ontological elements, checking for mismatches or commonalities. The integration of terminologies, dictionaries, and existing ontologies has been addressed paying special attention to the granularity of their content. Those concepts have been later discussed with the group members to commit to the desired meaning for our research. Finally, all the ontological constructs (concepts, relationships, attributes, axioms) have been defined in natural language.

- Minimal encoding bias: To prevent the incorrect conceptualisation of concepts based on the final implementation language syntax, intermediate tabular representations and graphs have been used to define the ontological constructs.
- Minimal ontological commitment: On each layer only the fundamental concepts have been described. Lower layers elaborate on these concepts by adding new ones to provide a deeper level of detail. New concepts will be added as new applications are developed using OASys.
- Standardization: A naming convention for the ontology elements have been chosen to make the ontology easier to understand and to avoid some common modeling mistakes.

Methodology: The research did not aim at developing a brand new ontological methodology, but to re-use a well-established among the several methodologies available [GPFLC04a]. METHONTOLOGY has been chosen as starting point since it proposes both a ontology development process, as well as an ontology life cycle closely intertwined [FLGPJ97]. The development process refers to which activities are performed when building the ontology. The ontology life cycle identifies when these activities should be carried out, by a set of stages that define which activities to perform in each stage and how they are related. Both the development process and the life cycle activities were initially followed for the development of OASys. Some additional guidelines described in [NM01, Miz04, BA07] have been also considered.

Formalisation: The ontology has been formalised using a software engineering general- and specific-purpose language such as UML [Obj09a, Obj09b]. UML is not an ontology development language, hence it has been considered to have some drawbacks for ontology engineering: lightweight ontologies [GPFLC04b], incomplete semantics and software heritage [Obj09c], or lack of inference mechanisms [GDD06d]. Its shortcomings have been addressed by the Ontology Definition Metamodel (ODM) [Obj09c] which defines metamodels, mapping and profiles to allow the interaction between UML and traditional ontological languages such as OWL [SWM04].

2. *An OASys–based Engineering Methodology*

The OASys–based Engineering Methodology, described in Ch. 8, is a proposal of a generic ontology-based autonomous systems development based on OASys ontological elements. The methodology provides some **guidelines and exemplify the application of the OASys ontological elements as semantic support in a generic autonomous system engineering process**. The OASys–based Methodology has proposed:

- A generic autonomous system engineering process that contain phases and tasks to carry out the autonomous system development.
- The techniques and guidances to assist the former tasks and subtasks.
- The work products to be obtained as a class of models.
- The related OASys elements to be used in the work products.

10.2 Major Contributions of the Research

OASys has gathered, conceptualised and analysed how the engineers developing the autonomous systems describe and characterise them, describing the different elements taking part in an autonomous system's operation. OASys has catered for the main elements in autonomous system's engineering: the perception process, the knowledge to be used, the system's goals, the thought process as functional decomposition, and the actors to carry out the system's actions.

OASys has considered the autonomous systems domain with a global view. Previous ontologies developed for this kind of systems focused on a constrained or limited approach, either being mobile robots or agent based systems. The approach here has been to define the different elements to describe the system structure and function in a way general enough to be reused among different applications. OAsys can be further complemented with additional subdomain, task or application ontologies, without losing its reusability and generality features.

OASys and its related methodology have provided several benefits:

- OASys has provided a common conceptualisation of the autonomous systems domain for autonomous system's developers.
- OASys allows to scale, manage the broad range of concepts, and prevent imprecise definitions and mismatches when referring to autonomous systems.
- The separation between the description and the engineering aspects in a system by means of the ASys Ontology and the ASys Engineering Ontology, has allowed to address independently the characterisation of the testbeds.
- The conceptualisation of different testbeds has allowed to identify commonalities between them, allowing to define the different testbeds in a subdomain–independent way. Particular properties for each testbed have been identified, which supports the ASys research programme idea of benefiting from existing (sub)domain ontologies as element of the asset base.

- The existence of different levels of abstraction within the ontology has shown its suitability to describe an autonomous system using the domain focalisation techniques.
- OASys improves the interoperability of applications, as they contain a core of common vocabulary.
- OASys serves as an ontological (meta)model for the conceptual modelling of an autonomous system.
- The OASys–based Engineering Methodology has suggested how autonomous systems can be characterised and should be engineered using conceptual models, in the form of a generic ontology–based engineering process for this kind of systems.
- The ontology–based conceptual process model provided, offers improvements on the knowledge sharing and reuse between the testbeds developers during the analysis phase.
- OASys and the methodology have helped to increase the realiability of the software models to be obtained.
- OASys for Model–driven and Ontology–driven Engineering: OASys and other software assets are to be used to obtain the autonomous system's models and views according to the user's requirements. OASys as an autonomous systems domain ontology can play an important role when using the Model Driven Architecture (MDA) ideas. Domain ontologies describe the entities and relationships of a domain, which can be used as support for the Computational Independent Model (CIM). The CIM is later extended towards a high level model, called Platform Independent Model (PIM), which in turn, is to be transformed to a platform specific model (PSM) and code.

10.3 Further Work

During the ontological engineering of OASys and its related methodology, some elements have appeared for a further development or to address shortcomings in the research [BASRH10a]. Here, a discussion on open issues and future lines of research is provided.

- OASys and domain ontologies

  OASys has allowed to conceptualise the domain of autonomous systems in a way general and reusable enough to address any kind of autonomous system. Nevertheless, the particular properties of different research testbeds has pinpointed the necessity to complement OASys with domain ontologies, by adding the differences of subdomains or applications. To address subdomain or application–oriented ontological elements, additional domain ontologies should be considered. A thorough analysis of these domain ontologies will decide its possible integration with the existing OASys, or suggest further extensions to

OASys in the form of additional packages or even an application oriented layer.

- OASys and the definition of Engineering Views

  The ASys research programme envisions the use of different views to define an autonomous system. This necessity has already been considered in the ASys Engineering Ontology by including the Perspective Package, which is specialised in the ASys Perspective Package, to represent the characterisation of a given autonomous system under the related viewpoint.

  The consideration of viewpoints relates closely to the different models to be developed and used by an autonomous system. To address the final chosen views, it might be necessary to partition the definition and the usage of the different concepts in OASys as related to these different viewpoints (structural, behavioural, functional, etc.). The utilisation of views will guide and ease the conceptual modelling process, considering different abstraction levels and the posible viewpoints.

- OASys evaluation and maintenance

  OASys has been developed following a methodology and considering the design criteria. However, ontological engineering is a task not exempt of issues. Starting from scratch, experts agreement on the terms, extendability, maintenance, integration, evaluation and so forth are problems that have arosen through the development of OASys.

  The research has paid special attention to the development activities of specification, and conceptualisation. It has also addressed the aspects of knowledge acquisition and integration of sources. Nonetheless, support activities such as the evaluation and the ontology maintenance are to be considered in the future use and development of OASys.

- The extension of the OASys–based Engineering Methodology

  The OASys–based Engineering Methodology is a proposal on how to follow an ontology–based engineering process, to suggest and to guide on the application of the OASys ontological elements for engineering any autonomous system. The approach to develop the OASys–based Engineering Methodology was made by tailoring, i.e., adapting existing software–related methodologies and engineering processes. In this sense, the most suitable features of each one of them were chosen to complete the OASys–based Methodology to address the ASys research programme.

  Nevertheless, the current stage of the ASys research programme has made it difficult to come up with a definite conceptualisation on how the engineering process of such systems is accomplished. Such engineering process is yet to be fully described, and hence conceptualised as part of the ASys Engineering Subontology of OASys. It is worthwhile pointing out, that this methodology was not conceived as a fully developed autonomous systems engineering process, which should address that the current system engineering practices could be applied to certain types of ASys, but some properties such as learning or being biologically inspired require additions or extensions.

- From the OASy–based Engineering Methodology to the ASys Modelling Methodology

  The OASys–based Methodology benefits from the underlying ontological commitments and relationships to build up the different models, ensuring against traditional meaning and conceptual mismatches during an autonomous system's development. The construction of the conceptual models has nonetheless been made ad–hoc, i.e., the ontological elements have been used and instantiated following a common sense approach without following a formal process.

  A proper methodology based on ontological and software patterns with the aid of conceptual modelling tools is a further step to enrich these conceptual models. This refinement process of concepts to address higher levels of details as well as the ontological elements usage is to be defined in the ASys Modelling Methodology, as a next stage in the overall ASys research programme. As part of this ASys Modelling Methodology, it is necessary to define among other aspects: how a concept is selected, how to integrate a concept into a pattern, how to establish and to import its relationships with other concepts, and how to detail or to add its attributes as part of the development of a concrete model.

10.4 Concluding remarks

For an autonomous system to behave appropriately in an uncertain environment, the system must have some kind of representation of what it feels and experiences as it perceives entities, events, and situations in the world. It would also be desirable for the system to have an internal model that captures what it knows and learns, as well as a mechanism to compute values and priorities that enables it to decide what objectives and goals to fulfill.

A major challenge in autonomous systems is the ability to maintain accurate representations of both the system and the environment in which it operates. The inability to do this might delay the effective task planning and execution. Furthermore, how these representations aid in the engineering process of defining the structure and behaviour of autonomous systems is worthwhile investigating. The tools, processes, and methodology used to express concepts related to a broader kind of autonomous systems have become critical, otherwise the engineering team will get lost in details.

Furthermore, the concepts used in the development of the autonomous systems will be useful for the systems themselves to be able to think about their inner workings. Concepts are used by an agent to think because they provide structure and methods of use of mental content. Concepts are used to perceive matching reality signal obtained through sensors to these concepts in mind. Concepts -ontologies- are used for cognitive interaction with other agents. Concepts are used by the engineer to think about the system being developed. Those very concepts can be put inside the system when it is a cognitive system to flesh-out its cognitive operation.

OASys expands the traditional role of common conceptualisation, to become an ontology that will be used at run time as an additional component co-operating with the autonomous system. OASys aims at providing the autonomous system itself with those same design–time concepts to be used as run–time knowledge in the form of models of itself. The aim is for any autonomous system to use its own design knowledge during its operation, captured in a machine-readable form such as an ontology: a set of concepts to be used by engineers in the description and construction of the system and by the system itself during its performance. The autonomous system will use models based upon the ontology developed in this dissertation [SHH$^+$09], as part of a model–based systems engineering strategy [SHG$^+$09]. A further line of research is how the conceptual models for the autonomous system will be generated from the OASys domain–ontology through a possible use of conceptual modelling tools, identified patterns, and building blocks library.

An autonomous system will perform using models of its environment, of its task and of itself to operate, as in the model-based control paradigm, where those models will be the same ones used by the engineers to build the system. This will ultimately provide the system with self-engineering capabilities required for robust autonomy [SLH07]. A further step will be for the agents to interpret those models based on OASys. Ontologies have been widely used within the agent community [BAS06], but it is yet to investigate how the ontology and related models can be used to generate meaning for the decision–making process of an autonomous system [SBE$^+$03], [SBLG08].

Chapter 11

OASYS GLOSSARY OF TERMS

This annex gathers the different ontology elements part of OASys. The ontology elements within each subontology are described following this format:

Name [Synonym] Acronym: Description (List of Sources)

where:

Name: the name given to the ontology element, following naming conventions described in Sec. 7.3.2.5.

Synonyms: a list of existing synonyms for the ontology element, if any.

Acronyms: short–form name for the ontology element, if any.

Description: textual definition of the ontology element.

Source: document(s) where the ontology element was found, if any.

11.1 ASys Ontology

The ASys Ontology provides, at different levels of abstraction, the ontological constructs necessary to describe and characterise an autonomous system. It comprises two subontologies: the System Subontology, and the ASys Subontology.

11.1.1 System Subontology

The System Subontology provides the concepts and relationships to the higher level of abstraction within OASys, to be used when describing any system.

11.1.1.1 General Systems Package

The General Systems package provides the concepts, relations and axioms related to General Systems Theory, as a theory developed to describe any kind of systems.

1. Concepts
   - **AbsoluteRelation**: relation satisfied over the entire time interval of every possible particular activity containing the quantities at the resolution level ([Kli69])
   - **Activity**: variations in time of all the quantities under consideration at the given resolution level ([Kli69])
   - **Backdrop**: any underlying property (such as time, space) that is actually used to distinguish different observations of the same property ([KE03])
   - **Behaviour**: a particular time-invariant relation specified for a set of quantities and a resolution level ([Kli69], [KE03])
   - **CompleteProgram**: the instantaneous state of the system, a set of some other states, and a set of transitions from the instantaneous state to the states under consideration in time ([Kli69])
   - **Coupling**: the set of all common external quantities between two elements; influence or relationship between two systems or elements; shared variable among elements which represent interactions between the elements ([Kli69], [KE03])
   - **Element [ElementarySubsystem]**: subsystem that cannot be further decomposed into subsystems ([MBW$^+$08])
   - **Environment**: system whose elements are not contained in the universe of the given system; it could be internal or external ([Kli69])
   - **ExternalQuantity**: observed quantity of the system ([Kli69])
   - **HypotheticCoupling**: type of coupling valid anywhere within a particular activity of the system ([Kli69])
   - **HypotheticStructure**: the structure forming the basis of the relatively permanent behaviour ([Kli69])
   - **InstantaneousProgram**: instantaneous state plus the transitions from this system state ([Kli69])
   - **InternalQuantity**: quantity that cannot be observed, but play a mediatory part ([Kli69])
   - **LocalRelation**: relation which apply only within some shorter time intervals of a particular activity ([Kli69])
   - **Organisation**: properties possessed by a system showing a particular behaviour ([Kli69])
   - **PermanentBehaviour** RealBehaviour: type of behaviour defined by the set of all local relations (the absolute relation) ([Kli69])
   - **Program**: the variable part of the organization of a system/ the instantaneous state of the system, a set of some other states, and a set of transitions from the instantaneous state to the states under consideration in time ([Kli69])

- **Property**: feature that characterize the studied phenomenon of the object ([KE03])
- **Quantity [Variable]**: an abstract image or operational representation of an attribute ([Kli69])
- **RealCoupling**: type of coupling valid over the entire time interval of anay activity of the system ([Kli69])
- **RealStructure**: the structure forming the basis of the permanent behaviour ([Kli69])
- **RelativelyPermanentBehaviour [KnownBehaviour]**: type of behaviour defined by the set of all local relations of a particular activity (the relative relation) ([Kli69])
- **RelativeRelation**: relation satisfied anywhere within a particular activity containing the quantities at the resolution level ([Kli69])
- **ResolutionLevel**: the determination of sets of those values of all the observed or given quantities to consider, together with a set of those time instants at which the corresponding values can be or want to be obtained ([Kli69])
- **Space**: a type of backdrop used to measure system's quantities based on space locations concept ([Kli69], [KE03])
- **Structure**: the constant part of the organisation of a system ([Kli69])
- **StateTransitionStructure [STStructure]**:type of structure defined by the complete set of states, plus the complete set of transitions between states ([Kli69])
- **Subprogram**: instantaneous state plus a nonempty subset of the set of all other states of the system, and a noempty subset of the set of all transitions from the inst. state to all the states under consideration ([Kli69])
- **Subsystem**: it is a system that is constituent of another system ([KE03], [KE03] , [MBW+08])
- **Support**: an operational representation of a backdrop ([KE03])
- **SupportInvariantRelation [TimeInvariantRelation]**: any relation between quantities that is satisfied within a time interval ([Kli69]
- **System**: abstraction of some aspects of the object under study ([Kli91])
- **SystemState**: the set of instantaneous values of all the quantities of the system (both external and internal) ([Kli69])
- **TemporaryBehaviour**: type of behaviour defined by the local relation ([Kli69])
- **Time**: a type of backdrop used to measure system's quantities based on time instants ([Kli69])
- **Transition**: admissible changes of the value of a quantity ([Kli69])
- **UniverseofDiscourseandCouplingsStructure [UCStructure]**: type of structure consisting of the set of all the elements of a system, their permanent (or relatively permanent) behaviour, and the set of aboslute (or relative) couplings ([Kli69])

2. Attributes
    - **quantityName**: label of a variable to distinguish it from others ([KE03])
    - **quantityUnit**: unit in which the quantity's value is measured
    - **quantityValue**: state of a variable ([Kli69], [KE03])
    - **supportInstance**: element of the support set attribute ([KE03])
    - **supportName**: name of a support to distinguish it from the rest ([KE03])

3. Relations
    - **accordingTo**: a backdrop is measured according to a given resolution level
    - **assignedTo [hasValue]** : a quantity value is assigned to a particular property ([MBW+08])
    - **changeThrough**: system's state change following defined transitions
    - **characterisedBy [hasProperty]**: a system is characterised by a set of properties relation ([MBW+08])
    - **exhibits**: a system exhibits a behaviour
    - **measuredUsing** isObservedAgainstBackdrop: a quantity is measured using a particular backdrop (time, space, time-space) ([MBW+08])
    - **placedIn**: a system is placed in a particular environment
    - **relatedIn**: system's quantities are related in the form of a support invariant relation
    - **varyInTimeAs**: quantities vary in time as an activity

4. Axioms
    - A system interacts with, or is related to, other systems
    - A system consists of subsystems and elements, which are themselves other systems
    - A system is placed in an environment, from which it can be separated by a conceptual or physical boundary
    - A system has properties, described as quantities that have a value

11.1.1.2 Mereology Package

The Mereology package provides the concepts and relationships related to mereological and meronymic aspects, i.e. part–whole relationships.

1. Concepts
    - **Object**: entity (physical or abstract) that exist in an application domain ([MWM07])
    - **Aggregate**: an object that has one or more distinct parts ([MWM07])

- **Composite**: an object that is composed of one or more objects, which are non-shareable ([MWM07])
- **Part [Individual]**: an object that is part of another object ([MWM07], [Bor97])
- **ParOfComposite**: an object that is part of a composite ([MWM07])
- **PartOnly [SimpleIndividual]**: an object that is part of another object, without parts of its own concept ([MWM07], [Bor97])
- **DirectPart**: an object that is part of another object at the next level of decomposition ([MWM07])

2. Relations
   - **equal**: an object is equals to another object, if they are the same ([Bor97])
   - **disjoint**: two objects do not have common parts ([Bor97])
   - **isPartOf [partOf]**: a part is related to an aggregate ([Bor97], [MWM07])
   - **isDirectPartOf [properPartOf]**: a part is related to an aggregate at the next level of decomposition ([Bor97], [MWM07])
   - **isExclusivelyPartOf**: a part of composite is related to a composite ([MWM07])
   - **isSpatialPartOf**: a part is related to a whole taking into account space or location ([KA07])
   - **isContainedIn** : type of isSpatialPartOf where a part is in a 3D containment ([KA07])
   - **isLocatedIn**: type of isSpatialPartOf where a part is in a 2D containment ([KA07])
   - **isInvolvedIn**: a part is related to a whole being both processes or events ([KA07])
   - **isMemberOf**: meronymic relation that relates a member to a bunch or collection is member of ([KA07], [Gui05])
   - **isMadeOf**: meronymic relation between an object and the material is made of ([KA07], [Gui05])
   - **isSubquantityOf**: meronymic relation between a smaller part-amoun of matter to a whole-matter, being of the same or similar type of stuff ([KA07], [Gui05])
   - **participatesIn**: meronymic relation that relates an object with the process or event it participates in ([KA07], [Gui05])

3. Axioms
   - An aggregate has at least one part
   - A composite is composed of at least one part of composite
   - A part is part of at least one aggregate
   - A part of composite is exclusively part of a composite
   - A part of composite cannot be shared
   - A part only cannot be an aggregate or a composite

- A part is not part of itself

These axioms are similarly applied at system's level considering the concepts of system, subsystem, and elements as described in Sec. 11.1.1.1.

11.1.1.3 Topology Package

The Topology package provide topological (connection among elements) concepts and relations existing in a system.

1. Relations
   - **isConnectedTo**: an object is connected to another object [MWM07]. Applied to systems, subsystems and elements at system's level.
   - **isDirectlyConnectedTo**: a non-transitive specialization of isConnectedTo [MWM07]. Applied to systems, subsystems and elements at system's level.

11.1.2 ASys Subontology

The ASys Subontology provides the ontological elements for the domain of (cognitive) autonomous system, by addressing autonomous systems' structure, behaviour and function.

11.1.2.1 Device Package

The Device package provides the concepts to describe the different devices, and their properties, that will be part of an autonomous system, organised in packages (ComputationalDevice and PhysicalDevice).

1. Concepts
   - **PowerSupply [Power]**: explanation of the power supply for a device ([HH08], [dlM09b])
   - **Artifact [SoftwareArtifact]**: the specification of a physical piece of information that execute or is used in a system's software. It is installed or deployed on a node. Examples are executable, library, source and configuratin files ([RJB04])
   - **ASLabDevice [Hardware]**: a type of node that represents a physical noncomputational unit or resource, with sensing, acting and other capability. ASLabDevices may be complex consisting of other (ASLab)Devices ([RJB04])

- **Component [Module]**: a type of node that represents an encapsulated, reusable, modular and replaceable part of a system's software whose behaviour is defined by interfaces ([RJB04])
- **Device [ComputationalHardware]**: a type of node that represents a physical computational unit or resource, with processing capability upon which artifacts may be deployed for execution. Devices may be complex, i.e consisting of other (ASLab)Devices ([RJB04])
- **Diameter**: a type of dimension ([HH08], [dlM09b])
- **Dimension**: a type of physical characteristic to show the size of a device, being it diametre, length, height, width, volume ([HH08], [dlM09b])
- **ExecutionEnvironment**: a type of node that offers an execution platform for specific types of components that are deployed on it in the form of executable artifacts ([RJB04])
- **Height**: a type of dimension ([HH08], [dlM09b])
- **Interface**:interaction among component. It could be ProvidedInterface (an interface a component realizes) or RequiredInterface (an interface a component needs) ([RJB04])
- **Length**: a type of dimension ([HH08], [dlM09b])
- **Material** : a type of physical characteristic to show the different substances a device can be made of ([HH08], [dlM09b])
- **Node**: a hardware or software resource that can host software or related files, upon which artifacts may be deployed for execution ([RJB04])
- **PhysicalCharacteristic**: physical feature of a device, such as dimensions, etc ([HH08], [dlM09b])
- **ProvidedInterface**: type of interface provided by a component ([RJB04])
- **RequiredInterface**: type of interface required by a component ([RJB04])
- **Volume**: a type of dimension ([HH08], [dlM09b])
- **Weight**: a type of physical characteristic to show the device's quantity of matter ([HH08], [dlM09b])
- **Width**: a type of dimension ([HH08], [dlM09b])

2. Attributes
   - **ASLabDescription**: a sentence describing the device ([HH08], [dlM09b])
   - **ASLabIdentifier**: name of device to identify it in documents ([HH08], [dlM09b])
   - **ASLAbName** : name of device to identify it orally, in a colloquial way ([HH08], [dlM09b])
   - **deviceManufacturer**: company from which the device was acquired ([HH08], [dlM09b])
   - **deviceModel**: model of device ([HH08], [dlM09b])
   - **deviceStatus** : situation of a device, such as available or fully operative ([HH08],[dlM09b])
   - **purchaseDate**: date when the device was purchased ([HH08], [dlM09b])

- **serialNumber** : device number provided by manufacturer ([HH08], [dlM09b])

11.1.2.2 Perception Package

The Perception package gathers concepts and relationships related to the perception processes that take place in an autonomous system.

1. Concepts
   - **CognitiveEquivalence**: equivalence between the distal stimulation and the referent, parting from singularities ([Lóp07])
   - **DirectedProcessing [CognitiveInformationProcessing, Directed-Perception] DP**: second phase of the perceptual process ([Lóp07], [UIT06e], [LSB07])
   - **DistalStimulation [DistalStimulus]** : type of stimulation by objects and events in the outside world ([Lóp07], [UIT06e], [LSB07])
   - **EnvironmentalCorrelation**: correspondence between the distal stimulation and the proximal stimulation ([Lóp07], [UIT06e], [LSB07])
   - **EnvironmentObject** : entity which the system perceives, senses, and interacts with ([DV07])
   - **ImplicitChange**: changes into the system, due to the dependence between the perceptor and the rest of the observer system ([Lóp07])
   - **ImplicitCoupling**: coupling between the perceptor and the perceptive environment, as part of percept ([Lóp07], [UIT06e], [LSB07])
   - **MarginalCoupling** ImplicitCoupling: type of implicit coupling related to the system's environment, as part of the perceptor ([Lóp07], [UIT06e], [LSB07])
   - **ObserverSystem**: system that perceives entities in the environment ([Lóp07], [UIT06e], [LSB07])
   - **PerceivedObject [InstantiatedReferent]**: instantiation of a referent/ a representation of a particular state of a referent which is recognised in the environment ([Lóp07], [LSB07])
   - **Percept**: changes in the perceptor as a result of perception. Combination of implicit changes and representation ([Lóp07])
   - **PerceptiveProcess [PerceptualProcess]**: Process by which a relation between the perceptor and its perceptive environment is established, by relating the proximal stimulation, the singularities and the environment object. Consists of two phases (sensory processing, and directed processing) ([Lóp07], [UIT06e], [LSB07])
   - **Perceptor PR** : perceiving element of an observer system; part of an (observer) system which carries out perception at a certain instant [Lóp07]
   - **PerceptiveEnvironment** : part of the environment upon which the perception process takes place. Includes the outside of the observer system, the system environment and parts of the system itself ([Lóp07], [UIT06e], [LSB07])

- **ProximalStimulation [ProximalStimulus, SensoryInput, Measurement, Reading]** : type of stimulation found at the receptors in the eye ([Lóp07], [UIT06e], [LSB07])
- **EnvironmentObject [PhysicalObject]**: entity which the system perceives, senses, and interacts with ([DV07])
- **Referent [PerceptiveReferent]**: conceptual entities to which perception is referred to ([Lóp07], [LSB07])
- **Representation [ConceptualRepresentation, PerceptualRepresentation]**: output of the perceptive process, as a part of the percept ([Lóp07], [UIT06e], [LSB07])
- **RepresentationSystem**: part of a perceptor that provides the representation as output of the perceptive process ([Lóp07], [UIT06e], [LSB07])
- **SensoryProcessing [ProximalInformationProcessing, SensoryPerception] SP**: first phase of the perceptive process ([Lóp07])
- **SensorySystem**: part of a perceptor that provides the input to the perceptive process ([Lóp07], [UIT06e], [LSB07])
- **Singularity [Feature, Invariant, Cue]**: pattern in the values of the proximal stimulus ([Lóp07], [UIT06e], [LSB07])
- **SubstratalCoupling** : type of implicit coupling related to the rest of the system, as part of the perceptor ([Lóp07], [UIT06e], [LSB07])

2. Relations
   - **analyses**: the perceptive process (sensory processing phase) considers the singularities
   - **becomes**: the distal stimulation is transformed into the proximal stimulation
   - **carriesOut**: the perceptor of the observer system performs the perceptive process
   - **isInputTo**: the output of the sensory processing phase is used as input to the directed processing phase
   - **isInstantiated** : a referent is instantiated into an instantiated referent or perceived object
   - **obtains**: the perceptive process (directed processing phase) obtains the representation
   - **recognises**: the perceptive process (directed processing phase) recognises the referents in the perceptive environment
   - **takesPlaceIn**: the perceptive process takes place in the perceptive environment
   - **transformedIn** : the proximal stimulation is transformed in a representation

3. Attributes
   - **perceptorName**: name of the perceptor
   - **referentName** : name of the referent

- **singularityName**: name of the singularity

11.1.2.3 Knowledge Package

The Knowledge package provides the concepts and relationships related to knowledge assets used by an ASys during its thinking, and evaluating operations, organised in different packages (Quantity, Model, Ontology, Goal, Directiveness and Procedure).

1. Concepts
   - **AbstractGoalForm [AbstractGoal, InactiveGoal]**: the goal does not determine the system's behaviour, being inactive ([UIT06d])
   - **AbstractQuantity [IntrinsicalyCognitiveQuantity]**: conceptual quantity that does not relate to an actual physical quantity ([HSL08], [Lóp07])
   - **Algorithm [FunctionalContent]**: conceptual specification of a function; specification of a particular way of realizing a function (of a goal) ([UIT06d], [UIT06b])
   - **CodedGoal [AbstractGoalForm]**: abstract form of a goal that refers to actual quantities in a system. It can become active when instantiated ([UIT06d])
   - **CognitiveModel [SelfModel]**: system model consisting in the set of instantiated quantities and its instantiated organization ([UIT06d])
   - **ConceptualQuantity [CognitiveQuantity]**: quantity that is a specific resource of the system whose state represents the state of a different part of the universe ([HSL08], [Lóp07])
   - **DomainOntology**: ontology that describes reusable concepts within a domain ([GPFLC04b])
   - **Essence** : abstract form of a goal that does not refer to actual quantities in a system. It can become active when coded ([UIT06d])
   - **Goal**: a state (or result) to be achieved, maintained or optimised——specific restriction of the system properties that a cognitive agent dealing with the system considers desirable under given circumstances / a prescriptive statement of intent about some system (existing or to-be) whose satisfaction in general requires the cooperation of some agents or elements forming it ([Kli91], [UIT06d])
   - **GoalForm**: situation or status in which the goal can be (abstract form, real form, essence, coded, instantiated) ([UIT06d])
   - **GoalFunction** : sucession of a system's states associated to a particular goal ([UIT06d])
   - **GoalStructure [GoalHierarchy]**: hierarchical structure of dependence among the set of goals of a system (from higher goal to local goals through intermediate goals) ([UIT06d])

- **InstantiatedGoal**: a goal in real form is instantiated into the system's quantities ([UIT06d])
- **InstantiatedQuantity**: conceptual quantity that relates to a real current physical quantitiy ([HSL08])
- **Knowledge**: an internalised information structure that is isomorphic from a certain useful perspective with some portion of reality ([SR08])
- **LocalGoal**: goal at the lowest level of the goal structure ([UIT06d])
- **Mission [DesignGoal]**: goal to which a system directs its behaviour ([UIT06d])
- **Model**: representation of a system (where system is an extended notion of any element, including the environment or world) ([Tea])
- **OASys**: ontology that describes concepts related to autonomous systems ([SR08])
- **Ontology**: a formal, explicit specification of a shared conceptualization ([SBF98])
- **PhysicalModel**: system model consisting in the set of system physical quantities and its organization ([UIT06d])
- **PhysicalQuantity**: type of quantity as a result of perception and sensing ([UIT06d])
- **PotentiallyInstantiatedQuantity**: conceptual quantity that are not related to a real current physical quantitiy, but which could be ([HSL08], [Lóp07])
- **RealGoalForm [ActivatedGoal, ActiveGoal]**: the goal determines the system's behaviour, being activated ([UIT06d])
- **RootGoal [GenerativeGoal]**:l goal at the higher level of the goal structure ([UIT06d])
- **Subgoal [IntermediateGoal]**: any lower-level goal related to one; goal at medium level of the goal structure that allows to realice root goals ([UIT06d])
- **SystemFunction**: desired behaviour of a system, as an abstraction of the actual behaviour —— a role played by a behavior specified in a context ([MBW$^+$08], [Miz08])
- **SystemModel** : a system's internal representation of itself ([SR08])
- **UpperOntology**: ontology that describes very general concepts that are common across domains ([GPFLC04b])
- **WorldModel [EnvironmentModel]**: a system's internal representation of the world. It may contain models of portions of the environment, of agents, of objects ([SR08])

2. Relations

   - **conceptualises**: an algorithm is a conceptualization of a goal function
   - **isCoded** : an essence goal can be or not coded
   - **isInA**: a goal can be in a particular goal form (abstract, real, etc)

- **uses**: OASys uses some concepts from upper ontologies

3. Attributes
   - **algorithmName**: name of an algorithm
   - **functionName**: name of a goal function
   - **levelOfAbstractio**: attribute of a goal to specify if it is a root goal, a subgoal at different levels, or a local goal ([UIT06d])
   - **goalName**: name of the goal ([UIT06d])
   - **order [goalLevel]**:a parameter to evaluate the relevance of a specific goal within the goal structure (RootGoal = 1, LocalGoal = 0, Subgoal in [0,1) range) ([UIT06d])
   - **priority**: a parameter to determine the relevance of a goal within a group of same level goals [UIT06d]
   - **timeScope [temporalScope]**: the time necessary for the system to realice or carry out a goal (the duration of the period during which the goal directs the system's behaviour) ([UIT06d])

11.1.2.4 Thought Package

The Thought package provides the ontological elements to characterise thinking as any knowledge–based abstract processing where knowledge refers to the concepts defined in the Knowledge Package.

1. Concepts
   - **AlgorithmGeneration**: phase of functional decomposition where an algorithm is generated ([UIT06d])
   - **AlgorithmGrounding**: phase of functional decomposition, where a grounded function is obtained (as a result of grounding an algorithm) ([UIT06d])
   - **AlgorithmSelection** : phase of functional decomposition where a particular algorithm is selected from available ones ([UIT06d])
   - **CodeGeneration**: type of goal generation where the abstract idea represented by the essence of a goal is coded ([UIT06b])
   - **EssenceGeneration**: expression of a solution to a problem regardless of its pertenance to an specific system ([UIT06b])
   - **FunctionalDecomposition**: process of mapping the goal structure into the functional structure ([UIT06d])
   - **FunctionalStructure**: topology of dependencies and hierarchies among the grounded functions of a system ([UIT06d], [UIT06b])
   - **GoalActivation**: phase of goal life cycle where a goal in abstract form is instantiated, by adapting the organization of the system to the goal ([UIT06d])
   - **GoalActivity**: the evolution of the system during the time when the goal is instantiated ([UIT06d])

- **GoalConclusion**: phase of goal life cycle where the goal's activity concludes as the goal is reached ([UIT06d])
- **GoalDeactivation**: phase of goal life cycle where the goal is deactivated by the instantiation of a new one ([UIT06d])
- **GoalGeneration**: phase of goal life cycle where the essence, the code (or both) of a goal is created ([UIT06d])
- **GoalLifecycle**: sequence of goal lifecycle phases (generation, activation, activity, conclusion or deactivation) ([UIT06d])
- **GoalReconfiguration**: rearrangement of goal definition and structure to adapt systems behaviour for disturbances ([UIT06d])
- **GoalSelection**: phase of goal reconfiguration ([UIT06d])
- **GroundedGoalFunction**: a goal function grounded considering the system's real state and resources ([UIT06d])

2. Relations
   - **ends** : the activity of a goal ends in the goal conclusion phase
   - **deactivates** : the goal deactivation deactivates a goal activity
   - **hasAsResult** : the grounding of an algorithm obtains a grounded goal function

11.1.2.5 Action Package

The Action package provides the concepts related to the different activities and entities involved in the operations of an autonomous system.

1. Concepts
   - **Action**: type of interaction that establishes a relation between the physical quantity in the system with the sensed physical quantity in the environment; set of actor motion or behaviour, developed by agents of the system and sensed by the system as changes in the environment ([HSL08], [Lóp07],)
   - **Actor**: any object that observes and can act upon its environment basing that action in a model of the environment directing this action towards achieving a goal ([Tea])
   - **Actuation**: type of operation where there is a mapping of a conceptual quantity with a physical quantity in the system (or viceversa). It refers to perception, grounding and embodiment ([Lóp07], [HSL08])
   - **Actuator** : device or set of devices which transform an electrical signal into a physical magnitude ([Lóp07])
   - **Agent**: type of actor that can sense, reason and is inteded to act ([Lóp07])

- **ConceptualOperation [CognitiveOperation]** : type of operation carried out by the cognitive part of a system with conceptual quantities ([HSL08], [Lóp07],)
- **Controller**: ( )
- **ControlLoop**: ( )
- **Embodiment**: type of actuation that establishes a relation between the conceptual quantity and the physical quantity that supports it, in which it is embodied ([HSL08])
- **Engine [ActionGenerator]**: ([Tea])
- **Evaluator [Predictor]**: ([Tea])
- **GroundedOperation**: type of operation carried out by the physical part of a system with system's physical quantities ([Lóp07])
- **Grounding**: mapping of conceptual quantities to physical quantities in the system ([Lóp07], [HSL08])
- **Human**: type of Actor to refer to human operators interacting with the system
- **HWAgent**: type of Agent to refer to hardware elements
- **InstantiatedOperation**: type of conceptual operation carried out with instantiated quantities ([Lóp07])
- **Interaction**: type of operation where there is a mapping of physical quantity in the system with a physical quantity in the environment (or viceversa). It refers to sensing and action ([HSL08])
- **NonInstantiatedOperation**: type of conceptual operation carried out with non instantiated quantities ([Lóp07])
- **Operation**: task or process carried out by an actor. It could be conceptual (in the cognitive system) or grounded (in the physical system) or interaction (between cognitive-physical, or physical-environment) ([HSL08])
- **Perception**: process by which a relation between the perceptor and its perceptive environment is established, which translates from system's physical state to mental state ; mapping of a physical quantity with a conceptual quantity in the cognitive subsystem ([Lóp07], [HSL08])
- **Sensor**: element of the sensory system, that dectects, measures or records physical phenomena ([Lóp07], [SHR07], [LSB07])
- **Sensing** : type of interaction where a physical quantity in the system's environment is mapped with a physical quantity in the system ([Tea], [HSL08])
- **SWAgent**: type of Agent to refer to software–based elements

2. Relations

   - **commands**: the control loop commands the actor to perform an operation

11.2 ASys Engineering Ontology

The ASys Engineering Ontology provides the system's engineering related ontological elements. It comprises two subontologies: the System Engineering Subontology gathers the concepts related to general systems' engineering, whereas the ASys Engineering Subontology gathers in a similar way the elements for ASys engineering.

11.2.1 System Engineering Subontology

The System Engineering subontology provides the ontological elements for process and software system's engineering. The subontology contains different packages to address different aspects to consider, as needed, within a system's development: requirements, engineering process, system perspective, model–driven.

11.2.1.1 Requirement Package

The Requirement package provides the ontological elements to express the system's requirements which are usually established during a system's engineering process.

1. Concepts
   - **UseCaseActor**: an object outside the scope of the system that has significant interaction with it ([Obj09b])
   - **Requirement**: a prescriptive statement of intent about some system (as is or to be) whose satisfaction requires the cooperation of some of the elements and agents in the system; a capability or condition that must (or should) be satisfied by a system [IEE90], ([Gro08a])
   - **Subject**: system, subsystem under consideration to which the use cases apply ([Obj09b])
   - **UseCase**: mean to capture a requirement of a system ([Obj09b])
   - **DesignRequiment**: requirement that specifies a constraint on the implementation of the system ([IEE90], [Gro08a])
   - **FunctionalRequirement**: requirement that specifies an operation or behaviour tht a system must perform ([IEE90], [Gro08a])
   - **InterfaceRequirement**: requirement that specifies the ports for connecting systems and elements ([IEE90], ([Gro08a])
   - **PerformanceRequirement**: requirement that quantitatively measures the extent to which a system satisfies a required capability ([IEE90], [Gro08a])
   - **PhysicalRequirement**: requirement that specifies physical characteristics and/or physical constraints ([IEE90], [Gro08a])
   - **TestCase**: a test case is a method for verifying a requirement is satisfied ([Gro08a])

2. Relations
    - **appliedTo**: Relation between a use case and the subject is applied to
    - **derive**: Dependency between two requirements in which a client requirement can be derived from the supplier requirement ([Gro08a])
    - **extend**: Relation that specifies that a use case can be extended by another one ([Obj09b])
    - **include**: Relation that specifies that a use case includes another use case ([Obj09b])
    - **organises**: Relation that specifies that a use case organises subject's requirements
    - **satisfy**: Relation between a requirement and an element that fulfills the requirement ([Gro08a])
    - **participatesIn** : Relation between a use case actor and the use case is related with ([Gro08a])
    - **verify**: Relation between a requirement and an element that can determine whether the system fulfills the requirement ([Gro08a])

3. Attributes
    - **useCaseActorName**: Identifier of the actor in a use case ([Obj09b])
    - **requirementIdentifier**: Unique identifier for a requirement ([Gro08a])
    - **risk**: Level of risk of the requirement. It could be high (unacceptable level of risk), medium (acceptable level) and low (minimal level of risk or no risk) ([Gro08a])
    - **source**: Origin of requirement, such as a stakeholder ([Gro08a])
    - **subjectName**: Identifier of the subject ([Gro08a])
    - **testCaseName** : Identifier of the test case that verifies a use case ([Gro08a])
    - **text**: Text description of a requirement ([Gro08a])
    - **useCaseName**: Identifier of the use case ([Obj09b])
    - **verifyMethod**: Method to verify the requirement, such analysis, demonstration, inspection, test ([Gro08a])

4. Axioms and Rules
    - A UseCase has an unique useCaseName
    - A UseCaseActor has an unique useCaseActorName
    - A TestCase has an unique testCaseName

11.2.1.2 Engineering Process Package

The Engineering Process Package provides general concepts to describe the stages and steps in a system's engineering process.

1. Concepts
    - **Constraint**: scenario, situation or other aspect which limits the realisation of a task (adapted from [Lóp07])
    - **Guidance**: description that provides additional information on how to perform a particular task or grouping of tasks, or that provides additional detail, rules, and recommendations on work products and their properties ([Gro08b])
    - **Phase**: type of activity that describe a basic units of work within a Process // a significant period in a project, ending with major management checkpoint, milestone, or set of work products ([Gro08b])
    - **Process [Lifecycle]** : a complete description of a system's engineering process, in term of a sequence of Phases that achieve a specific purpose, Performers, Roles, Work Products, and associated Guidance ([Gro08b])
    - **Resource**: part of a system dedicated to the realization of a task (adapted from [Lóp07])
    - **Role**: an individual carrying out tasks, and responsible for certain work product ([Gro08b])
    - **Subtask** : used to organize a task into parts or subunits of work ([Gro08b])
    - **Task [Step]**: A Work Definition describing what a Process Role performs. Activities are the main ([Gro08b])
    - **WorkProduct [Product]**: a description of a piece of information or physical entity produced or used by the activities of the software engineering process. Examples of work products include models, plans, code, executables, documents, databases, and so on ([Gro08b])

2. Relations
    - **additional**: relation to express that a role is additional in an activity ([Gro08b])
    - **allocatedTo**: a specific resource is allocated to a task realisation ([MHHB94])
    - **controls**: a constraint controls a task realisation [MHHB94]
    - **guides**: a type of guidance serves as a guide for the process ([Gro08b])
    - **input**: to express that a work product is required as input for a task ([Gro08b])
    - **manages**: a role manages resources [MHHB94]
    - **mandatory**: to express that a task is mandatory within an activity ([Gro08b])
    - **optional**: to express that a task can be optional within an activity ([Gro08b])
    - **output**: to express that a work product is obtained as output in a task ([Gro08b])
    - **performs**: relationship of a work performer to a work ([Gro08b])
    - **postcondition**: any kind of constraint that must evaluate to true before the work can be declared completed or finished ([Gro08b])

- **precondition**: any kind of constraint that must evaluate to true before the work can start ([Gro08b])
- **predecessor**: to express that an activity is predecessor of another one in a sequence ([Gro08b])
- **primary**: to express that a role is primary in an activity ([Gro08b])
- **responsible**: a role is responsible for an activity ([Gro08b])

3. Axioms
   - Every Taks is owned by a Role ([Gro08b])
   - Lifecycle only contains Phases ([Gro08b])
   - One Task can produce one or more WorkProducts ([Gro08b])
   - One WorkProduct can be specified through one or more Artifacts ([Gro08b])

11.2.1.3 Perspective Package

The Perspective package gathers the ontology elements necessary to consider the different possible views, and viewpoints when developing any system's architecture.

1. Concepts
   - **Concern** : Those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders ([Ins00])
   - **EngineeringModel**: An abstraction of a system, specifying it from a certain viewpoint for a category of system stakeholder (e.g. designer, user, etc) ([Ins00])
   - **Mission**: A use for which a system is intended by one or more stakeholders to meet some set of goals ([Ins00])
   - **SystemStakeholder**: An individual, team or organisatin with interests in, or concerns, relative to, a system ([Ins00])
   - **View**: A representation of a whole system from the perspective of a related set of concerns ([Ins00])
   - **Viewpoint [Perspective, Aspect]**: A specification of the conventions for constructing and using a view ([Ins00])

2. Relations
   - **conformsTo**: relation between a viewpoint an a view, where a viewpoint conforms to a particular view ([Ins00])
   - **consistsOf**: relation between a view and the different engineering models it consists of ([Ins00])
   - **fulfills**: relation between a system designed to fulfill a mission ([Ins00])

- **identifies**: relation between a stakeholder and a concern, where a stakeholder identifies a particular concern ([Ins00])
- **selects**: relation between a system stakeholder who chooses a particular viewpoint ([Ins00])
- **usedToCover**: relation between a viewpoint and a concern, where a viewpoint is used to cover a particular concern ([Ins00])

3. Attributes
   - **viewpointName**: name given to the viewpoint ([Ins00])
   - **purpose**: the reason for considering such viewpoint ([Ins00], [FMS08])
   - **viewpointConcern**: the concern the stakeholders wish to adress in the viewpoint ([Ins00], [FMS08])
   - **viewpointStakeholder**: the stakeholders who have an interest in this viewpoint ([Ins00], [FMS08])
   - **language**: the languages used to present the view ([Ins00], [FMS08])
   - **method**: the methods used to establish the view ([Ins00], [FMS08])

4. Axioms
   - Each View corresponds exactly with one Viewpoint ([Ins00])
   - A Concern drives a Viewpoint selection ([Ins00])
   - A View addresses one or more Concerns ([Ins00])

11.2.1.4 Modeldriven Package

The Modeldriven Package provides fundamental concepts and relations related to model–driven (software) engineering (MDSD), which refers to a range of development approaches that are based on the use of (software) modeling as a primary form of expression.

1. Concepts
   - **AbstractSyntax**: specification of the language's structure ([SV06])
   - **ConcreteSyntax**: specification of the language's concrete syntax ([SV06])
   - **Domain**: a bounded field of interest ([SV06])
   - **DomainArchitecture**: a specification of the metamodel, a platform, and transformations ([Obj03])
   - **DomainSpecificLanguage [ModellingLanguage] DSL**: language to express and describe formally the relevant contents of a domain ([SV06])
   - **FormalModel**: model expressed in the DLS's concrete syntax, based on DLS's semantics ([SV06])
   - **Metamodel**: the structure of a domain which comprises the abstract syntax and the static semantics of a language([SV06])

- **ModelTransformation**: the process of converting one model to another model of the same system([SV06])
- **Platform** : a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns ([SV06])
- **StaticSemantic**: criteria for the well-formedness of a language ([SV06])
- **Subdomain**: smaller part of a domain ([SV06])

2. Relations
   - **describes**: a metamodel describes a domain ([SV06])
   - **getsMeaningFrom**: a formal model get its meaning from semantics ([SV06])
   - **specifiedBasedOn**: a formal model is specified based on the concrete syntax ([SV06])
   - **specifiedUsing**: the abstract syntax is specified using the static semantics ([SV06])
   - **supports**: a platforms supports a domain ([SV06])

11.2.2 ASys Engineering Subontology

The ASys Engineering Subontology is the ontology which addresses autonomous systems' engineering process, by gathering concepts and relationships to describe a generic engineering process or lifecycle for an autonomous system. It is a refinement or specialisation of the concepts already defined in the System Engineering Subontology.

11.2.2.1 ASys Requirement Package

The ASys Requirement package provides the concepts to define an autonomous system's requirements, either as new concepts or as specialisation of the System Requirement Package which defined requirements at a higher level of abstraction.

1. Concepts
   - **Accesibility**: the degree to which a system can be accessed to its parts and workings for the different purposes of long term operation [SR08]
   - **Adaptivity**: capacity of adaptation (either algorithms, functional structure, etc). Element of system's autonomy [Lóp07]
   - **Adequacy**: the capacity of a system for some particular purpose or application [UIT06d]

- **Autonomy**: ability to generate one's own purposes without any instructions from outside. Decisional if referred to goals; Operational if refered to operation. / The quality of a system of behaving independently while pursuing the goals it was commanded to. [HSL08]
- **Availability**: the degree to which a system is operable and in a committable state at the start of a mission. Measure of the time a system is in a functioning condition [SR08], [HSL08]
- **Constructability**: the degree to which it requires effort to build the system [SR08]
- **Dependability** : the capacity of a system of being dependent, in terms of availability, safety, reliability, maintainability, etc [SR08], [HSL08]
- **Integrability**: the degree to which a system can be part of a bigger system [SR08]
- **Maintainability**: the degree to which a system can be kept into operation, by solving new or old problems [SR08], [HSL08]
- **Modularity**:feature of a system to be structured in modules [UIT06d]
- **Performance**: the closeness between the actual and desirable manifestations of those system properties involved in a goal —— capacity of a system to maintatin convergent local behaviour against perturbances [Lóp07]
- **Predictability**: the system is designed for its behaviour to be predicted within appropriate safe limits [HSL08]
- **Reconfigurability**: the degree to which the system can be changed and it keeps its function [SR08]
- **Regulatory [Compliance]**: the degree of compliance to common policies and regulations [SR08]
- **Reliability**: the ability of a system to perform its required function(s)/functionalities under stated conditions for a specified period of time [SR08], [HSL08]
- **Robustness [Resilience]**: the capability of system to keep function in the presence of disturbances [SR08]
- **Reusability**: the possibility of reusing elements or a system in the construction of a new system [SR08]
- **Safety**: the capacity of a system of contributing to guarantee or to minimise personal or system damage in case of failure [HSL08]
- **Scalability**: the possibility of increasing the size of the task the system is handling without negative effect on other traits [SR08], [HSL08]
- **Security** : the protection against intrussion [SR08]
- **Self-X**: The capability of a system to operate on intself (monitoring, reflection, repairing, maintenance, reconfiguration) [SLB$^+$05], [SLB07], [HSL08]
- **Survivabiliity**: The ability of a system to preserve its core services, even when they are faulty of compromised [HSL08]
- **Upgradability**: The availability of a series of level of performance with seamless transitions between them [SR08]

- **Self-reconfiguration**: The capacity of the system to change its operation or configuration to adapt to unexpected or new situations [SLB$^+$05], [SLB07], [HSL08]
- **Self-optimisation [Self-Maintenance]**: The capacity of the system of handling its own maintenance [SLB$^+$05], [SLB07], [HSL08]
- **Self-healing [Self-Repairing]** : The capacity of the system of detecting errors and eliminate them [SLB$^+$05], [SLB07], [HSL08]
- **Self-protection**: The capacity of the system of automatically defends against malicious attacks or cascading failures. [SLB$^+$05], [SLB07], [HSL08]
- **Self-awarenes**: The capacity of the system of monitoring its own state, understanding functional implications, and taking appropriate actions over itself [SLB$^+$05], [SLB07], [HSL08]
- **Self-monitoring**: The capacity of the system of supervising the system's operation [SLB$^+$05], [SLB07], [HSL08]
- **ProductCharacterisation** : The definition and characterisation of product's characteristics based on users' and designers' goals, performance, quality of service, etc [SR08]
- **SystemCharacterisation** : The definition and characterisation of system's characteristics based on its observability, controlability, etc [SR08]
- **ProcessCharacterisation**: The definition and characterisation of process's characteristics based on its reliability, safety, security, etc [SR08]

11.2.2.2 ASys Perspective Package

1. Concepts
   - **RequirementViewpoint**: specialisation of Viewpoint regarding autonomous system requirements.
   - **ASysRequirement**: specialisation of Concern related to ASys requirements
   - **StructuralViewpoint**: specialisation of Viewpoint to address the autonomous system structure in terms of system, subsystems, elements, interfaces, etc.
   - **ASysStructure**: specialisation of Concern referring to the ASys physical structure
   - **FunctionalViewpoint**: specialisation of Viewpoint to address the autonomous system function .
   - **ASysFunction**: specialisation of Concern related to the ASys function as desired or expected behaviour of the different elements in the autonomous system.
   - **BehaviouralViewpoint**: specialisation of Viewpoint to consider the autonomous system behaviour
   - **ASysBehaviour**: specialisation of Concern related to the ASys behaviour on how the autonomous system operates under certain conditions.

- **PerformanceViewpoint**: specialisation of Viewpoint to address the autonomous system performance
- **ASysPerformance**: specialisation of Concern related to the ASys performance as evaluation and benchmarking of the autonomous system.

11.2.2.3 ASys Engineering Process Package

1. Concepts
   - **ASysGenericEngineeringProcess**: a generic process defined for autonomous systems engineering
   - **ASysAnalysis**: phase of the generic process dedicated to the analysis tasks of the autonomous system
   - **ASysDesign**: phase of the generic process dedicated to the design tasks of the autonomous system
   - **ASysEvaluation**: phase of the generic process dedicated to the evaluation tasks of the autonomous system
   - **ASysImplementation**: phase of the generic process dedicated to the implementation tasks of the autonomous system
   - **ASysRequirement**: phase of the generic process dedicated to the requirement engineering of the autonomous system
   - **BehaviouralAnalysis**: task of the ASysAnalysis phase dedicated to analysing the autonomous system from a behavioural viewpoint
   - **BehaviouralModel**: model kind as workproduct of the BehaviourModelling and ThoughtModellig subtask that contains the system or susbsystem behaviour and thought definition in a graphical form
   - **BehaviourModelling**: subtask of the BehaviouralAnalysis task that obtains the system or susbsystem behaviour models in the form of a graphical BehaviouralModel
   - **DeviceModel**: each one of the models to detail a system or subsystem devices in a graphical form
   - **FunctionModel**: each one of the models part of the FunctionalModel to detail a system or subsystem goal functions in a graphical form
   - **FunctionalModel**: model kind as workproduct of the FunctionModelling subtask that contains the system or susbsystem functions, roles, responsibilities and operations definition in a graphical form
   - **FunctionModelling**: subtask of the FunctionalAnalysis task that obtains the system or susbsystem function models in the form of a graphical FunctionalModel
   - **FunctionalAnalysis**: task of the ASysAnalysis phase dedicated to analysing the autonomous system from a functional viewpoint
   - **FunctionalRequirementSpecification**: subtask of the RequirementCharacterisation task that obtains the system or susbsystem functional requirement specification in the form of a textual description

- **GoalStructureModel**: each one of the models part of the KnowledgeModel to detail a system or subsystem goal hierarchical structure in a graphical form
- **KnowledgeModel**: model kind as workproduct of the KnowledgeModelling subtask that contains the system or susbsystem knowledge definition in a graphical form
- **KnowledgeModelling**: subtask of the StructuralAnalysis task that obtains the system or susbsystem knowledge models in the form of a graphical KnowledgeModel
- **Non–functionalRequirementSpecification**: subtask of the RequirementCharacterisation task that obtains the system or susbsystem non-functional requirement specification in the form of a textual description
- **OntologyModel**: each one of the models part of the KnowledgeModel to detail a system or subsystem domain ontology in a graphical form
- **OperationModel**: each one of the models part of the FunctionalModel to detail a system or subsystem operations in a graphical form
- **ProcedureModel**: each one of the models part of the KnowledgeModel to detail a system or subsystem procedures in a graphical form
- **ProcessCharacterisation**: each one of the documents part of the RequirementSpecification to define the non–functional requirements for the autonomous system or subsystems in a textual or tabular form
- **QuantityModel**: each one of the models part of the KnowledgeModel to detail a system or subsystem quantities in a graphical form
- **RequirementAnalysis**: task of the ASysAnalysis phase dedicated to defining the autonomous system's requirement
- **RequirementCharacterisation**: task of the ASysRequirement phase dedicated to defining the autonomous system and subsystems functional and non-functional requirements in terms of the ASys Requirement Package
- **RequirementSpecification**: model kind as workproduct of the Requirement Characterisation subtask that contains the system or susbsystem requirements in a textual or tabular form
- **ResponsibilityModel**: each one of the models part of the FunctionalModel to detail a system or subsystem responsibilities in a graphical form
- **RoalModel**: each one of the models part of the FunctionalModel to detail a system or subsystem actors and their role in a graphical form
- **StructuralAnalysis**: task of the ASysAnalysis phase dedicated to analysing the autonomous system from a structural viewpoint
- **StructuralModel**: model kind as workproduct of the SystemModelling subtask that contains the system or susbsystem structure definition in a graphical form
- **StructureModel**: each one of the models part of the StructuralModel to detail a system or subsystem whole-part structure in a graphical form
- **SystemCharacterisation**: each one of the documents part of the RequirementSpecification to define the functional requirements for the autonomous system or subsystems in a textual or tabular form

- **SystemModelling**: subtask of the StructuralAnalysis task that obtains the system or susbsystem structure models in the form of a graphical StructuralModel
- **SystemUseCase**: task of the ASysRequirement phase dedicated to defining the autonomous system and subsystems use cases in terms of the Requirement Package
- **SystemUseCaseModel**: each one of the models part of the UseCaseModel to formalise the autonomous system or subsystems use cases in a graphical form
- **ThoughtModelling**: subtask of the BehaviouralAnalysis task that obtains the system or susbsystem thought models in the form of a graphical BehaviouralModel
- **TopologyModel**: each one of the models part of the StructuralModel to detail a system or subsystem topological connections in a graphical form
- **UseCaseDescription**: each one of the documents part of the UseCaseSpecification to detail a related SystemUseCaseModel for the autonomous system or subsystems in a textual or tabular form
- **UseCaseDetailing**: subtask of the SystemUseCase task that obtains the system or susbsystem use cases in the form of a textual UseCaseSpecification
- **UseCaseModel**: model kind as workproduct of the UseCaseModelling subtask that contains the system or susbsystem use cases in a graphical form
- **UseCaseModelling**: subtask of the SystemUseCase task that obtains the system or susbsystem use cases in the form of a graphical UseCaseModel
- **UseCaseSpecification**: model kind as workproduct of the UseCaseDetailing subtask that contains the system or susbsystem use cases in a textual or tabular form

Bibliography

[ACGB$^+$06] A. Abran, J.J. Cuadrado, E. García-Barriocanala, O. Mendes, S. Sánchez-Alonso, and M.A. Sicilia. Engineering the ontology for the SWEBOK: Issues and techniques. In C. Calero, F. Ruiz, and M. Piattini, editors, *Ontologies for Software Engineering and Software Technology*, chapter 3, pages 103–121. Springer-Verlag Berlin Heidelberg, 2006.

[ADHJ06] L. Alawneh, M. Debbabi, F. Hassaine, and Y. Jarraya. A unified approach for verification and validation of systems and software engineering models. In *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)*, 2006.

[AdOD06] N. Anquetil, K.M. de Oliveira, and M.G.B. Dias. Software maintenance ontology. In C. Calero, F. Ruiz, and M. Piattini, editors, *Ontologies for Software Engineering and Software Technology*, chapter 5, pages 153–173. Springer-Verlag Berlin Heidelberg, 2006.

[AK03] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003.

[Ash56] W. R. Ashby. *An Introduction to Cybernetics.* Chapman Hall, London, 1956.

[Ash58] W. R. Ashby. General systems theory as a new discipline. *General Systems Yearbook*, 3, 1958.

[AT06a] A. Akerman and J. Tyree. Using ontology to support development of software architectures. *IBM Systems Journal*, 45(4):813–825, 2006.

[AT06b] ASLab Team. Core mental terminology: from an autonomous system perspective. Technical Report R-2006-XXX, Autonomous Systems Laboratory (ASLab), September 2006.

[AV00] J. C. Arpirez-Vega. WebODE 1.0: User's manual. http://webode.dia.fi.upm.es/WebODEWeb/index.html, December 2000.

[AZW06] U. Assmann, S. Zschaler, and G. Wagner. Ontologies, meta–models, and the model–driven paradigm. In C. Calero, F. Ruiz, and M. Piattini, editors, *Ontologies for Software Engineering and Software Technology*, chapter 9, pages 249–273. Springer-Verlag Berlin Heidelberg, 2006.

[BA07] J. Bermejo-Alonso. A simplified guide to create an ontology. Technical Report ASLab-R-2007-004, v 0.1 Draft, Autonomous Systems Laboratory (ASLab), May 2007.

[Bal06] L. Balmelli. An overview of the systems modeling language for products and systems development. Technical Report TR-20060603, IBM Research Division, October 2006.

[BAM$^+$04] T. Barbera, J. Albus, E. Messina, C. Schlenoff, and J. Horst. How task analysis can be used to derive and organize the knowledge for the control of autonomous vehicles. *Robotics and Autonomous Systems*, 49:67–78, 2004.

[BAS06] J. Bermejo-Alonso and R. Sanz. A survey on ontologies for agents: Integrating cognition and emotion for autonomous systems. Technical Report ASLab-ICEA-R-2006-002, Autonomous Systems Laboratory (ASLab), April 2006.

[BASRH10a] J. Bermejo-Alonso, R. Sanz, M. Rodríguez, and C. Hernández. An ontology–based approach for autonomous systems' description and engineering: the OASys Framework. In R. Setchi, I. Jordanov, R. J. Howlett, and L. C. Jain, editors, *14th International Conference on Knowledge–Based and Intelligent Information and Engineering Systems (KES 2010)*, volume 6276 of *LNAI*, pages 522–531, Cardiff, Wales, U.K., 2010. Springer, Heidelberg.

[BASRH10b] J. Bermejo-Alonso, R. Sanz, M. Rodríguez, and C. Hernández. Ontology–based engineering of autonomous systems. In M. Bauer, J. Lloret Mauri, and O. Dini, editors, *Proceedings of the The Sixth International Conference on Autonomic and Autonomous Systems (ICAS 2010)*, pages 47–51, Cancun, Mexico, 7–13 March 2010. IEEE Computer Society.

[BAT97] P. Borst, H. Akkermans, and J. Top. Engineering ontologies. *International Journal of Human-Computer Studies*, 46:365–406, 1997.

[BBCM06] L. Balmelli, D. Brown, M. Cantor, and M. Mott. Model–driven systems development. *IBM Systems Journal*, 45(3):569–585, 2006.

[BCKS04] M. Broxvall, S. Coradeschi, L. Karlsson, and A. Safotti. Have another look on failures and recovery planning in perceptual anchoring. In P. Doherty, G. Lakemeyer, and A.P. del Pobil, editors, *Fourth International Cognitive Robotics Workshop*, pages 95–100, Valencia, Spain, 23-24 August 2004.

[BCT05] A.W. Brown, J. Conallen, and D. Tropeano. *Model-Driven Software Development*, chapter Introduction: Models, Modeling, and Model-Driven Architecture (MDA), pages 1–16. Springer-Verlag Berlin Heidelberg, 2005.

[Bek05] George A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*, chapter Autonomy and Control in Animals and Robots. Intelligent Robotics and Autonomous Agents. The MIT Press, 2005.

[BHS07] F. Baader, I. Horrocks, and U. Sattler. *Handbook on Knowledge Representation*, chapter Description Logics. Number 3. Elsevier, 2007.

[BKK$^+$02] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, J. Letkowski, and P. Emery. Extending the unified modeling language for ontology development. *Software Systems Modeling*, (1):142–156, 2002.

[BKS$^+$02] K. Baclawski, M. Kokar, J. Smith, E. Wallace, J. Letkowski, M. Koethe, and P. Kogut. UOL: Unified ontology language. Assorted papers discussed at the DC Ontology SIG Meeting, 2002.

[BLC96] A. Bernaras, I. Laresgoiti, and J. Corera. Building and reusing ontologies for electrical network applications. In W. Wahlster, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI'96)*, pages 298–302, Budapest, Hungary, 1996. John Wiley and Sons, Chichester, U.K.

[Boc05] C. Bock. Systems engineering in the product lifecycle. *International Journal of Product Development*, 2(1-2):123–137, 2005.

[Boc06] C. Bock. SysML and UML 2.0 support for activity modeling. *Systems Engineering*, 9(2):160–186, 2006.

[Bor97] W. N. Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Centre for Telematics and Information Technology, University of Tweenty. Enschede. The Netherlands, 1997.

[Bou56] K. E. Boulding. General Systems Theory – the skeleton of science. *Management Science*, 2(3):197–208, 1956.

[BP01] F. Bergenti and A. Poggi. Exploiting UML in the design of multi-agent systems. *Lecture Notes in Computer Science*, 1972:106–113, 2001.

[BP02] F. Bergenti and A. Poggi. Supporting agent–oriented modelling with UML. *International Journal of Software Engineering and Knowledge Engineering*, 12(6):605–618, 2002.

[BPSM$^+$06] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible markup language (xml) 1.1. WebPage, September 2006.

[Bri93] D. Brill. *Loom Reference Manual.* University of Southern California, version 2.0 edition, December 1993.

[BVG06] M.F. Bertoa, A. Vallecillo, and F. GarcÌa. *Ontologies for Software Engineering and Software Technology*, chapter An Ontology for Software Measurement, pages 175–196. Springer-Verlag Berlin Heidelberg, 2006.

[Cal09] R. Calinescu. *Autonomic Computing and Networking*, pages 3–29. Springer, April 2009.

[CCPR02] A. Chella, M. Cossentino, R. Pirrone, and A. Ruisi. Modeling ontologie for robotic environments. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02*, pages 77–80, Ischia, Italy, July 15-19 2002.

[CD00] Q. Chen and U. Dayal. Multi–agent cooperative transactions for e–commerce. In *Conference on Cooperative Information Systems*, pages 311–322, 2000.

[Ceb05] M. Cebulla. Knowledge-based assessment of behaviour in dynamic environments. In *Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems*, pages 17–26, Bremen, Germany, November 2005. ACM Press.

[CFF$^+$98] V. K. Chaudri, A. Farquhar, R. Fikes, R. D. Karp, and J. P. Rice. Open Knowledge Base Connectivity 2.0.3. Technical report, Knowledge Systems Laboratory, 1998.

[CH08] R.P. Conde and A. Hernando. Manual de higgs: plataforma de pruebas Pioneer 2AT–8. Technical Report M–2005–009, Autonomous Systems Laboratory (ASLab), January 2008.

[Cha04] C.W. Chan. Knowledge and software modeling using UML. *Software Systems Modelling*, (3):294–302, 2004.

[CHP01] S. Cranefield, S. Haustein, and M. Purvis. UML-based ontology modelling for software agents. In *Workshop on Ontologies in Agent Systems, OAS'2001*, 2001.

[CJB98] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. Ontology of tasks and methods. In *11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Banff, Canada, 1998.

[CJB99] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. What are ontologies and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[Con05] R. P. Conde. Mapeo facial de emociones sintéticas. Final year project, E.T.S.I.I.M., Marzo 2005.

[Con07] P. Conejo. Sistema portable de operación remota para Higgs. Final year project, E.T.S.I.I.M., Septiembre 2007.

[CP02a] S. Cranefield and M. Purvis. Ontologies for interaction protocols. In *Proceedings of the Workshop on Ontologies in Agent Systems*, Bologna, Italy, July 2002.

[CP02b] S. Cranefield and M. Purvis. A UML profile and mapping for the generation of ontology–specific content languages. *The Knowledge Engineering Review, Special Issue on Ontologies in Agent Systems*, (17):21–39, 2002.

[CPL05] O. Corcho, A. Gómez Pérez, and M. Fernández López, editors. *CAEPIA2005: Taller Ontologías y Web Semántica*, Santiago de Compostela, Spain, Noviembre 2005.

[CPN00] S. Cranefield, M. Purvis, and M. Nowostawski. Is in an ontology or an abstract syntax?: Modelling objects, knowledge and agent messages. In *Workshop on Applications of Ontologies and Problem–Solving Methods*, pages 16.1–16.4, 2000.

[CPNH05] S. Cranefield, M. Purvis, M. Nowostawski, and P. Hwang. Ontologies for interaction protocols. In V. Tamma, S. Cranefield, T.W. Finnin, and S. Willmott, editors, *Ontologies for Agents: Theory and Experiences*, Whitestein Series in Software Agent Technologies, pages 1–17. Birkhäuser, 2005.

[CTSB04] N.L. Cassimatis, J.G. Trafton, A.C. Schultz, and M.D. Bugajska. Integrating cognition, perception and action through mental simulation in robots. *Robotics and Autonomous Systems*, 49:13–23, 2004.

[Cyc] Cycorp. Open cyc documentation and software. http://www.opencyc.org/.

[DAdO03] M.G. Dias, N. Anquetil, and K.M. de Oliveira. Organizing the knowledge used in software maintenance. *Journal of Universal Computer Science*, 9(7):641–658, 2003.

[Dev01] V. Devedzic. Knowledge modeling – state of the art. *Integrated Computer–Aided Engineering*, 8:257–281, 2001.

[Dev02] V. Devedzic. Understanding ontological engineering. *Communications of the ACM*, 45(4):136–144, April 2002.

[Dju04] D. Djuric. MDA-based ontology infrastructure. *Computer Science and Information Systems*, 1(1):91–116, 2004.

[dlM09a] J. L. de la Mata. CSTR mathematical model: The main PCT testbed. Technical Report R-2009-005, Autonomous Systems Laboratory (ASLab), March 2009.

[dlM09b] J. L. de la Mata. CSTR overall specification: The main PCT testbed. Technical Report R-2009-001, Autonomous Systems Laboratory (ASLab), 2009.

[DMC99] J. Domingue, E. Motta, and O. Corcho. Knowledge modelling in WebOnto and OCML: a user guide. http://projects.kmi.open.ac.uk/webonto/, 1999.

[DMQ05] D. Dou, D. McDermott, and P. Qi. Ontology translation by ontology merging and automated reasoning. In V. Tamma, S. Cranefield, T.W. Finnin, and S. Willmott, editors, *Ontologies for Agents: Theory and Experiences*, Whitestein Series in Software Agent Technologies, pages 73–94. Birkhäuser, 2005.

[Dou04] B. P. Douglass. *Real Time UML: advances in the UML for real–time systems.* The Addison–Wesley object technological. Addison-Wesley, 3rd edition, 2004.

[dOZR$^+$04] K.M. de Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Galotta, and C.S. de Menezes. Domain–oriented software development environment. *Journal of Systems and Software*, (72):145–161, 2004.

[DTC08] SEAS DTC. Engineering autonomous system architectures. In *Proceedings of Third Annual Conference.* Systems Engineering for Autonomous Systems Defence Technology Centre, Edimburgh, UK, 24 and 25 June 2008. SER013.

[DV07] S. A. DeLoach and J. L. Valenzuela. An agent-environment interaction model. In P. Lin and F. Zambonelli, editors, *Agent-Oriented Software Engineering VII: The 7th International Workshopp (AOSE 2006)*, volume 4405 of *LNCS*, 2007.

[Ebe03] A. Eberhart. *Ontology-Based Infrastructure for Intelligent Applications.* Phd thesis, University of Saarbrücken, December 2003.

[Eps04] S. L. Epstein. Metaknowledge for autonomous systems. In *Proceedings of the 2004 AAAI Spring Symposium on Knowledge Representation and Ontologies for Autonomous Systems*, Stanford, Palo Alto, Ca., March 2004.

[Est08] J. A. Estefan. Survey of model–based systems engineering (MBSE) methodologies. Technical Report INCOSE-TD-2007-003-01 (Rev. B), ModelBased Systems Engineering (MBSE) Initiative, International Council on Systems Engineering (INCOSE), 2008.

[Fal04] R.A. Falbo. Experiences in using a method for building domain ontologies. In *Proceedings of Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2004)*, pages 474–477, Alberta, Canda, June 2004.

[FCF+09] D. G. Firesmith, P. Capell, D. Falkenthal, C.B. Hammons, D. T. Latimer IV, and T. Merendino. *The Method Framework for Engineering System Architectures.* CRC Press, 2009.

[FFR96] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: A tool for collaborative ontology construction reference:. *Knowledge Systems*, 1996.

[FG96] S. Franklin and A. Graesser. Is it an agent, or just a program? a taxonomy for autonomous agents. In *Third International Workshop on Agent Theories, Architectures and Languages*, Intelligent Agents III, pages 21–35. Springer Verlag, 1996.

[FGD02] R.A. Falbo, G. Guizzardi, and K.C. Duarte. An ontological approach to domain engineering. In *Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 351–358, Ischia, Italy, July 15-19 2002.

[FLGPJ97] M. Fernández-López, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: from ontological art towards ontological engineering. In A. Farquhar, M. Grüninger, A. Gómez-Pérez, M. Uschold, and P. van der Vet, editors, *AAAI'97 Spring Symposium on Ontological Engineering*, pages 33–40, Stanford University, CA, U.S.A., 1997.

[FMR98] R.A. Falbo, C.S. Menezes, and A.R. Rocha. A systematic approach for building ontologies. In Springer-Verlag Berlin Heidelberg, editor, *Proceedings of 6th Ibero–American Conference on AI*, number LNCS1484 in Lecture Notes in Artificial Intelligence, pages 349–360, Lisbon, Portugal, October 1998.

[FMS06] S. Friedenthal, A. Moore, and R. Steiner. OMG Systems Modeling Language (OMG SysML) tutorial. July 2006.

[FMS08] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: The Systems Modeling Language.* Morgan Kaufmann and OMG Press, 2008.

[FNM+03] R.A. Falbo, A.C.C Natali, P.G. Mian, G. Bertollo, and F. B. Ruy. ODE: Ontology–based software development environment. In *Proceedings of IX Congreso Argentino de Ciencias de la ComputaciÛn*, pages 1124–1135, La Plata, Argentina, October 2003.

[For01] Software Engineering Task Force. Ontology driven architectures and potential uses of the semantic web in systems and software engineering, 2001.

[FRM05] R.A. Falbo, F.B. Ruy, and R.D. Moro. Using ontologies to add semantics to a software engineering environment. In *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005)*, pages 151–156, Taipei, China, July 2005.

[FRPM04] R.A. Falbo, F.B. Ruy, J. Pezzin, and R. Dal Moro. Ontologias e ambientes de desenvolvimento de software sem?nticos. In *Proceedings of IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'2004)*, volume I, pages 277–292, Madrid, Spain, November 2004.

[GDD06a] D. Gasevic, D. Djuric, and V. Devedzic. Mappings of mda-based languages and ontologies. In *Model Driven Architecture and Ontology Development*, chapter 10. Springer-Verlag Berlin Heidelberg, 2006.

[GDD06b] D. Gasevic, D. Djuric, and V. Devedzic. An MDA based ontology platform: AIR. In *Model Driven Architecture and Ontology Development*, chapter 12. Springer-Verlag Berlin Heidelberg, 2006.

[GDD06c] D. Gasevic, D. Djuric, and V. Devedzic. *Model Driven Architecture and Ontology Development.* Springer–Verlag Berlin Heidelberg, 2006.

[GDD06d] D. Gasevic, D. Djuric, and V. Devedzic. The ontology uml profile. In *Model Driven Architecture and Ontology Development*, chapter 9. Springer-Verlag Berlin Heidelberg, 2006.

[GDDD04] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovic. From UML to ready-to-use OWL ontologies. In *Proceedings of the Second IEEE International Conference on Intelligent Systems*, pages 485–490, June 2004.

[GdFB04] R. Girardi, C. Gomes de Faria, and L. Balby. Ontology–based domain modeling of multi–agent systems. In *Proceedings of Third International Workshop on Agent–Oriented Methodologies (OOPSLA 2004)*, Vancouver, Canada, 24-28 October 2004.

[GF95] M. Grüninger and M.S. Fox. Methodology for the design and evaluation of ontologies. In D. Skuce, editor, *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 6.1–6.10, Montreal, Canada, 1995.

[GF03] R. Girardi and C. Faria. A generic ontology for the specification of domain models. In S. Overhage and K. Turowski, editors, *Proceedings of 1st International Workshop on Component Engineering Methodology (WCEM'03)*, pages 41–50, Erfurt, Germany, September 22-25 2003.

[GG95] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In Mars N, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95)*, pages 25–32, University of Twente, Enschede, The Netherlands, 1995. IOS Press, Amsterdam, The Netherlands.

[GN87] M.R. Genesereth and L. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, California, 1987.

[God07] M. F. Godoy. Desarrollo de sistema distribuido sobre robot móvil autónomo para caracterización de terrenos. Final year project, E.T.S.I.I.M., 2007.

[GPFLC04a] A. Gómez Pérez, M. Fernández López, and M. Corcho. Methodologies and methods for building ontologies. In *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, Advanced Information and Knowledge Processing, chapter 3, pages 107–197. Springer, 2004.

[GPFLC04b] A. Gómez Pérez, M. Fernández López, and M. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web.* Advanced Information and Knowledge Processing. Springer, 2004.

[GPHS06] C. González-Pérez and B. Henderson-Sellers. An ontology for software development methodologies and endeavours. In C. Calero, F. Ruiz, and M. Piattini, editors, *Ontologies for Software Engineering and Software Technology*, chapter 4, pages 123–151. Springer-Verlag Berlin Heidelberg, 2006.

[GPR04] F. García, M. Platini, and F. Ruiz. FMESP: Framework for the modelling and evaluation of software processes. In *Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process QUTE–SWAP*, pages 5–13, Newport Beach, CA., U.S.A., 5 November 2004.

[Gro08a] Object Management Group. *OMG SysML Specification.* Object Management Group, v 1.1 edition, November 2008.

[Gro08b] Object Management Group. Software and systems process engineering meta–model specification version 2.0. OMG Formal Specification 2008–04–01, Object Management Group, Inc., 2008.

[Gru93a] T. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gru93b] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, Padova, Italy, 1993. Kluwer Academic Publishers.

[GSV04] T. Gabel, Y. Sure, and J. Voelker. Kaon - an overview. http://kaon.semanticweb.org, April 2004.

[Gua97] N. Guarino. Understanding, building and using ontologies. *International Journal of Human and Computer Studies*, 46:293–310, 1997.

[Gua98] N. Guarino. Formal ontology in information systems. In *Proceedings of FOIS'98*, pages 3–15, Trento, Italy, June 1998. IOS Press, Amsterdam.

[Gui05] G. Guizzardi. *Ontological Foundations for Structural Conceptual Models*. Phd thesis, University of Twente, The Netherlands, 2005.

[Gyu06] S. M. De Gyurky. *The Cognitive Dynamics of Computer Science*. Wiley–Interscience, 2006.

[HB06] J. Hallam and H. Bruynickx. An ontology of robotics science. In H. I. Christensen, editor, *Proceedings of the European Robotics Symposium 2006 (STAR 22)*, pages 1–14. Springer–Verlag Berlin Heidelberg, 2006.

[Her05] A. Hernando. Versión RT–CORBA del servidor Pioneer 2AT–8. Final year project, E.T.S.I.I.M., Junio 2005.

[Hes05] W. Hesse. Ontologies in the software engineering process. In R. Lenz, editor, *Proceedings of Tagungsband Workshop on Enterprise Application Integration (EAI2005)*, Berlin, Germany, 2005. GITO–Verlag.

[HH08] C. Hernández and A. Hernando. RCT overall specification: Higgs platform. Technical Report R-2008-XXX, Autonomous Systems Laboratory (ASLab), April 2008.

[HJM+07] M. Horridge, S. Jupp, G. Moulton, A. Rector, R. Stevens, and C. Wroe. A practical guide to building OWL ontologies using Protege 4 and CO-ODE tools edition 1.1. http://protege.stanford.edu/, October 2007.

[HM07] M. Huebscher and J. McCann. A survey of autonomic computing – degrees, models and applications. *ACM Computing Surveys*, December 2007.

[Hru05] P. Hruby. Ontology–based domain–driven design. In *Proceedings of Object-Oriented Programming, Systems, Languages And Applications (OOPSLA'05)*, San Diego, California, U.S.A., October 2005.

[HS06] H.J. Happel and S. Seedorf. Applications of ontologies in software engineering. In *Proceedings of 2nd International Workshop on Semantic Web Enabled Software Engineering (SEWE 2006)*, Athens, GA, U.S.A., November 2006.

[HSGP08] B. Henderson-Sellers and C. Gonzalez-Peres. *Metamodelling for Software Engineering*. John Wiley and Sons, Ltd., 2008.

[HSL08] C. Hernández, R. Sanz, and I. López. Consciousness in cognitive architectures: a principled analysis of RCS, Soar and ACT–R. Technical Report R–2008–004, Autonomous Systems Laboratory (ASLab), February 2008.

[HTM05] M. Hause, F. Thorn, and A. Moore. Inside sysML. *IEE Computing Control Engineering*, pages 10–15, August-September 2005.

[Hua04] Hui-Min Huang. Autonomy Levels for Unmanned Systems (ALFUS) Framework. NIST Special Publication 1011, National Institute of Standards and Technology (NIST), September 2004.

[IBGM08] J. Ierache, M. Bruno, and R. García-Martínez. Ontología para el aprendizaje y compartición de conocimientos entre sistemas autónomos. In *XIV Congreso Argentino de Ciencias de la Computación*, Universidad Nacional de Chilecito, 2008.

[IBM01] IBM. Autonomic computing: IBM's perspective on the state of information technology. http://www.research.ibm.com/autonomic/manifesto/, 2001.

[IEE90] IEEE Computer Society, New York. *IEEE Standard Glossary of Software Engineering Terminology*, IEEE std 610.12 1990 edition, 1990.

[IMDt94] IDEF5-Method-Development-team. IDEF5 method report. Technical report, Knowledge Based Systems, Inc., September 1994.

[Ins00] Institute for Electrical and Electronics Engineering, New York. *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*, IEEE std 1471-2000 edition, 2000.

[JBK$^+$04] H. Jung, J.M. Bradshaw, S. Kulkarni, M. Breedy, L. Bunch, P. Feltovich, R. Jeffers, M. Johnson, J. Lott, N. Suri, W. Taysom, G. Tonti, and A. Uszok. An ontology–based representation for policy–governed adjustable autonomy. In *Proceedings of the 2004 AAAI Spring Symposium on Knowledge Representation and Ontologies for Autonomous Systems*, Stanford, California, March 2004.

[JL04] K.C. Tsui J. Liu, X. Jin. *Autonomy Oriented Computing: From Problem Solving to Complex Systems Modeling (Multiagent Systems, Artificial Societies, and Simulated Organizations)*. Springer, 2004.

[JU99] R. Jasper and M. Uschold. A framework for understanding and classifying ontology applications. In *12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Alberta, 1999.

[KA07] C.M. Keet and A. Artale. Representing and reasoning over a taxonomy of part–whole relations. *Applied Ontology*, 0(1):1–17, 2007.

[Kal00] Y. Kalfoglou. *Deploying Ontologies in Software Design.* Phd thesis, University of Edinburgh, 2000.

[KC03] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[KCL$^+$02] P. Kogut, S. Cranefield, L.Hart, M. Dutra, K. Baclawski, M. Kokar, and J. Smith. UML for ontology development. *The Knowledge Engineering Review*, 1(17):61–64, 2002.

[KE03] G. J. Klir and D. Elias. *Architecture of Systems Problem Solving*, volume 21 of *IFSR International Series on Systems Science and Engineering*. Kluwer Academic Publishers, 2003.

[Kit06] Y. Kitamura. 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)*, pages 59–69, Poland, September 2006.

[Kli69] G. J. Klir. *Approach to General Systems Theory*. Van Norstrand Reinhold, New York, 1969.

[Kli85] G. J. Klir Klir. The emergence of two–dimensional science in the information society. *Systems Research*, 2(1):33–41, 1985.

[Kli91] G. J. Klir. *Facets of Systems Science*. Plenum Press, 1991.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object–oriented and frame–based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KMAM00] Y. Kalfoglou, T. Menzies, K. Athoff, and E. Motta. Meta–knowledge in systems design: panacea ... or undelivered promise? *The Knowledge Engineering Review*, 15(4):381–404, 2000.

[Knu04] H. Knublauch. Ontology–driven software development in the context of the semantic web: An example scenario with protëgë/owl. In *Proceedings of International Workshop on the Model–Driven Semantic Web (MDSW2004)*, Monterey, California, 2004.

[KS06] H. Kaiya and M. Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *Proceedings of 14th IEEE International Requirements Engineering Conference,(RE'06)*, pages 189–198, Minneapolis/St. Paul, Minnesota, USA, Sep. 2006. IEEE CS.

[LD01] O. Lassila and McGuinness D. The role of frame-based representation on the Semantic Web. Technical Report KSL-01-02, Knowledge Systems Laboratory, Stanford University, Stanford, California, 2001.

[LG90] D.B. Lenat and R.V. Guha. *Building large knowledge-based system: representation and inference in the Cyc project*. Addison-Wesley, New York, 1990.

[LG05] S.W. Lee and R.A. Gandhi. Ontology-based active requirements engineering framework. In *Proceedings of 12th Asia-Pacific Software Engineering Conference (ASPEC'05)*, Taipei, Taiwan, 2005. IEEE.

[LLS03] W. Liu, Z-T. Liu, and K. Shao. UML–based domain ontology modeling for multi–agent system. In *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, pages 407 – 412, Xi'an, 2-5 November 2003.

[Lóp07] I. López. *A Foundation for Perception in Autonomous Systems.* Phd thesis, Departamento de Automática, Ingeniería Electrónica e Informática Industrial, Universidad Politécnica de Madrid, 2007.

[LQL05] L. Liao, Y. Qu, and H.K.N. Leung. A software process ontology and its application. In *Proceedings of Workshop on Sematic Web Enable Software Engineering (SWESE)*, Galway, Ireland, November, 6-10 2005.

[LSAF06] E. Lehtihet, J. Strassner, N. Agoulmine, and M. O Foghlu. Ontology-based knowledge representation for self-governing systems. In R. State, editor, *Proceedings of the 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, volume LNCS 4269. IFIP International Federation for Information Processing, 2006.

[LSB07] I. López, R. Sanz, and J. Bermejo. A unified framework for perception in autonomous system. In D. S. McNamara and J. G. Trafton, editors, *Proceedings of the 29th Annual Cognitive Science Society*, pages 1229–1234, Austin, TX, 2007. Cognitive Science Society.

[LSRH97] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[MA02] Alexander M. Meystel and James S. Albus. *Intelligent Systems: Architecture, Design, and Control.* Wiley Series on Intelligent Systems. Wiley–Interscience, 2002.

[MA05a] O. Mendes and A. Abran. Issues in the development of an ontology for an emerging engineering discipline. In *Proceedings of 1st Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering (ONTOSE)*, volume 163, Alcalá de Henares, Madrid, Spain, June 2005. CEUR Workshop Proceedings.

[MA05b] O. Mendes and A. Abran. Software engineering ontology: A development methodology. *METRICS NEWS*, (9):68–76, 2005.

[Mac] R. MacGregor. Retrospective on LOOM. http://www.isi.edu/isd/LOOMpapers/macgregor/Loom_Retrospective.html.

[MBG$^+$03] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Wonderweb deliverable d18: Ontology library. Technical report, Laboratory for Applied Ontology -ISTC-CNR, December 2003.

[MBW$^+$08] J. Morbach, B. Bayer, A. Wiesner, A. Yang, and W. Marquardt. OntoCAPE 2.0: the upper level. Technical Report LPT–2008–25, RWTH Aachen University, July 2008.

[MHHB94] N.H. Madhavji, D. Holtje, W. Hong, and T. Bruckhaus. Elicit: a method for eliciting process models. In *Proceedings of the 3rd International Conference on Software Process*. IEEE Computer Society, October 1994.

[Miz04] R. Mizoguchi. Tutorial on ontological engineering - part 2: ontology development, tools and languages. *New Generation Computing*, 22(1):61–96, 2004.

[Miz08] R. Mizoguchi. Functional ontology of artifacts. In *Proceeding of the Interdisciplinary Ontology Conference (InterOntology08)*, Tokyo, Japan, February 2008.

[MO04] M.A. Martín and L. Olsina. Towards an ontology for software metrics and indicators as the foundation for a cataloging web system. In *Proceedings of 4th International Conference Web Enginering (ICWE 2004)*, volume 3140 of *Lecture Notes in Computer Science*, Münich, Germanynich, Germany, July 2004. Springer Berlin-Heidelberg.

[MP04] H.R. Maturana and B. Poerksen. *From Being to Doing, The Origins of the Biology of Cognition*. Carl-Auer, 2004.

[MPO04] A. Malucelli, D. Palzer, and E. Oliveira. B2b transactions enhanced with ontology–based services. In *ICETE'04 – 1St International Conference on E–Business And Telecommunication Networks*, pages 10–17, Portugal, August 2004.

[MPO05] A. Malucelli, D. Palzer, and E. Oliveira. Combining ontologies and agents to help in solving the heterogeneous problem in e-commerce negotiations. In *International Workshop on Data Engineering Issues in E-Commerce (DEEC 2005), IEEE Computer Society*, pages 26–35, Tokyo, Japan, April 2005.

[MSSS$^+$07] J. Martin-Serrano, J. Serrat, J. Strassner, G. Cox, R. Carroll, and M. O Foghlu. Policy–based context integration and ontologies in autonomic applications to facilitate the information interoperability in NGN. In *Proceedings of the Workshop on Hot Topics in Autonomic Computing*, Jacksonville, FL, U.S.A., 2007. USENIX Association.

[MW] Incorporated Merriam-Webster. Merriam-Webster's Online Dictionary. http://www.merriam-webster.com/dictionary/.

[MWM07] J. Morbach, A. Wiesner, and W. Marquardt. A meta model for the design of domain ontologies. Technical Report LPT-2008-24, RWTH Aachen University, July 2007.

[NCL06] H-S. Na, O-H. Choi, and J-E. Lim. A method for building domain ontologies based on the transformation of UML models. In *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Application (SERA'06)*, 2006.

[NF05] M.H. Nodine and J. Fowler. On the impact of ontological commitments. In V. Tamma, S. Cranefield, T.W. Finin, and S. Willmott, editors, *Ontologies for Agents: Theory and Experiences*, Whitestein Series in Software Agent Technologies, pages 19–42. Birkhäuser, 2005.

[NH97] N.F. Noy and C.D. Hafner. The state of the art in ontology design. *AI Magazine*, 18(3):53–74, 1997.

[NM01] N.F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL–01–05, Stanford Knowledge Systems Laboratory, March 2001.

[NP01] I. Niles and R. A. Pease. Towards a standard upper ontology. In B. Smith and C. Welty, editors, *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'01)*, pages 2–9, Maine, 2001. ACM Press.

[NS06] M. R. Nami and M. Sharifi. *Intelligent Information Processing III*, volume 228 of *IFIP International Federation for Information Processing*, chapter A survey of autonomic computing systems, pages 101–110. Springer, 2006.

[Obj03] Object Management Group. *MDA Guide Version 1.0.1*, June 2003.

[Obj05] Object Management Group. *Unified Modeling Language (UML), V. 2.0.* Object Management Group, July 2005.

[Obj06a] Object Management Group. *Meta Object Facility (MOF) Core Specification Version 2.0*, January 2006.

[Obj06b] Object Management Group. *Object Constraint Language, OMG Available Specification Version 2.0*, May 2006.

[Obj09a] Object Management Group. *OMG Unified Modeling Language (OMG UML) Infrastructure Version 2.2*, February 2009.

[Obj09b] Object Management Group. *OMG Unified Modeling Language (OMG UML) Superstructure Version 2.2*, February 2009.

[Obj09c] Object Management Group. *Ontology Definition Metamodel Version 1.0*, May 2009.

[OESV04] D. Oberle, A. Eberhart, S. Staab, and R. Volz. Developing and managing software components in an ontology–based application server. In H.A. Jacobsen, editor, *Proceedings of Middleware 2004, ACM/IFIP/USENIX 5th International Middleware Conference*, volume 3231 of *LNCS*, pages 459–477, Toronto, Ontario, Canada, 2004.

[OH05] J.S. Osmundson and T.V. Huynh. A systems engineering methodology for analyzing systems of systems. In *Proceedings of the 1st Annual System of Systems Engineering Conference (SoSECE'02*, pages 1–44, Pennsylvania, U.S.A., June 13-14 2005.

[OK03] B. Omelayenko and M.C.A. Klein, editors. *Knowledge Transformation for the Semantic Web*, volume 95 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, 2003.

[ont] LOOM Ontosaurus.

[OVRT06] K.M. Oliveira, K. Villela, A.R. Rocha, and G.H. Travassos. *Ontologies for Software Engineering and Software Technology*, chapter Use of Ontologies in Software Development Environments, pages 275–309. Springer-Verlag Berlin Heidelberg, 2006.

[Pan05] C. Panizo. Módulo de habla CORBA para SOUL. Final year project, E.T.S.I.I.M., Noviembre 2005.

[Par04] I. Pareja. Sistema de comunicaciones para Pioneer 2–AT8. Final year project, E.T.S.I.I.M., Marzo 2004.

[PB06] D. Price and R. Bodington. On the use of semantic web technology for requirements satisfaction, or how do I find a good bike? In *Proceedings of the 16th Annual International Symposium of the International Council On Systems Engineering (INCOSE)*, 8 - 14 July 2006.

[Pre] Oxford University Press. Compact Oxford English Dictionary of Current English. http://www.askoxford.com.

[PUS] R. Provine, M. Uschold, and S. Smith. Observations on the use of ontologies for autonomous vehicle navigation planning. In *Proceedings of the 2004 AAAI Spring Symposium on Knowledge Representation and Ontologies for Autonomous Systems*, Stanford, California, March.

[RBF04] F.B. Ruy, G. Bertollo, and R.A. Falbo. Knowledge-based support to process integration in ODE. *CLEI Electronic Journal*, 7(1), August 2004.

[RGPP02] F. Ruiz, F. García, M. Piattini, and M. Polo. *Advances in Software Maintenance Management: Technologies and Solutions*, chapter Environment for Managing Software Maintenance Projects, pages 255–290. Idea Group Inc., 2002.

[RH06] F. Ruiz and J.R. Hilera. Using ontologies in software engineering and technology. In C. Calero, F. Ruiz, and M. Piattini, editors, *Ontologies for Software Engineering and Software Technology*, chapter 2, pages 49–95. Springer-Verlag Berlin Heidelberg, 2006.

[RHJ99] D. Ragett, A. Le Hors, and I. Jacobs. HTML 4.01 specification. http://www.w3.org/TR/html401/, December 1999.

[RJB04] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Object Technology. Addison-Wesley, second edition, 2004.

[Roc06] C. Roche. Is ontology overrated?: terminology, concept modelling and ontology. In *Proceedings of the XXVI VAKKI Symposium*, pages 1–20, Vaasa, Finland, February 2006.

[Rod07] M. Rodríguez. PCT use cases. Technical report, Autonomous Systems Laboratory (ASLab), 2007.

[Ros86] R. Rosen. Some comments on systems and systems theory. *International Journal of General Systems*, 1986.

[RVPG04] F. Ruiz, A. Vizcaíno, M. Piattini, and F. García. An ontology for the management of software maintenance projects. *International Journal of Software Engineering*, 14(3):323–349, 2004.

[Sae04] M. Saeki. Ontology-based software development techniques. *ERCIM News*, (58):14–15, 2004.

[SAL06] J. C. Strassner, N. Agoulmine, and E. Lehtihet. FOCALE: a novel autonomic networking architecture. In *Proceedings of the First Latin American Autonomic Computing Symposium (LAACS 2006)*, 2006.

[Sán05] S. Sánchez. Integración de SOAR de ICA. Final year project, E.T.S.I.I.M., Marzo 2005.

[San07] R. Sanz. RCT use cases. Technical report, Autonomous Systems Laboratory (ASLab), 2007.

[SAS$^+$99] R. Sanz, I. Alarcón, I. Segarra, M.J. de Antonio, and J.A. Clavijo. Progressive domain focalization in intelligent control systems. *Control Engineering Practice*, 7(5):665–671, 1999.

[SASS04] L. Stojanovic, A. Abecker, N. Stojanovic, and R. Studer. Ontology–based correlation engines. In IEEE Computer, editor, *Proceedings of the International Conference on Autonomic Computing (ICAC'04)*, pages 304–305, 2004.

[SB04] C. Scrapper and S. Balakirsky. Knowledge representation for on–road driving. In *Proceedings of the 2004 AAAI Spring Symposium on Knowledge Representation and Ontologies for Autonomous Systems*, Stanford, California, March 2004.

[SBE$^+$03] R. Sanz, J. Bermejo, J. Escasany, R. Chinchilla, and C.A. Garcia. Meaning generation and artificial wisdom. In *Proceeding of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS' 03)*, pages 401–405, Boston, MA, USA, 30 Sept.-4 Oct. 2003.

[SBF98] R. Studer, V.R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *IEEE Transactions on Data and Knowledge Engineering*, 25(1-2):161–197, 1998.

[SBLG08] R. Sanz, J. Bermejo, I. López, and J. Gomez. *Toward Artificial Sapience: Principles and Methods for Wise Systems*, chapter A real-time agent system perspective of meaning and sapience, pages 61–73. Springer London, 2008.

[SBM05] C. Scrapper, S. Balakirsky, and E. Messina. Self awareness in the mobility open architecture simulation and tools framework. In *Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems*, pages 35–41. ACM Press, 2005.

[SBU$^+$03] C. Schlenoff, S. Balakirsky, M. Uschold, R. Provine, and S. Smith. Using ontologies to aid navigation planning in autonomous vehicles. *The Knowledge Engineering Review*, 18(3):243–255, 2003.

[SCR05] M.A. Sicilia, J.J. Cuadrado, and D. RodrÌguez. Ontologies of software artifacts and activities: Resource annotation and application to learning technologies. In *Proceedings of 17th Software Engineering and Knowledge Engineering Conference (SEKE'05)*, pages 145–150, Taipei, Taiwan, July 2005.

[Seg05] M. J. Segarra. *CORBA control systems*. Phd thesis, Universidad Politécnica de Madrid, 2005.

[SHG$^+$09] R. Sanz, C. Hernández, J. Gomez, J. Bermejo-Alonso, M. Rodríguez, A. Hernando, and G. Sanchez. Systems, models and self–awareness: towards architectural models of consciousness. *International Journal of Machine Consciousness*, 1(2):255–279, December 2009.

[SHH$^+$09] R. Sanz, C. Hernández, A. Hernando, J. Gomez, and J. Bermejo. Grounding robot autonomy in emotion and self–awareness. In *Proceedings of the FIRA RoboWorld Congress 2009 on Advances in Robotics*, volume 5744 of *Lecture Notes in Computer Science*, pages 23–43, Incheon, Korea, 2009. Springer-Verlag Berlin Heidelberg.

[SHR07] R. Sanz, C. Hernández, and M. Rodríguez. ASys models: Model–driven engineering in ASys. Technical Report R–2007–016, Autonomous Systems Laboratory (ASLab), August 2007.

[SHR10] Ricardo Sanz, Carlos Hernández, and Manuel Rodriguez. The epistemic control loop. In *Proceedings of CogSys 2010 - 4th International Conference on Cognitive Systems*, Zurich, Switzerland, January 2010.

[SLB$^+$05] R. Sanz, I. López, J. Bermejo, R. Chinchilla, and R.P. Conde. Self-X: The control within. In *Proceedings of the 16th IFAC World Congress*, Praga, Czech Republic, 4-8 July 2005. IFAC.

[SLB07] R. Sanz, I. López, and J. Bermejo. *Artificial Consciousness*, chapter A rationale and vision for machine consciousness in complex controllers, pages 141–155. Imprint Academic, Exeter, UK, 2007.

[SLBH06] R. Sanz, I. López, J. Bermejo, and A. Hernando. Reverse–engineering rat brains. In *Proceedings of the NiSIS Symposium*, Puerto de la Cruz, Spain, November-December 2006. NiSIS.

[SLH07] R. Sanz, I. López, and C. Hernández. Self–awareness in real–time cognitive control architectures. In *Proceedings of the AAAI Fall Symposium on Consciousness and Artificial Intelligence: Theoretical foundations and current approaches*, Washington, D.C., November 2007.

[SM05] C. Schlenoff and E. Messina. A robot ontology for urban search and rescue. In *Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems*, pages 27–34, Budapest, Hungary, November 2005. ACM Press.

[SMP99] R. Sanz, F. Matia, and E. A. Puente. *Microprocessor–based and intelligent systems engineering*, chapter The ICa approach to intelligent autonomous ssytems. Kluwer Academic Publishers, 1999.

[Soc04] IEEE Computer Society. Guide to the software engineering: Body of knowledge 2004 version, 2004.

[SR08] R. Sanz and M. Rodríguez. The ASys vision: Engineering any-x autonomous system. Technical Report R-2007-001, Autonomous Systems Laboratory (ASLab), February 2008.

[SRKR97] B. Swartout, P. Ramesh, K. Knight, and T. Russ. Toward distributed use of large–scale ontologies. In A. Farquhar, M. Grüninger, A. Gómez-Pérez, M. Uschold, and P. van der Vet, editors, *AAAI'97 Spring Symposium on Ontological Engineering*, pages 138–148, Stanford University, CA, U.S.A., 1997.

[SS01] Y. Sure and R. Studer. On–to–knowledge ontoedit. Technical Report OTK-2001-D3-v1.0, Institute AIFB, University of Karlsruhe, May 2001.

[SSM$^+$04] L. Stojanovic, J. Schneider, A. Maedche, S. Libischer, R. Studer, Th. Lumpp, A. Abecker, G. Breiter, and J. Dinger. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, 43(3):598 – 616, 2004.

[SSSS01] S. Staab, H.P. Schnurr, R. Studer, and Y. Sure. Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[SV06] T. Stahl and M. Völter. *Model-Driven Software Development: technology, engineering, management.* John Wiley and Sons, Ltd., 2006.

[SWM04] M.K. Smith, C. Welty, and D. L. McGuiness. OWL Web Ontology Language guide. http://www.w3.org/TR/owl-guide, February 2004.

[SZ03] Ricardo Sanz and Janusz Zalewski. Pattern-based control systems engineering. *IEEE Control Systems Magazine*, 23(3):43–60, June 2003.

[Tam01] V. Tamma. *An Ontology Model supporting Multiple Ontologies for Knowledge Sharing*. Phd, University of Liverpool, 2001.

[Tan05] V. Tanasescu. An ontology–driven life–event portal. Master, Computer Science, 2005.

[TCFW05] V. Tamma, S. Cranefield, T.W. Finin, and S. Willmott, editors. *Ontologies for Agents: Theory and Experiences*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser, 2005.

[Tea] ASLab Team. ASLab wiki on the ASLab intranet. www.aslab.org.

[TLP$^+$05a] L. Tang, H. Li, Z. Pan, S. Tan, B. Qiu, S. Tang, and J. Wang. Podwis: A personalized tool for ontology developing in domain specific web information systems. In *Proceedings of 7th Asia Pacific Web Conference (APWeb 2005)*, Shanghai, China, 2005.

[TLP$^+$05b] L. Tang, H. Li, Z. Pan, D. Yang, M. Li, S. Tang, and Y. Ying. An ontology based approach to construct behaviors in web information systems. In *Proceedings of 6th International Conference on Web-Age Information Management (WAIM 2005)*, Hangzhou, China, October 2005.

[TLQ$^+$06] L. Tang, H. Li, B. Qiu, M. Li, J. Wang, L. Wang, B. Zhou, D. Yang, and S. Tang. Wise: a prototype for ontology driven development of web information systems. In X. Zhou et al., editor, *Proceedings of 8th Asia Pacific Web Conference (APWeb 2006)*, number 3841 in LNCS, pages 1163–1167, Harbin, China, January 2006. Springer-Verlag Berlin Heidelberg.

[TT03] G. Tziallas and B. Theodoulidis. Building autonomic computing systems based on ontological component models and a controller synthesis algorithm. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, pages 674–680, Prague, Czech Republic, September 2003.

[TvW98] C. Tautz and C. G. von Wangenheim. REFSENO: A representation formalism for supporting reuse of software engineering knowledge. Technical Report 015.98/E, Version 1.1, Fraunhofer IESE, October 1998.

[UG96] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[UIT06a] UPM-ICEA-Team. Case studies of perception and system analysis. Technical Report ASLab-ICEA-R-2006-015, 1.0 Final, Autonomous Systems Laboratory (ASLab), December 2006.

[UIT06b] UPM-ICEA-Team. ICEA glossary: integration, cognition, emotion, autonomy. Technical Report ASLab-ICEA-R-2006-014, Autonomous Systems Laboratory (ASLab), April 2006.

[UIT06c] UPM-ICEA-Team. UPM contribution to A1-D1, initial theoretical framework. Technical Report A1-WP1-D1-UPM, Autonomous Systems Laboratory (ASLab), August 2006.

[UIT06d] UPM-ICEA-Team. A vision of general autonomous systems. Technical Report ASLab-ICEA-R-2006-018, 1.0 Final, Autonomous Systems Laboratory (ASLab), November 2006.

[UIT06e] UPM-ICEA-Team. A vision of perception in autonomous systems. Technical Report ASLab-ICEA-R-2006-017, Autonomous Systems Laboratory (ASLab), December 2006.

[UK95] M. Uschold and M. King. Towards a methodology for builiding ontologies. In D. Skuce, editor, *Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 6.1–6.10, Montreal, Canada, 1995.

[UPS$^+$03] M. Uschold, R. Provine, S. Smith, C. Schlenoff, and S. Balikirsky. Ontologies for world modeling in autonomous vehicles. In *18Th International Joint Conference on Artificial Intelligence, IJCAI'03*, 2003.

[VAO$^+$05] A. Vizcaíno, N. Anquetil, K. Oliveira, F. Ruiz, and M. Piattini. Merging software maintenance ontologies: Our experience. In *Proceedings of 1st Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering (ONTOSE)*, volume 163, Alcalá de Henares, Madrid, Spain, June 2005. CEUR Workshop Proceedings.

[vB50] L. von Bertalanffy. An outline of general systems theory. *British Journal of the Philosophy of Science*, 1:pp. 134–164, 1950.

[vB68] L. von Bertalanffy. *General Systems Theory.* Braziller, New York, 1968.

[VB96] A. Valente and J. Breuker. Towards principled core ontologies. In B.R. Gaines and M. Mussen, editors, *Proceedings of the KAW-96*, Banff, Ca., 1996.

[vHSW97] G. van Heijst, A.Th. Schreiber, and B.J. Wielinga. Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 45:184–292, 1997.

[vL08] A. van Lamsweerde. Requirements engineering: From craft to discipline. In *Proceedings of FSE'2008: 16th ACM Sigsoft International Symposium on the Foundations of Software Engineering*, Atlanta, U.S.A, November 2008.

[VS02] U. Visser and C. Schlieder. Modelling real estate transactions: The potential role of ontologies. In *The Ontology and Modelling of Real Estate Transactions in European Juristictions*, 2002.

[Wan07a] Y. Wang. Perspectives on autonomic computing. *International Journal of Cognitive Informatics and Natural Intelligence*, 1(3):1–6, 2007.

[Wan07b] Y. Wang. Toward theoretical foundations of autonomic computing. *International Journal of Cognitive Informatics and Natural Intelligence*, 1(3):1–16, 2007.

[WCC05] P. Wongthongtham, E. Chang, and C. Cheah. Software engineering sub-ontology for specific software development. In *Proceedings of 29th IEEE/NASA Software Engineering Workshop (SEW'05)*, pages 27–33, Greenbelt, Maryland, April 2005.

[WCD05] P. Wongthongtham, E. Chang, and T.S. Dillon. Towards ontology-based software engineering for multi-site software development. In *Proceedings of 3rd IEEE International Conference on Industrial Informatics (INDIN)*, pages 362–365, Perth, Australia, August 2005.

[WCH02] X. Wang, C.W. Chan, and H.J. Hamilton. Design of knowledge–based systems with the ontology–domain–system approach. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02*, pages 233–237, Ischia, Italy, July 15-19 2002.

[WLB04] R.E. Wray, S.A. Lisse, and J.T. Beard. Ontology infrastructure for execution–oriented autonomous agents. *Robotics and Autonomous Systems*, 2004:113–122, 2004.

[Woo04] S. Wood. Representation and purposeful autonomous agents. *Robotics and Autonomous Systems*, 49:79–90, 2004.

[WVH04] T. Wagner, U. Visser, and O. Herzog. Egocentric qualitative spatial knowledge representation for physical robots. *Robotics and Autonomous Systems*, 49:25–42, 2004.

[WVV$^+$01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information – a survey of existing approaches. In H. Stuckenschmidt, editor, *(IJCAI'01) Workshop: Ontologies And Information Sharing*, pages 108–117, 2001.

[ZHSL06] T. Ziemke, C. Herrera, R. Sanz, and I. López. ICEA Deliverable D2, Theoretical background document. Technical Report ICEA-R-2006-012v1, HIS and UPM, December 2006.