2023/2024 2Q

Transparencias del Curso

ECSDI

Javier Béjar

Departament de Ciències de la Computació

Grau en Enginyeria Informàtica - UPC

FIB Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÁTICA DE BARCELONA
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Primera edición, Febrero 2013
Esta edición, Febrero 2024

Índice general

Prefacio

Esta es la recopilación de las transparencias del curso Enginyeria del Coneixement i Sistemes Distribuïts Intel.ligents (ECSDI), asignatura complementaria de la especialidad de ingeniería de software del grado en informática de la Facultat d'Infomàtica de Barcelona (UPC).

Las transparencias separadas se pueden obtener en la pagina de web:

https://sites.google.com/upc.edu/ecsdi/ecsdi-gei

Es un trabajo abrumador el preparar una asignatura que incluye temas tan diversos a caballo entre la ingeniería de software orientada a servicios y el desarrollo de sistemas multiagentes. Va por adelantado mi agradecimiento a los múltiples autores de todo el material que he utilizado para elaborar este material. A continuación cito las fuentes principales en las que me he basado, pero me dejo a mucha gente que me ha inspirado de alguna manera en la forma en que he organizado el material y el curso.

El material introductorio tiene como fuentes los documentos del grupo de interés de la Unión Europea sobre la Future Internet Initiative y material usado en presentaciones de diferentes proyectos europeos en el área de agentes inteligentes que ha elaborado Javier Vázquez, profesor del departamento de ciencias de la computación de la UPC.

El capítulo relacionados con Arquitecturas Orientadas a Servicios (SOA) provienen de los documentos de los estandares de OASIS y el W3C sobre SOA y web services (SOA Reference Model, SOA Reference Architecture Foundation y Webservices Architecture).

El capítulo sobre fundamentos de sistemas multiagentes esta tomado completamente del libro *An Introduction to Multiagent Systems* de Michael Woolridge y sus transparencias.

El capítulo de ingeniería de software orientada a agentes tiene múltiples fuentes pero usa como base el artículo *Agent Oriented Software Engineering* de Nicolas Jennings y Michael Woolridge.

El capítulo de la metodología Prometheus se basa en el libro *Developing Intelligent Agent Systems* de Lin Padgham y Michael Winikoff y el material que han usado para enseñar la metodología. El ejemplo de PACMAN se basa en trabajos de los estudiantes del cursos de sistemas multiagente del master de Inteligencia Artificial de la UPC que enseña el profesor Javier Vázquez.

El capítulo sobre comunicación en sistemas multiagente proviene de los diversos artículos de la primera edición del libro *Multiagent Systems* editado por Gerhard Weiss y de los documentos sobre protocolos de comunicación entre agentes del estándar de FIPA.

El capítulo de introducción a las ontologías proviene en parte del material sobre representación del conocimiento que se usa en la asignatura Intel.ligència Artificial del grado en informática de la FIB elaborado por mi mismo y otros profesores que han pasado por la asignatura. El resto proviene de múltiples fuentes.

El capítulo de desarrollo de ontologías se basa en el artículo *Ontology Development 101: A Guide to Creating Your First Ontology* de Natalya Noy y Deborah McGuinness.

El capítulo de lenguajes de ontologías y web semantica se nutre principalmente de los documentos de los diferentes estándares del W3C para la web semántica sobre RDF, RDFS, OWL y SPARQL.

El capítulo de organización en sistemas multiagentes proviene también de capítulos del libro *Multiagent Systems* y del libro *An Introduction to Multiagent Systems*.

El capítulo de composición de servicios proviene de múltiples fuentes que describen diferentes tecnologías y estándares como SOAML, WS-BPEL o WS-CDL.

El capítulo de composición dinámica proviene en su primera parte de diferentes artículos sobre composición dinámica y de la descripción de la ontología de descripción de servicios OWL-S. La parte de planificación es una interpretación de los primeros capítulos de planificación del libro *Artificial Intelligence: A Modern Approach* de Peter Norvig y Stuart Russell sacada de material elaborado por Javier Vázquez y yo mismo.

El capítulo de aprendizaje y personalización es una versión concentrada de parte del material que se usaba para la asignatura Aprenentatge de la antigua ingeniería superior en informática de la FIB elaborado por mi mismo. La parte de sistemas recomendadores proviene del libro *Recommender Systems Handbook* de su cuarto capítulo escrito por Christian Desrosiers y George Karypis.

Javier Béjar
Barcelona, 2024

1 Introduccion

Introducción

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Motivación de la necesidad de sistemas distribuidos
- Características de los sistemas distribuidos
- Orientación a servicios (SOA/Microservicios)
- Inteligencia artificial como metodología de desarrollo de Sistemas Distribuidos

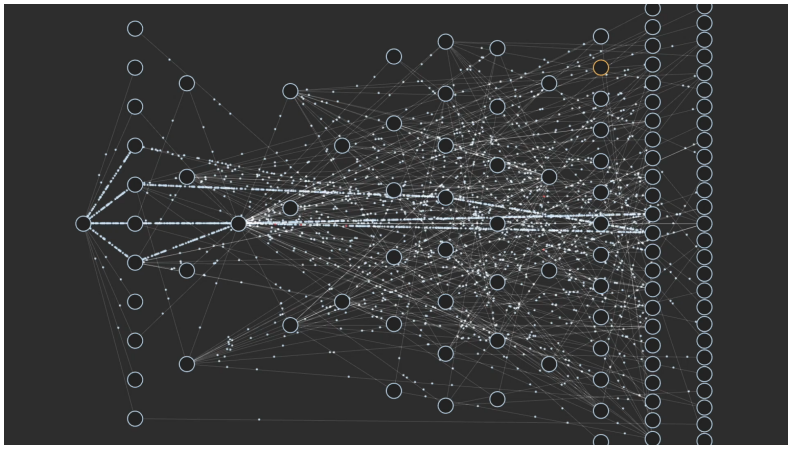
Los sistemas software han dejado de ser entornos cerrados y monolíticos

- El elemento del sistema es el servicio
- Múltiples componentes y sistemas colaboran para realizar tareas complejas

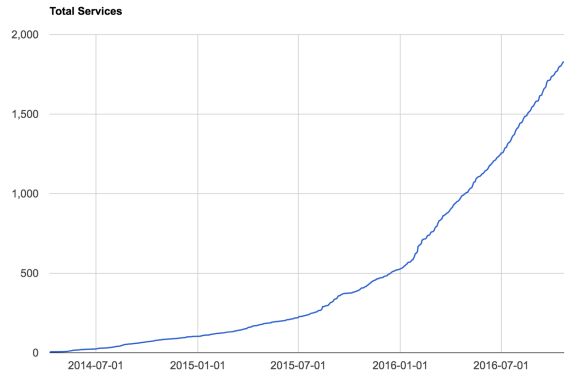
- Distribución geográfica
- Necesidad de comunicarse de maneras complejas
- Necesidad de organizarse/colaborar/coordinarse
- Interacciones dinámicas y cambiantes (colaboración con múltiples entidades)
- Autonomía (toman sus propias decisiones)

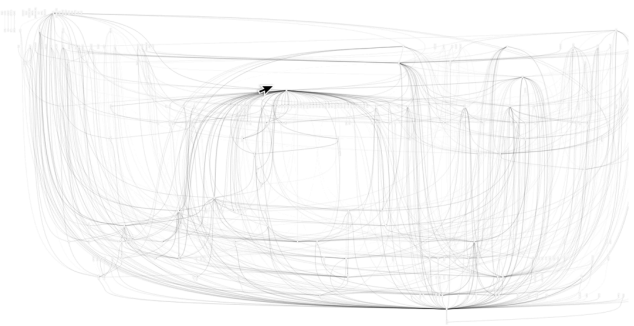
- **Next Generation Internet Initiative** (2017-2030)
- Estudiar los cambios que producirá en la sociedad
- Estudiar el impacto qué tendrá en como las empresas desarrollarán su actividad
- Estudiar las oportunidades económicas y de innovación
- Estudiar las necesidades tecnológicas y de investigación a distintos niveles (software, hardware, comunicaciones)
- Promover estándares para facilitar la adopción de las tecnologías

- La vida real y digital estarán más entrelazadas
- Virtualización de las empresas, transformando los roles de clientes y empleados
- Nuevas herramientas y métodos para desarrollar y verificar sistemas interconectados y seguros
  - **Nuevas maneras de desarrollar software**



Mastering Chaos - A Netflix Guide to Microservices
https://www.youtube.com/watch?v=CZ3wIuvmHeM





What Comes after Microservices? (Uber)
https://www.youtube.com/watch?v=UDC3kwkBvkA

- Un **sistema distribuido** es un conjunto de sistemas computacionales que se comunican a través de una red
- Los diferentes sistemas que los componen interaccionan entre ellos para conseguir un objetivo (común)
- La tarea a realizar no tiene por qué involucrar inteligencia o replicar mecanismos congnitivos complejos
- La **inteligencia artificial distribuida** es la parte de la IA que aporta soluciones a la construcción de sistemas distribuidos complejos

- Una forma de repartir la carga de trabajo entre un conjunto de computadores
- Una manera de ahorrar recursos
- Una posible manera de mejorar la eficiencia del hardware

- Un sistema distribuido aparece como una sola unidad desde el exterior
- Es escalable, se pueden añadir nuevos componentes y nuevas capacidades
- Permiten coordinar sistemas que están separados físicamente
- Permiten ser tolerante a fallos (los componentes pueden ser reemplazados)

- **Heterogeneidad**, los componentes computaciones hardware y software pueden ser muy diferentes
- Es difícil gestionar **fallos múltiples**
- **Seguridad**, hay múltiples puntos de entrada
- **Fiabilidad**, depende de la fiabilidad de la red de conexión
- **Latencia**, la comunicación introduce retrasos y bloqueos
- **Complejidad de diseño**

- Perspectiva de desarrollo de sistemas distribuidos desde el punto de vista de la ingeniería de software
- La orientación a servicios está pensada a la vez como diseño arquitectónico y metodología de desarrollo de software.
- Basada en el concepto de servicio como elemento de diseño
- Servicio: Componente software que provee sus funcionalidades a otras aplicaciones
- Metodología independiente de la tecnología de implementación

- Cada servicio está diseñado para hacer una actividad:

  Servicio = Un fragmento de código

- El reuso solo implica cambiar cómo se interacciona con otros servicios (vs cambiar el código del servicio)
- En lugar de una API, un servicio define una interfaz en términos de protocolos y funcionalidad
- Se puede ver SOA como una evolución natural de la computación distribuida y la programación modular

- Un servicio es auto descriptivo
- Implementa una funcionalidad bien definida
- Usa el encapsulamiento de información como principio de diseño (caja negra)
- Es independientes de su plataforma de implementación
- Pensado para operar en red
- Pensado para facilitar la integración de aplicaciones/organizaciones

- **Acoplamiento débil**: Minimización de dependencias, conocimiento básico sobre la funcionalidad de los otros
- **Abstracción**: Lógica de funcionamiento no visible más allá de entradas y salidas
- **Contratos entre servicios**: Comunicación entre servicios descrita de manera global mediante un documento público
- **Reusabilidad**: La funcionalidad está diseñada para poder formar parte de diferentes problemas

- **Composabilidad**: El acoplamiento de servicios permite proveer servicios más complejos
- **Autonomía**: El servicio tiene control sobre la funcionalidad que provee
- **Sin estado**: El servicio guarda la mínima información para su funcionamiento (estado a nivel superior es responsabilidad de otros)
- **Descubrimiento**: La descripción de los servicios provee la información que les permite ser descubiertos (qué hacen) e interpretados (cómo invocarlos)

Una arquitectura orientada a servicios define

- Un conjunto de buenas prácticas
- Un conjunto de principios de diseño
- Un conjunto de patrones de diseño

Objetivo: Guiar el diseño orientado a servicios para poder desarrollar software más flexible y complejo

(!) Microservicios vs SOA

Definición

"Un campo de estudio que busca explicar y emular el comportamiento inteligente en términos de procesos computacionales" (Schalkoff, 1990)

- Adoptando metáforas de IA que acerquen el desarrollo a como pensamos
- Acercando la interacción entre sistemas software a como interactuamos
- Adoptando una visión declarativa en la resolución de problemas (qué hacer vs cómo hacerlo)
- Permitiendo sistemas software adaptativos que puedan resolver requerimientos/situaciones/interacciones no previstas

La Inteligencia Artificial Distribuida (DAI) y los Sistemas Multiagente aporta soluciones a problemas complejos de los sistemas distribuidos, Puede utilizarse como marco general de desarrollo de software

- Semántica de comportamiento
- Descripción, organización y coordinación de componentes
- Toma de decisiones en entornos heterogéneos y abiertos
- Descubrimiento y composición dinámica
- Recuperación de fallos

- La complejidad de los sistemas distribuidos exige que tratemos los datos de manera más compleja y desde diferentes puntos de vista
- La representación del conocimiento estudia los formalismos para representar la información en sistemas inteligentes
- La representación del conocimiento permite describir:
  - La comunicación, interacción, objetivos y funcionamiento de los servicios
  - El contexto del servicio
  - El razonamiento sobre todos estos elementos

- Los sistemas distribuidos se componen de elementos heterogéneos con cambios dinámicos en sus interacciones (no podemos preprogramar todo)
- La toma de decisiones es compleja y requiere el uso de conocimiento declarativo
- La inteligencia artificial provee de mecanismos que permiten tratar esas interacciones y la toma de decisiones
  - Planificación Automática: Generación de composiciones de tareas simples en tareas más complejas dado un objetivo, programación automática
  - Razonamiento Automático: Toma de decisiones dinámica a partir de conocimiento declarativo

- Las aplicaciones evolucionan, las interacciones con otras entidades cambian, las interacciones requieren personalización/adaptación
- No podemos programar para cambios que no podemos prever, ni programar todas las posibles variedades de interacciones existentes
- Aprendizaje Automático: adaptación del comportamiento a diferentes niveles, por ejemplo:
  - Perfilado y preferencias de los servicios que interaccionan
  - Aprendizaje de resolución de tareas sin programación previa
  - Adaptación a los cambios en el contexto/entorno

2

Ingeniería de Software en SOA

Ingeniería de Software en Servicios

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Ingeniería de Servicios SOA
- Modelo de referencia SOA (OASIS)
- Arquitectura de referencia SOA (OASIS)
- Arquitectura de servicios web (W3C)

Directrices para la IS en SOA

Modelo de referencia SOA (OASIS)

Aquitectura de referencia SOA (OASIS)

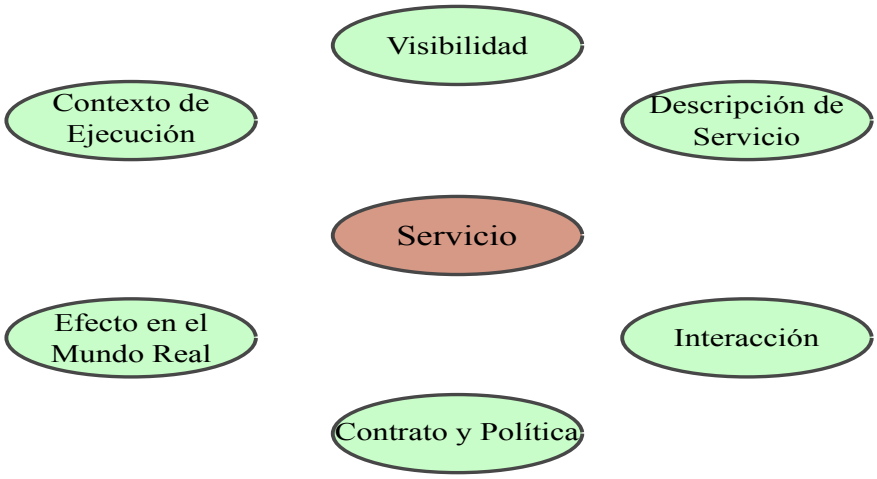
SOA y microservicios

Arquitectura de servicios web (W3C)

- Hasta no hace mucho la Ingeniería de Software ha visto los sistemas computacionales como entidades monolíticas
  - El análisis ve el sistema completo como una caja negra
  - A partir de ahí se especifican las funcionalidades
  - A posteriori se pueden distribuir funcionalidades de acuerdo con requisitos funcionales o no funcionales
- Problema: La distribución estructural y funcional se aborda demasiado tarde

- Grupos de estandarización han definido directrices sobre el desarrollo de Arquitecturas Orientadas a Servicios de una manera abstracta
- OASIS
  - Reference Model for Service Oriented Architecture 1.0 (10/2006)
  - Reference Architecture Foundation for Service Oriented Architecture 1.0 (12/2012)
- W3C
  - W3C Web Services Architecture (2/2004)

- Modelo abstracto para el desarrollo de Arquitecturas Orientadas a Servicios
- Marco de referencia para entender las relaciones entre los elementos de un entorno SOA
- El modelo de referencia define SOA en un sentido abstracto
  - Por ejemplo, indica que se puede tener una descripción de servicio, pero no su realización concreta, por ejemplo usando WSDL (Web Service Description Language)



¿Que es el SOA Reference Arquitecture Foundation?

- Supone un paso de refinamiento sobre el reference model
- Entra en los detalles de los conceptos de SOA-RM
- Muestra como los sistemas SOA pueden ser implementados, planteado desde un nivel abstracto y de manera independiente de las tecnologías usadas
- Da una perspectiva de SOA desde el punto de sistemas dinámicos en lugar de un sistema software estático

SOA-RAF: Ecosistema SOA

- Uso de recursos que están distribuidos más allá del entorno cercano de los actores
- Personas y sistemas interaccionando entre ellos también más allá de su entorno cercano
- Seguridad, manejo de recursos y gobernanza distribuidas
- Interacción a través de un intercambio de mensajes con fiabilidad y apropiado para las intenciones y usos establecidos

SOA-RAF: 3 Puntos de vista

1. Participación en un ecosistema SOA, captura la visión como un entorno para personas que realizan actividades que involucran múltiples participantes
2. Realización del ecosistema SOA, captura los elementos de infraestructura que son necesarios para soportar la construcción de sistemas SOA
3. Propiedad en un sistema SOA, captura los elementos necesarios para definir la propiedad y la gestión del sistema, como se toman las decisiones, como se promulgan y como se hacen efectivas

Problemas de SOA

- SOA es complicada de llevar a la práctica
- Muchos problemas no requieren la complejidad que representa
- Las implementaciones de SOA son complicadas de desplegar y mantener
- Las empresas líder en desarrollo de software no han convergido en una estandarización suficiente

Microservicios

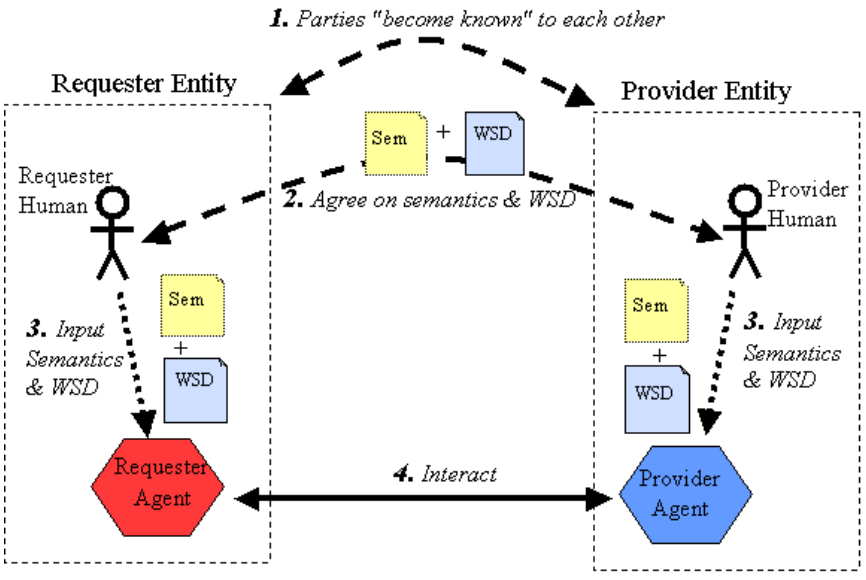
- Un microservicio resuelve una tarea simple (¡deja vu!)
- Están al mínimo nivel de abstracción/granularidad (no son composición de otros)
- Son autónomos, deben necesitar mínima coordinación para realizar su tarea (o ninguna)
- No dependen de una infraestructura que los conecta (middleware)

- Los microservicios son los elementos sobre los que se construyen servicios más complejos
- Necesitarán: coordinación, comunicación, interoperabilidad, seguridad, contratos, regulaciones
- Implementaciones de microservicios utilizan: sistemas de mensajes (comunicación), servicios de descubrimiento/coordinación, seguridad/encriptación, ...

- Los servicios web pueden verse como un paso más en la concreción del desarrollo
- Un servicio web es una noción abstracta que se implementa a través de un agente
- Servicio web: Recurso descrito por un conjunto de funcionalidades abstractas
- Agente: código software/hardware concreto que envía y recibe mensajes
  - Proveedor (provider): Organización que pone a disposición el agente que presta un servicio
  - Solicitador (requester): Organización que desea usar un servicio y provee el agente que intercambia mensajes con el agente del proveedor

- **Descripción de servicio**
  - Los intercambios de mensajes entre servicios están documentados en una descripción de servicio
  - Este documento define todo lo necesario para la interacción
  - Está descripción se hace utilizando WSDL
  - Opcionalmente puede incluir una descripción funcional

- **Semántica**
  - La semántica está definida por las expectativas acerca del comportamiento del servicio
  - Supone un contrato entre partes sobre el propósito y consecuencias de la interacción
  - No se define la forma de su realización, pudiendo estar documentada o no, ser implícita o explícita, formal o informal, negociada o no negociada



¿Son suficientes estos modelos y arquitecturas?

- SOA-RM, SOA-RAF y W3C-WSA solo describen los conceptos y elementos que aparecen en SOA
- No dan ninguna guía sobre cómo distribuir funcionalidades
- Las metodologías de ingeniería de software estándar no dan una guía de como desarrollar sistemas completamente distribuidos
- ¿Dónde debemos mirar para obtener esas guías?

3

Sistemas Multiagente

Sistemas Multiagente

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Sistemas Multiagente
- Definición/Propiedades de los Agentes Inteligentes
- Arquitecturas de Agentes
- Arquitecturas de Sistemas Multiagente

Introducción

Tendencias en la historia de la computación

- Ubicuidad
- Interconexión
- Inteligencia
- Delegación
- Orientación a las personas

("Introduction to Multi-Agent Systems", M. Wooldridge, 2001)

- **Ubicuidad**
  - La capacidad de cómputo se ha incluido en multitud de elementos gracias a su abaratamiento (IoT)
  - El aumento de su potencia permite que la sofisticación (e inteligencia) sea ubicua
- **Interconexión**
  - Los sistemas ya no son elementos aislados, están conectados en red formando sistemas distribuidos
  - Esto lleva a la idea de modelar la computación como un proceso de interacción (computation as interaction)

- **Inteligencia**
  - La complejidad de las tareas automatizables ha ido creciendo (llegando a poder considerarse *inteligentes*)
- **Delegación**
  - Más tareas se realizan automáticamente (sin supervisión), incluso en tareas críticas
- **Orientación a las personas**
  - La creación de programas se basa en abstracciones y metáforas de cada vez más alto nivel (menos centrada en el computador, más cercana a nuestra visión)

Agente: primera definición

Un sistema computacional capaz de actuar de manera independiente como representante de su usuario (satisfaciendo unos objetivos de diseño y sin supervisión)

Sistema multiagente

Un sistema computacional compuesto de múltiples agentes que interaccionan entre ellos

- Cada agente tendrá sus propios objetivos y motivaciones
- El éxito de la interacción requerirá de cooperación, coordinación y negociación (precisamente las cualidades de las personas)

- Esta metáfora nos lleva a plantear los sistemas software basados en agentes desde dos perspectivas: Individuos y sociedades
  1. **Diseño de agentes**: ¿Cómo diseñamos agentes capaces de resolver de manera autónoma las tareas que se les delegan?
  2. **Diseño de sociedades**: ¿Cómo diseñamos agentes capaces de interaccionar con otros de manera que resuelvan sus tareas, especialmente en caso de objetivos conflictivos?

Agente: segunda definición

Sistemas computacionales capaces de realizar acciones de manera autónoma en algún entorno, con el propósito de alcanzar una serie de objetivos que tiene delegados

- El principal interés de los agentes es que son autónomos (capaces de actuar de manera independiente)
- Un agente está fuertemente ligado y en continua interacción con su entorno:

  percepción $\mapsto$ decisión $\mapsto$ acción $\mapsto$ percepción $\mapsto$ decisión $\mapsto$ ...

- Agentes Simples: Termostato, demon unix
- No estamos interesados en agentes simples

Agente: tercera definición

Sistemas computacionales capaces de realizar acciones de manera autónoma y **flexible** en algún entorno, con el propósito de alcanzar una serie de objetivos que tiene delegados

Entendiendo como flexible: **Reactivo, proactivo y social**

- Los entornos reales son dinámicos, sus elementos cambian, su información es incompleta y/o incierta
- En un entorno fijo, un agente no debe preocuparse del resultado de sus acciones, puede actuar sin pensar en las consecuencias
- En los entornos dinámicos se ha de tener en cuenta la posibilidad de resultados no esperados, es posible plantearse no realizar una acción siempre
- Un sistema reactivo ha de mantener una interacción continua con el entorno y responder a los cambios adecuadamente (reflexión)

- Reaccionar al entorno es fácil, pero queremos que los agentes hagan cosas por nosotros, esto implica un comportamiento dirigido por objetivos
- Proactividad = Generar e intentar cumplir objetivos, no estar dirigidos únicamente por eventos, tomando la iniciativa
- Esto implica poder y saber reconocer oportunidades, se ha de saber cuándo se puede actuar y planificar las acciones en consecuencia
- Proactividad significa tener una visión a largo plazo que nos lleva a cumplir los objetivos delegados

Reactividad vs Proactividad

- Reactividad y proactividad son propiedades que pueden interferir entre ellas
- Necesitamos que un agente reaccione apropiadamente a los cambios en el entorno (reactividad)
- Necesitamos que un agente sea capaz de cumplir objetivos a largo plazo (proactividad)
- Una de las dificultades de los sistemas distribuidos es el conseguir una combinación adecuada de ambas propiedades

Habilidad Social

- El mundo real es un entorno multi-agente, no es posible obtener los objetivos propios sin considerar los de otros
- Algunos objetivos solo se pueden cumplir con la interacción de otros
- La habilidad social en agentes es la capacidad de interactuar con otros agentes (incluidos humanos) vía cooperación, coordinación y negociación
- Esto implica el tener que usar algún lenguaje de comunicación

Agentes (AOP) vs Objetos (OOP)

Objetos y agentes:

- Encapsulan un estado
- Se comunican por paso de mensajes
- Tienen métodos que corresponden con las acciones que se pueden realizar según su estado

Pero los agentes son:

- Autónomos, deciden por si mismos si actúan o no
- Inteligentes, capaces de comportamientos flexibles
- Activos

> "Los objetos lo hacen gratis, los agentes porque quieren y por dinero"
>
> (M. Woolridge, Introd. to Multiagent Systems)

- **Arquitecturas reactivas puras**
  - Los agentes poseen sensores y actuadores conectados al entorno y su conducta se basa en estímulo-respuesta
  - Deciden sus acciones independientemente de la historia
  - Sus decisiones se basan en reglas simples que hacen coincidir las observaciones del estado con las decisiones
  - La conexión entre sensores-actuadores hace emerger una conducta inteligente

- **Arquitecturas reactivas con estado interno**
  - Los agentes tienen una estructura interna que representa información del estado y su historia
  - Poseen una función que percibe e interpreta el entorno:

$$Observar : Entorno \Rightarrow Percepcion$$

  - El estado interno es usado para decidir la actuación:

$$Accion : Estado \Rightarrow Actuacion$$

  - El estado es actualizado combinando la percepción y el estado interno actual:

$$Siguiente : Estado \times Percepcion \Rightarrow Estado$$

- **Arquitecturas deliberativas** (orientadas por objetivos)
  - Poseen una representación interna del mundo, siguen una aproximación simbólica y su funcionamiento se basa en el razonamiento (lógicas)
  - Los agentes se implementan para que puedan resolver tareas especificadas por nosotros
  - Queremos decirle al agente qué hacer, pero no cómo hacerlo (declarativo vs imperativo)
  - El agente debe tener capacidad para elegir sus objetivos y como conseguirlos

- Basadas en una visión simbólica de la IA
- Las decisiones se basan en la lógica simbólica
- Se tiene un modelo del entorno y se actúa según ese conocimiento
- Problemas:
  - Cómo representar del mundo exterior a partir de formalismos lógicos (transductor problem)
  - Cómo resolver el proceso de razonamiento que lleva a las decisiones (representation/reasoning problem)

- Razonamiento basado en el denominado razonamiento práctico
- Modela el proceso que realizamos al decidir qué acción llevamos a cabo cada momento para perseguir unos fines (análisis medios - fines)
- Basado en dos procesos
  - Decidir qué objetivos queremos conseguir (Deliberación)
  - Decidir cómo conseguirlos (Razonamiento de medios fines)
- Los agentes se definen en función de:
  - Creencias (Beliefs): Cuál es mi visión del mundo
  - Deseos (Desires): Qué opciones tengo según mis creencias
  - Intenciones (Intentions): Qué objetivos voy a perseguir

- La interacción social entre agentes obliga a definir y desarrollar arquitecturas que soporten esta dimensión
- Es necesaria una capa intermedia entre los agentes que permita la interconexión/organización/comunicación
- Este software de soporte se denomina **plataformas de agentes**
- Este middleware correspondería a la infraestructura definida por SOA RM/RAF

- **FIPA** (Foundation for Intelligent Physical Agents) es un grupo de estandarización de IEEE que ha definido un conjunto de estándares sobre agentes
- FIPA definió una arquitectura abstracta que debería seguir toda implementación de una plataforma multiagente
- Está compuesta por:
  - Un directorio de agentes
  - Un directorio de servicios
  - Un mecanismo de transporte de mensajes
  - Un lenguaje de comunicación de agentes (ACL)

4

Ingeniería de Software Orientada a Agentes

Ingeniería de software orientado a agentes

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Software y sistemas complejos
- Ingeniería de Software orientada a agentes
  - Abstracciones, Autonomía, Interacción, Organización
- Medodologías orientadas a agentes - Proceso/fases

Ingeniería de software

Sistemas Complejos

Ingeniería de software orientada a agentes

Metodologías de software orientadas a agentes

- El desarrollo de aplicaciones software cada vez más complejas hace que se necesiten mayores niveles de abstracción
- Conceptos, modelos, metodologías, tecnologías y herramientas evolucionan forzando a afrontar cambios radicales en las formas de desarrollar software
- El paradigma de orientación a objetos no puede ser considerado la última respuesta en esta tendencia, es solo un paso más

Sistemas computacionales

- **Pasado:** Sistemas centralizados, modelo de programación monolítico
- **Presente:** Sistemas distribuidos, heterogéneos, escalables, abiertos, modelo de programación distribuido

Desarrollo de software

- **Pasado:** Modelo de desarrollo en cascada
- **Presente:** Modelo incremental, ágil, procesos de desarrollo experimentales

- El desarrollo de software basado en agentes se plantea como una nueva perspectiva para el desarrollo de sistemas software
- La **orientación a agentes** subsume los conceptos soportados por los previos paradigmas de programación y en particular los de la programación orientada a objetos.
  - Elevan el nivel de abstracción
  - Son una aproximación más adecuada para el desarrollo de software complejo

- Los sistemas complejos están formados por una jerarquía de subsistemas interrelacionados
- La elección de los componentes primitivos es relativamente arbitraria (objetivos y necesidades)
- En la jerarquía podemos distinguir entre:
  - Interacciones intra-sistema (más frecuentes y predecibles)
  - Interacciones inter-sistema (menos frecuentes)

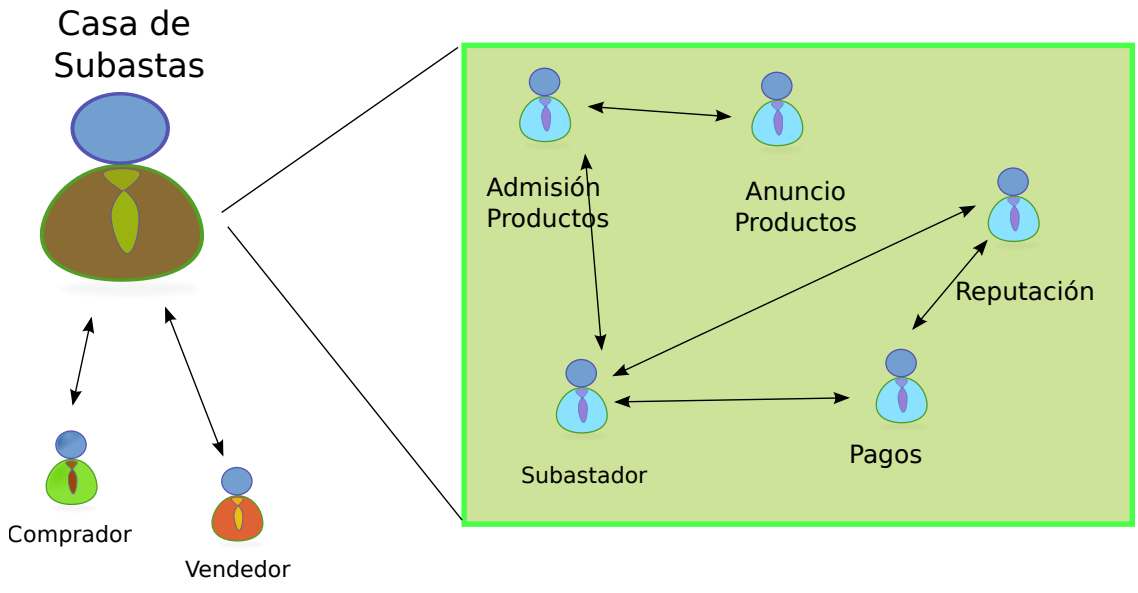
- Esto hace que sean **casi-separables**
- Lo que no los hace totalmente separables son las interacciones inter-sistema
- Algunas de estas interacciones no son predecibles en tiempo de diseño

- Los problemas complejos son descentralizados
- Con múltiples puntos de control de ejecución (subproblemas)
- Con múltiples perspectivas sobre el problema y múltiples objetivos (según los subsistemas)
- Los diferentes subsistemas deben interaccionar para obtener sus objetivos y resolver sus dependencias
  - A través de mensajes (de alto a bajo nivel)
  - Mediante interacciones sociales (coordinación, cooperación, negociación)

- **Abstracciones:**
  - Componentes autónomos que buscan unos objetivos
  - Componentes que interaccionan a alto nivel
  - Componentes que se organizan socialmente
  - Componentes que pueden cambiar su relaciones dinámicamente
  - Componentes que se pueden ver a diferentes niveles de granularidad
- El desarrollo se basa en la agregación de componentes de manera jerárquica con una filosofía **de abajo a arriba**

- Asignamos TAREAS al software (p. ej.: Software para subasta de mercancías)
  - El **qué** y el **cómo** se especifican por adelantado (casos de uso, escenarios)
  - No son tolerables los cambios en los requerimientos
- Asignamos ROLES a los agentes (p. ej.: Agente subastador)
  - El **qué** es especificado por adelantado, el **cómo** se determina dinámicamente (librería de métodos)
  - Son tolerables cambios en los requerimientos

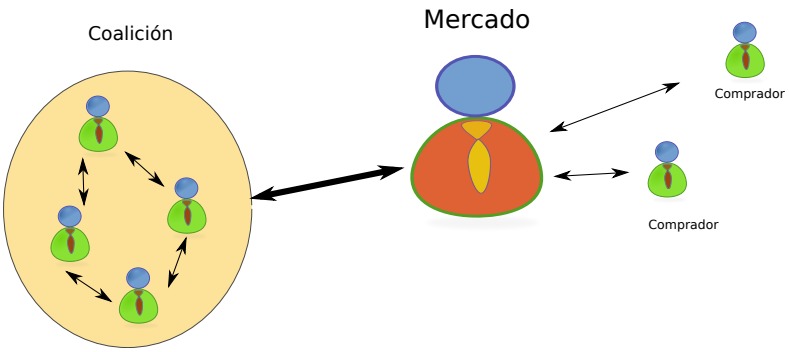
- La inter-acción se ve como organización social:
  - De relaciones entre pares a relaciones jerárquicas
  - De relaciones puntuales a relaciones a largo plazo
- Esto permite:
  - Caracterizarlas y describirlas de manera abstracta
  - Agrupar diferentes componentes y usarlos como una unidad facilitando la descomposición
  - Usar en el análisis y desarrollo métodos/protocolos bien conocidos en organizaciones sociales



- No es posible determinar todas las inter-acciones en el diseño
- Los agentes son capaces de decidir como reaccionar y resolver las interacciones en ejecución (flexibilidad)
  - Tomando decisiones sobre situaciones imprevistas
  - Tomando decisiones sobre interacciones erróneas
  - Pidiendo asistencia a otros agentes
- Su comportamiento no esta predeterminado totalmente, resulta de la interacción dinámica entre los participantes

Ejemplo

- En un sistema de comercio electrónico:
  - Agentes compradores determinan que pueden obtener un mejor precio si se coaligan y hacen compras más grandes en lugar de compras individuales



- Comunicación:
  - Software: Comunicación a nivel de señal
  - Agentes: Comunicación a nivel simbólico
- Los agentes
  - persiguen objetivos y necesitan que otros agentes persigan objetivos relacionados (delegación)
  - necesitan llegar a acuerdos
  - necesitan tomar decisiones organizacionales
  - necesitan de comunicarse sus creencias/conocimiento

- Para que un mensaje sea entendido, se deben asignar un significado a los elementos de su contenido
- Requeriremos una **Ontología** para hacer la correspondencia entre una codificación y un significado
- **Paso de mensajes dinámico**
  - Los agentes han de descubrir si comparten un conocimiento mutuo del dominio (la ontología) para poder continuar la comunicación

- Los sistemas software necesitan poseer un conocimiento mutuo completo para poder interaccionar (todo está predeterminado)
- Los agentes software pueden tener (o no) conocimiento mutuo completo acerca de:
  - Los objetivos de otros agentes
  - Sus estrategias (acciones que pueden usar)
  - Sus utilidades (beneficio de sus acciones)

- Interacciones que asumen el completo conocimiento se clasifican como cooperación y coordinación
  - Todos los agentes buscan un fin común
  - Los agentes necesitan delegar objetivos para cumplir su cometido
- En muchos casos la suposición de conocimiento completo puede no ser cierta: competición
  - Solo algunos agentes pueden cumplir su cometido (p.ej.: recursos limitados)

- Los puntos de decisión de los sistemas software son deterministas
- Los agentes software (inteligentes) están dotados de mecanismos de razonamiento
- La **toma de decisiones** involucra múltiples flujos de control:
  - Pensamos también en consecuencias inmediatas
  - Reaccionamos a estímulos
  - Planteamos objetivos a corto plazo
  - Replanteamos objetivos a largo plazo

- **Agentes**, entidades autónomas, elementos de control independientes, situación en un entorno, interacción
- **Entorno**, mundo de entidades y recursos que el agente percibe, controla, explota o consume.
- **Roles** e **interacciones**, funcionalidades, actividades, responsabilidades y patrones de interacción.
- **Reglas de organización**, restricciones a roles e interacciones, o relaciones entre roles y entre protocolos
- **Estructuras** y **patrones de organización**, topología de interacción, régimen de control de actividades

Inteligencia Artificial (fuerte)

Un sistema multiagente es una sociedad de individuos (agentes software inteligentes) que interaccionan intercambiando conocimiento y negociando entre ellos para lograr sus propios intereses o un objetivo global.

Ingeniería de Software (débil)

Un sistema multiagente es un sistema software compuesto por múltiples elementos de control independientes y encapsulados (agentes) interaccionando entre ellos en el contexto de una aplicación específica

- Se focaliza en las características de los agentes que tienen impacto en el desarrollo de software:
  - Concurrencia, interacción, múltiples elementos de control
  - La *inteligencia* puede verse como una forma particular de control independiente, las *conversaciones* como una forma particular de interacción.
- Es más general:
  - Diferentes sistemas software, incluso si no se conciben como basados en agentes, pueden caracterizarse en términos de sistemas multiagente débiles

- Una metodología de software tiene como objetivo introducir una disciplina en el desarrollo:
  - qué producir y cuando
- Define el marco conceptual del desarrollo
- Define las abstracciones a usar para modelar el software:
  - Orientada a datos, flujos, objetos...

- **Análisis**, qué debería hacer el sistema y cuáles son las restricciones de desarrollo
- **Diseño**, especificación del sistema cumpliendo los objetivos y restricciones
- **Desarrollo**, proceso de producción del sistema software
- **Validación**, comprobar que el software es lo que el cliente quiere
- **Evolución**, cambiar el software en respuesta a los cambios

¿Cuál es el proceso del software ideal?

No existe un proceso ideal

- AOSE es el paso siguiente en la evolución de orientación a objetos, patrones de diseño y diseño basado en componentes
- Apropiado para sistemas abiertos y sistemas distribuidos
- Sus características están más alineadas con este tipo de entornos (p. ej.: Internet, Cloud computing, IoT...)

- La fase de **análisis** consiste en entender:
  - Cuáles son los actores principales que interaccionan con el sistema
  - Cómo el sistema interacciona con esos actores
  - Qué tiene que hacer el sistema (globalmente)
- En la fase de análisis vemos el sistema como una entidad cerrada para no anticipar decisiones de diseño

- Asociamos agentes con las entidades de los escenarios que se analizan
- Dentro de esos escenarios asignamos:
  - Roles, responsabilidades y capacidades
  - Patrones de interacción entre agentes
- El objetivo es tener una visión neutra del problema
- Algunas metodologías no usan la palabra agente para denotar las entidades en esta fase

Analogía cinematográfica

- Agentes software = Actores representando papeles
- Casos de uso = Guión
- Ingeniero de software = Productor/director

- Cuáles son los principales componentes que interaccionan dentro del sistema
- Cuáles son las responsabilidades y capacidades de cada componente del sistema
- Cómo los componentes interaccionan para implementar el sistema (su arquitectura)

- Se asocia agentes con los componentes que se usan para construir el sistema
- A partir de ahí se refinan:
  - Roles, responsabilidades y capacidades
  - Patrones de interacción entre agentes
- A diferencia del análisis, hay que escoger qué agentes usar y como interaccionan

- **GAIA**: desarrollo como el diseño de una organización
- **TROPOS**: enfocada en el análisis de requerimientos
- **PASSI**: metodología paso a paso de requerimientos a código que integra modelos de diseño y conceptos de orientación a objetos e inteligencia artificial
- **Prometheus**: se focaliza en el diseño organizacional y en el de la arquitectura interna del agente (diseño de sistemas de agentes BDI)

5

Metodología Prometheus

Diseño de sistemas multiagente

Prometheus

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Metodología Prometheus - Fases
- Especificación
  - Actores, Objetivos, Escenarios, Roles, Acciones, Percepciones
- Diseño Arquitectónico
  - Agentes, Fuentes de datos, Protocolos
- Diseño Detallado
  - Capacidades, Planes, Mensajes

- **Prometheus** es una metodología iterativa que cubre el proceso completo de ingeniería de software
  - Análisis
  - Diseño
  - Diseño detallado
  - Implementación

- Su objetivo es el desarrollo de agentes inteligentes, orientado en particular a agentes BDI
  - Usa como abstracciones objetivos, creencias, planes y eventos
- La especificación resultante puede ser desarrollada en cualquier implementación de agentes que cubra esas abstracciones
  - En particular está pensada para la implementación con el lenguaje de agentes JACK

- Es una metodología que ha evolucionado a partir de la experiencia práctica en el desarrollo de sistemas multiagente
- Está enfocada al desarrollo de software en la industria
- Ha recibido la experiencia de uso en proyectos tanto a pequeña escala (estudiantes) como en la industria (AOS Group `http://aosgrp.com/`)

- Está enfocada en dar una guía y estructuración detallada para facilitar el desarrollo de herramientas de diseño
- Es una mezcla de:
  - Una notación gráfica que facilita una vista general del sistema
  - Una notación textual estructurada para una vista detallada
- Es jerárquica y modular
- Existe un prototipo de herramienta para el diseño (PDTool)

- Estamos diseñando sistemas distribuidos (no monolíticos)
- El sistema interactúa con múltiples entidades externas (también distribuidas) al mismo tiempo
- El sistema puede tener múltiples réplicas de cada entidad que lo compone
- Las entidades internas y externas pueden aparecer y desaparecer dinámicamente
- Las acciones que realiza el sistema no han de ser síncronas
- Puede haber múltiples flujos de ejecución a la vez

(1) Especificación del sistema:

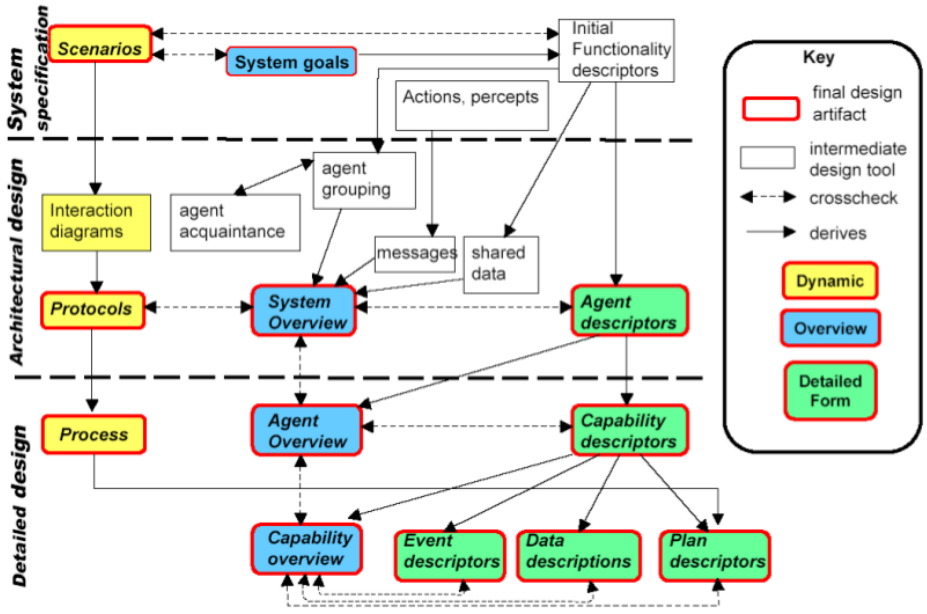
- Focalizada en la identificación de las funciones básicas del sistema:
  - Roles (funcionalidades)
  - Objetivos
  - Escenarios
  - Entradas (*percepts*), salidas (*actions*) y su procesamiento

(2) Diseño arquitectónico:

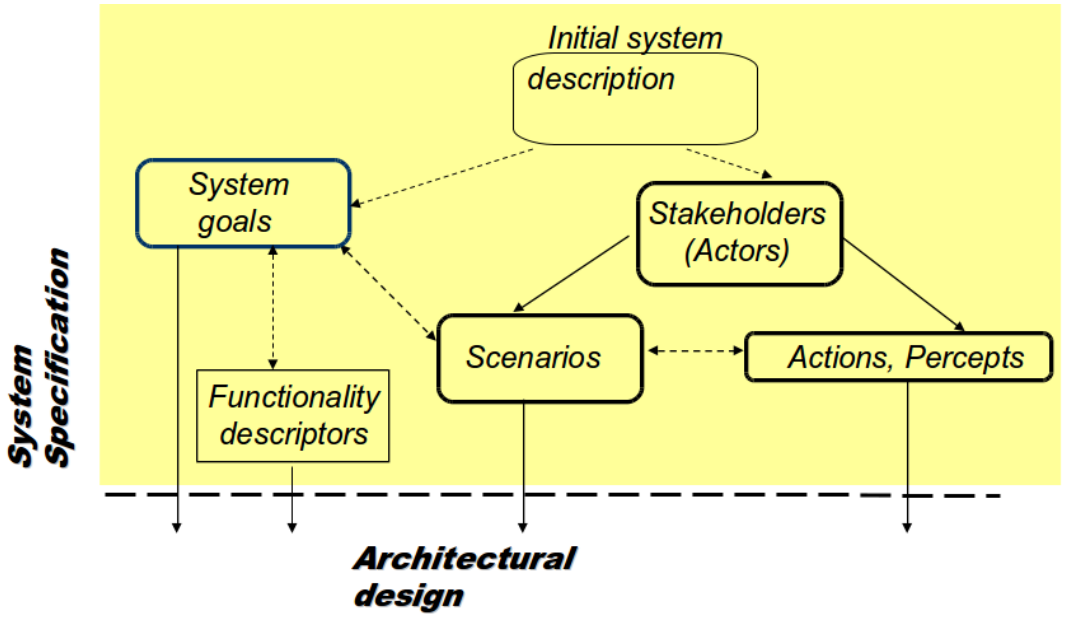
- Determina qué agentes contendrá el sistema y cómo interaccionarán

(3) Diseño detallado:

- Describe los elementos internos de los agentes
- La forma en la que realizarán sus tareas dentro del sistema
- Enfoque en describir capacidades (*capabilities*, módulos del agente), eventos internos, planes y estructuras de datos internas



- Sistema definido por:
  - Objetivos: Diagrama de objetivos
  - Escenarios: Escenarios de casos de uso
  - Roles: Descriptores de funcionalidades
- La interacción con el entorno se describe en términos de:
  - Acciones
  - Percepciones
  - Datos externos



- Comenzar con una descripción a alto nivel del sistema (textual)
- Identificar los actores
- Identificar los objetivos del sistema (y subobjetivos)
- Identificar los roles (funcionalidades) que cubren los objetivos
- Identificar/definir escenarios
- Identificar entradas y salidas (percepciones y acciones)

¡Atención!

¡No es un proceso secuencial!

Cada elemento realimenta/se interrelaciona con los otros

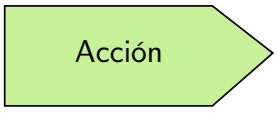
- ¿Para que se construye el sistema? Objetivos principales
- ¿Cuáles son los subobjetivos que permiten conseguirlos?
- Descritos en el diagrama de objetivos
- **¿Cómo?** (subobjetivos)
- **¿Porqué?** (objetivos padre)

Fase de especificación - Percepciones

- Los agentes están en un entorno y recibirán información de él (**percepciones**)
- Habrá cosas que sucederán en el entorno que serán significativas para los agentes (**eventos**)
- Estos pueden ser directamente percepciones u obtenerse de percepciones después de un procesamiento

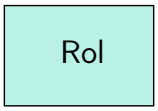
Fase de especificación - Acciones

- Las **acciones** se hacen para modificar el entorno
- Pueden ser simples y directas o interacciones complejas
- Preguntas:
  - ¿Tienen una duración?
  - ¿Pueden fallar?
  - ¿Sabremos si lo hacen?
  - ¿Si fallan tendrán un efecto en el entorno?



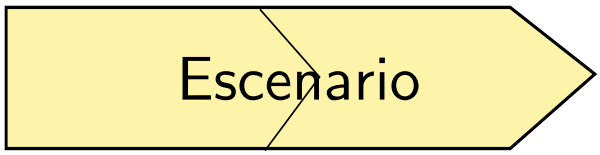
Fase de especificación - Roles

- Los **roles** describen funcionalidades que el diseñador piensa que debe tener el sistema
- Deben ser específicas y poder definirse en una o dos frases
- Los roles estarán ligados a los objetivos
  - Puede haber más de un objetivo por rol
  - Un rol puede aparecer en más de un objetivo
- Hay unos descriptores que deben usarse para detallarlos



- Los **escenarios** muestran una instancia particular de la ejecución del sistema (pueden tener variaciones dependiendo de condiciones)
- Un escenario siempre es iniciado por una percepción
- Un escenario consiste en un colección de pasos (acciones, percepciones, objetivos, escenarios)
  - Los pasos que los componen están ordenados secuencialmente, pero no tiene porque corresponder a su orden cuando se ejecuten
- Cada paso está ligado a un rol y a los datos usados y producidos

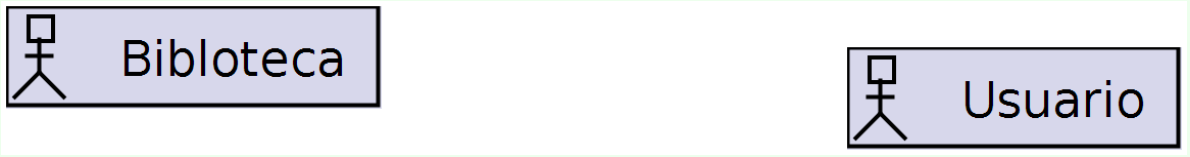
- Cada objetivo del sistema debe aparecer en al menos un escenario
- Los escenarios habitualmente involucran más de un rol (funcionalidad)



- Queremos un sistema que sea capaz de manejar los requerimientos de una biblioteca, básicamente:
  - Permitir sacar libros, informando al usuario de la fecha de retorno
  - Permitir retornar libros
  - Permitir la reserva de libros aún no disponibles
  - Permitir la notificación de libros prestados que están fuera de plazo
  - Permitir la notificación de llegada de libros reservados

Ejemplo: Especificación - Actores

- Podemos identificar dos actores
  - Biblioteca (sistema)
  - Usuario (actor externo)
- Estos no tienen por qué ser agentes en el diseño final
- También pueden ser actores que no modelemos, pero que queramos tener para representar, por ejemplo, los elementos del entorno que generan las percepciones o reciben las acciones



Ejemplo: Especificación - Objetivos, primer nivel

- Podríamos identificar como objetivos de alto nivel del sistema:
  - Prestar libros (notificando fecha retorno)
  - Retornar libros
  - Reservar libros no disponibles
  - Notificar libros fuera de plazo
  - Notificar la llegada de libros reservados

Ejemplo: Especificación - Subobjetivos

Para cada uno de los objetivos nos preguntamos **cómo** son obtenidos

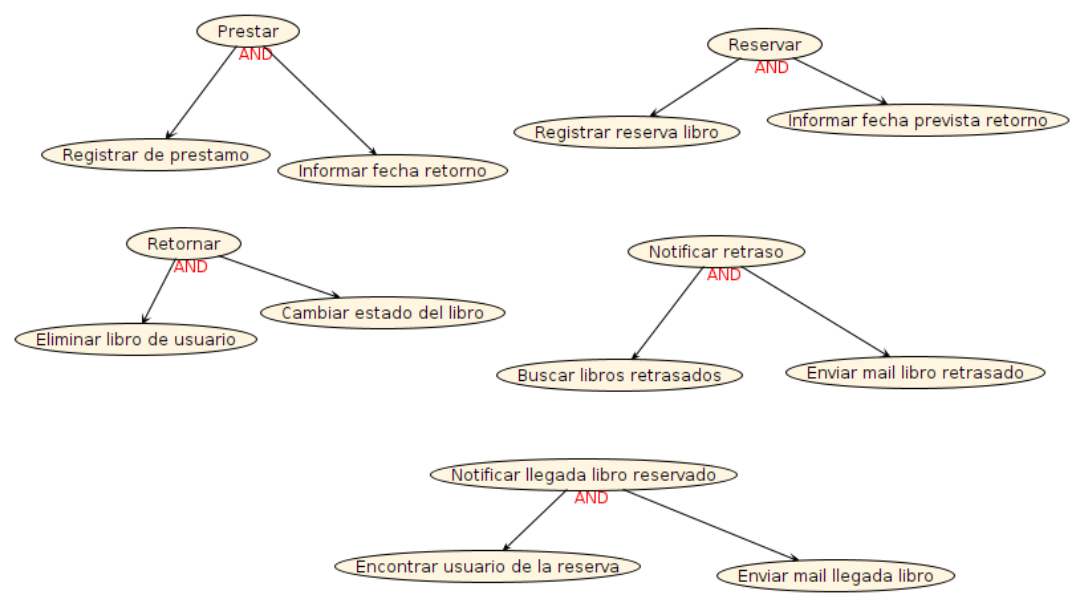
- Prestar libro $\Longrightarrow$
  - Registrar ID del libro a ID del usuario
  - Informar de la fecha de retorno
- Retornar libro $\Longrightarrow$
  - Borrar ID de libro del ID de usuario
  - Registrar ID del libro como disponible
- Reservar libros no disponibles $\Longrightarrow$
  - Registrar ID del libro como reservado por ID del usuario
  - Informar de la fecha de retorno del libro al usuario

Ejemplo: Especificación - Subobjetivos

Para cada uno de los objetivos nos preguntamos **cómo** son obtenidos

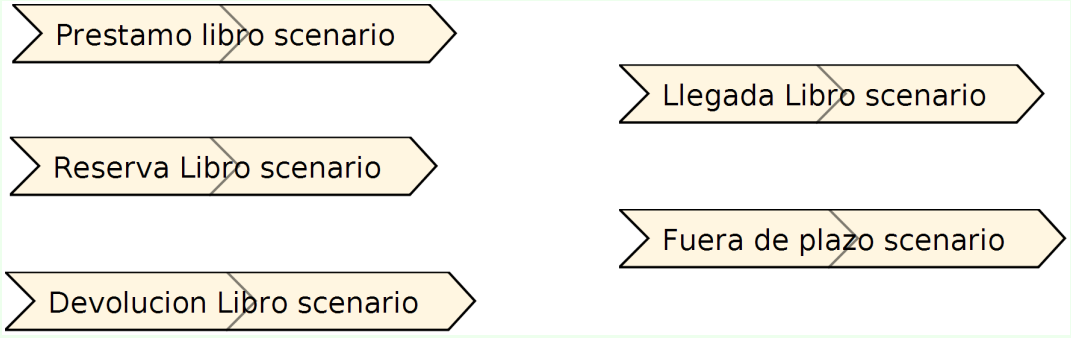
- Notificar libro fuera de plazo $\Longrightarrow$
  - Acceder al registro de libros al inicio del día
  - Enviar correo para los libros no entregados a tiempo
- Notificar la llegada de libros reservados $\Longrightarrow$
  - Acceder a la lista de libros reservados por los usuarios
  - Enviar correo informando de la llegada del libro

Ejemplo: Especificación - Diagrama de objetivos



Ejemplo: Especificación - Escenarios

1. Cuando el usuario viene a pedir un libro (Préstamo Libro)
2. Cuando el usuario retorna un libro (Devolución Libro)
3. Cuando un libro está fuera de plazo (Fuera de plazo)
4. Cuando el usuario pide reservar un libro (Reserva Libro)
5. Cuando el libro reservado llega (Llegada libro)



- Subescenarios/variantes: Descomposición del escenario o variantes
- Objetivos: Cuáles son los objetivos que se persiguen en este escenario
- Percepciones: Qué eventos exteriores ponen en marcha el escenario
- Acciones: Cuáles son las acciones que se desarrollan dentro del escenario que se realizan dentro del escenario
- Roles: Cuáles son los roles que participan en el escenario

Podemos esbozar los pasos que componen cada escenario

- **Préstamo Libro**
  1. (P) Petición de libro
  2. (O) Registrar el código de libro como prestado
  3. (O) Informar de la fecha de retorno
  4. (A) Dar fecha de retorno al usuario
  5. (A) Entregar el libro
- **Devolución Libro**
  1. (P) Devolver libro
  2. (O) Borrar el código del libro de los prestados
  3. ...
- ...

1. Procesador de préstamos de libros
2. Procesador de libros retornados
3. Procesador de libros fuera de plazo
4. Procesador de reserva de libros
5. Procesador de llegada de libros reservados

Los roles pueden estar asignados al mismo actor/agente o varios pueden compartir roles

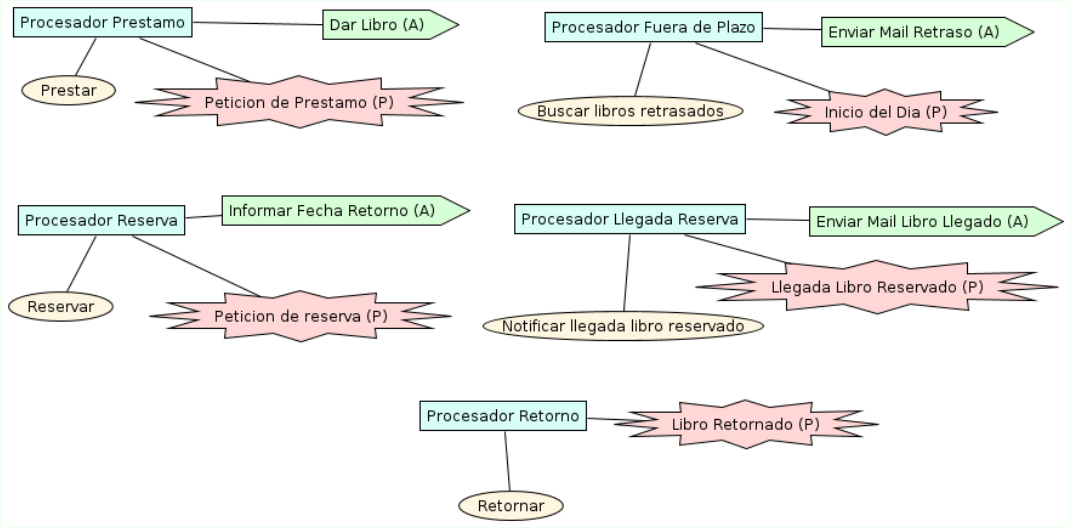
Hemos de identificar las entradas y salidas del sistema en forma de percepciones y acciones y asignarlas a los roles

- Préstamo de libros:
  - **Percepción**: Petición de libro
  - **Acción**: Dar libro
- Retorno de libro:
  - **Percepción**: Libro retornado
- Libros fuera de plazo
  - **Percepción**: Inicio del día
  - **Acción**: Enviar correo de libro retrasado

Hemos de identificar las entradas y salidas del sistema en forma de percepciones y acciones y asignarlas a los roles

- Reserva de libro
  - **Percepción**: Petición de reserva
  - **Acción**: Informar de fecha de retorno del libro
- Llegada de libros reservados
  - **Percepción**: Llegada de libro reservado
  - **Acción**: Enviar correo de llegada de libro

- Hemos de ligar los objetivos con los escenarios y los roles
- Hemos de ligar las percepciones y acciones a roles

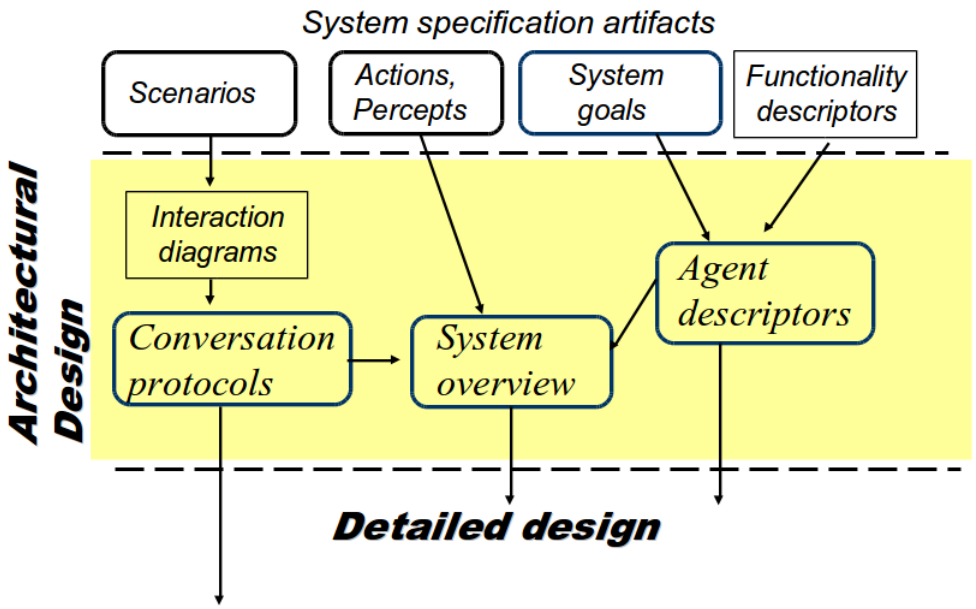


Rol	
Nombre	Procesador Fuera de Plazo
Descripción	Se encarga de informar a los usuarios que no han devuelto un libro a tiempo
Evento Iniciador	Inicio del día
Acciones	Enviar un mail avisando del retraso
Información usada	Fecha de retorno de los libros prestados
Información Producida	ninguna
Objetivos	Notificar retraso

Escenario			
Nombre	Fuera de plazo		
Descripción	Hay libros prestados que no han sido retornados a tiempo		
Evento	Inicio del día		
Pasos			
Tipo	**Nombre**	**Rol**	**Datos**
Percepción	Inicio Día	P. Libro fuera Plazo	
Objetivo	Notificar Retraso	P. Libro fuera Plazo	
Objetivo	Buscar Libros Retrasados	P. Libro fuera Plazos	Libros Prestado
Objetivo	Enviar Mail Libro Retrasado	P. Libro fuera Plazos	
Acción	Enviar e-mail	P. Libro fuera Plazo	e-mail usuario

Percepción	
Nombre	Inicio del día
Descripción	Evento que indica el inicio de un nuevo día
Información	Ninguna
Conocimiento Actualizado	Hay un nuevo día
Fuente	Entorno
Procesamiento	Ninguno
Agentes que responden	Por determinar
Frecuencia	Una vez al día

Acción	
Nombre	Enviar mail retraso
Descripción	Enviar un mail a un usuario que no ha retornado un libro
Parámetros	E-mail usuario, código del libro
Duración	Instantáneo
Fallo	Puede no llegar al usuario
Efectos Laterales	Ninguno



- ¿Cuando un componente del sistema es un agente?
  - ¿Es autónomo?
  - ¿Tiene objetivos?
  - ¿Es activo? (tiene procesos internos en ejecución de manera concurrente)
  - ¿Hace varias cosas a la vez? ¿Debe razonar respecto a la interacción entre ellas?
  - ¿Debe cambiar la manera de hacer las cosas basándose en cambios en el entorno?
- Si la respuesta es mayoritariamente sí, deberíamos modelarlo como un agente

- Agrupar las funcionalidades en tipos de agentes basándose en:
  - Cohesión (forman parte del mismo contexto)
  - Acoplamiento (tienen dependencias funcionales fuertes)
- Agrupar funcionalidades que:
  - Están relacionadas según el sentido común
  - Requieren la misma información

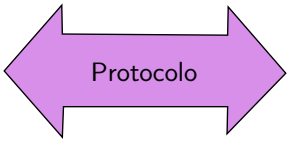
- No agrupar funcionalidades que
  - No están claramente relacionadas
  - Han de ejecutarse en plataformas separadas
  - No pueden compartir información por seguridad/privacidad
  - Han de ser modificadas por diferentes desarrolladores

- ¿Cuántos agentes de cada tipo debe haber (uno, muchos, dinámico)?
- ¿Cuál es el tiempo de vida del agente (permanente/ocasional)?
- ¿Cuál es el estado inicial del agente?
- ¿Qué pasa cuando un agente desaparece del sistema?
- ¿Cuáles son los datos usados/producidos por el agente?
- ¿A qué eventos reaccionan los agentes?
- ¿Cuáles son sus interacciones? ¿Cómo se realizan?

- Usar fuentes de datos centralizadas destruye el propósito de hacer un sistema distribuido (introduce acoplamientos) hay que buscar la máxima localidad
- Los roles/funcionalidades ven los datos desde diferentes perspectivas, eso permite partirlos
- Diferentes roles/funcionalidades tienen diferentes derechos de acceso a los datos
- No es un problema que ciertos datos estén replicados si eso ayuda a la distribución del sistema
- En la práctica el mantener los datos de un sistema distribuido es complejo y necesita sopesar ventajas y desventajas de las alternativas
- **CAP theorem** (Consistency/Availability/Partition Tolerance)

- El diagrama de **agrupación de roles a agentes**
- El diagrama de **relación entre agentes**
- El diagrama de **acoplamiento entre datos y roles**

- Identificamos y diseñamos la interacción entre los agentes
- Usamos como base los pasos de los escenarios:
  - Si un paso involucra un rol asignado a un agente y el siguiente involucra otro rol asignado a otro agente, tendrá que haber un intercambio de mensajes
- Definir los protocolos de interacción



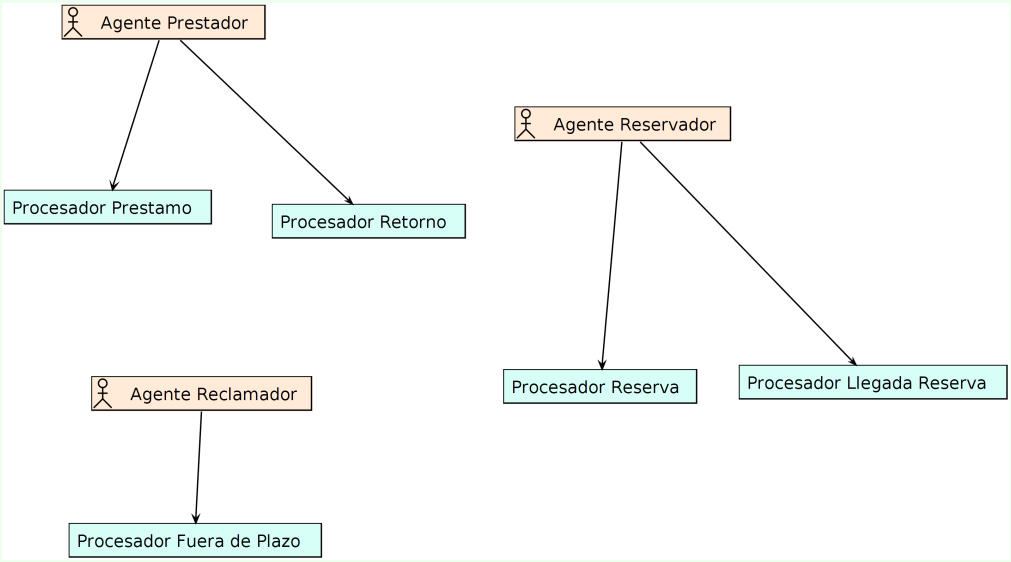
- Identificamos como creencias (datos) que han de manejar los agentes del sistema la información de todos los libros prestados y reservados
- El rol del procesador de préstamo y procesador de retorno modifican la creencia del préstamo de un libro
- El rol de procesador de fuera de plazo utiliza esa creencia para realizar su acción

- El rol de procesador de reserva modifica la creencia de la reserva de un libro
- El rol de procesador de llegada de reserva usa esta creencia para realizar su acción

Ejemplo: Diseño Arquitectónico - Diagrama de roles/agentes

- Identificamos tres agentes diferentes:
  - **Prestador**
  - **Reclamador**
  - **Reservador**
- Agrupamos los roles según su coherencia semántica

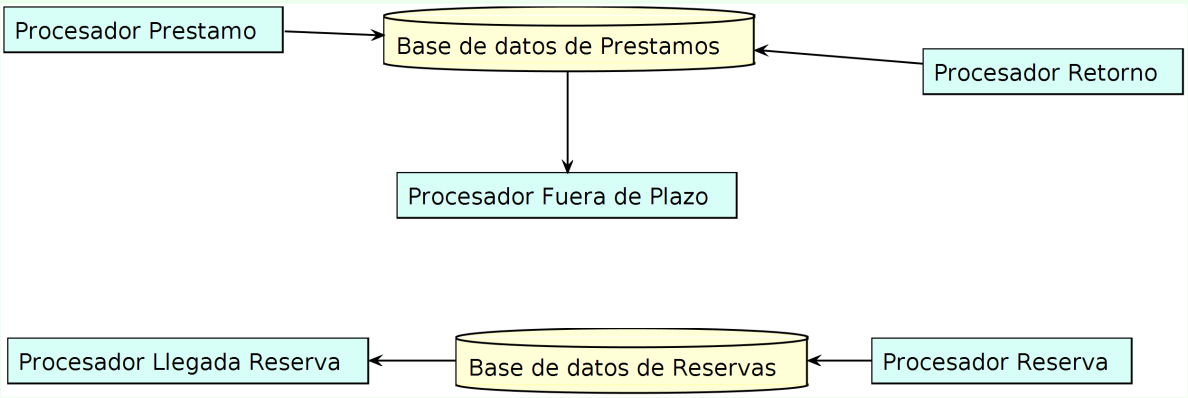
Ejemplo: Diseño Arquitectónico - Diagrama de roles/agentes



Ejemplo: Diseño Arquitectónico - Diag. de acoplamiento de datos

- Identificamos dos fuentes de datos (préstamos y reservas)
- Asignamos los roles a las fuentes de datos indicando en que dirección (entrada/salida) va la relación
- No tienen por qué corresponder a fuentes de datos físicamente separadas
- Tampoco una fuente de datos tiene por qué corresponder a una fuente de común entre todos los roles

Ejemplo: Diseño Arquitectónico - Diag. de acoplamiento de datos

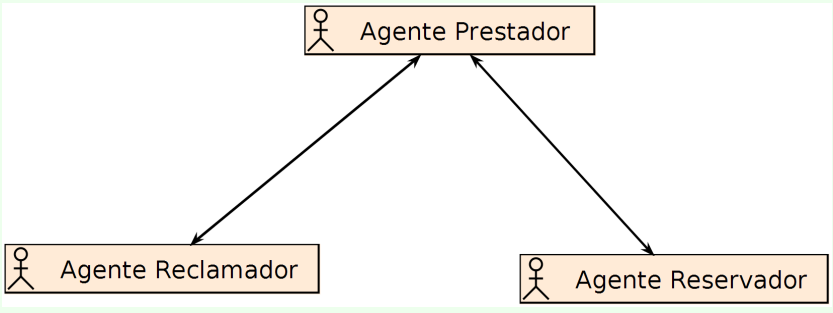


Diseño Arquitectónico - Descriptores - Datos

Datos	
Nombre	Base de datos de préstamos
Descripción	Contiene los registros de los préstamos de libros a usuarios
Tipo de datos	Registro préstamo de libro a usuario
Campos	Id usuario, Id Libro, Fecha retorno
Persistente	Sí
Externa al sistema	No
Incialización	Vacía
Producida por	Agente prestador, plan registrar préstamo
Usada por	Agente Prestador, Agente Reclamador...
Usada cuando	El usuario pide un libro...

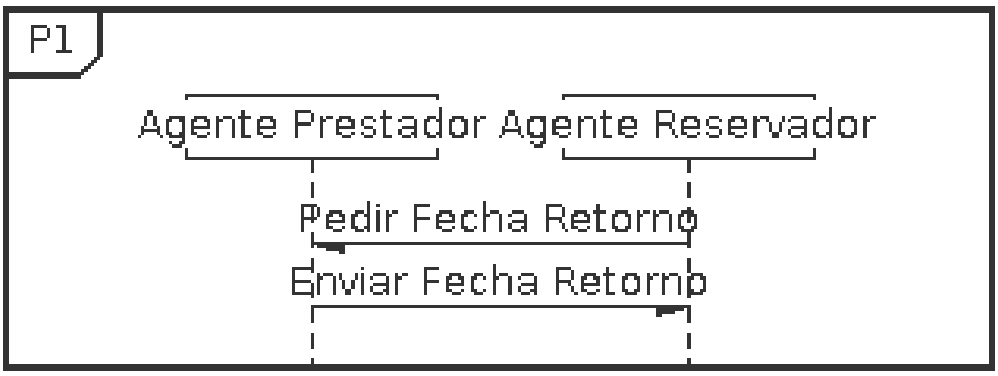
Ejemplo: Diseño Arquitectónico - Diag. de relación entre agentes

- Decidimos que los agentes se vayan comunicando
  - Reservador pide a Prestador las fechas de devolución de los libros prestados cuando se piden
  - Prestador informa de la fecha de devolución de los libros prestados a Reclamador
- Esto relaciona a los agentes prestador con el reservador y el prestador con el reclamador



Ejemplo: Diseño Arquitectónico - Mensajes y Protocolos

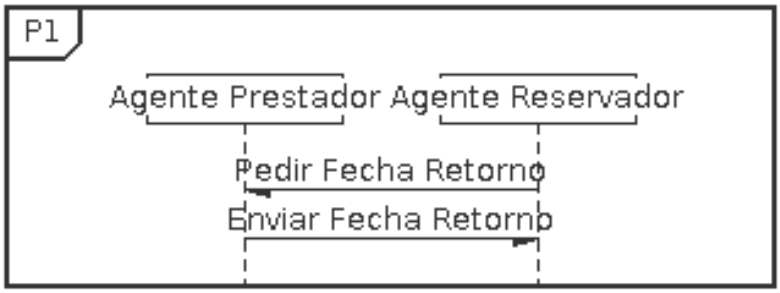
- La comunicación entre los agentes nos obligará a definir:
  - Mensajes: ¿Cuál es su contenido y su significado?
  - Protocolos: ¿Quién interviene en el protocolo? ¿Cómo se realiza el intercambio de mensajes?



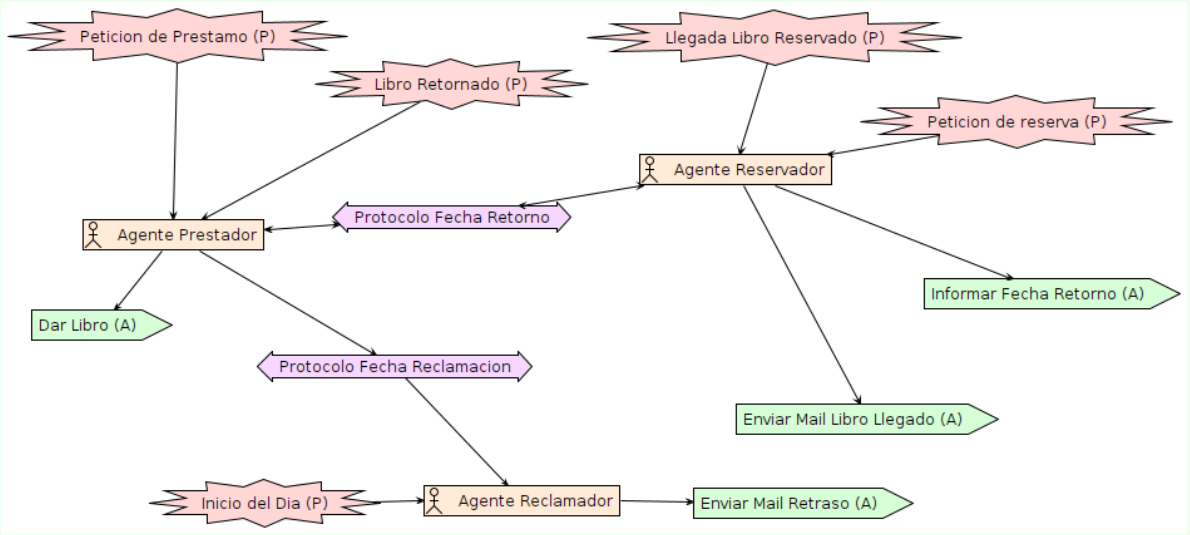
Diseño Arquitectónico - Descriptores - Agente

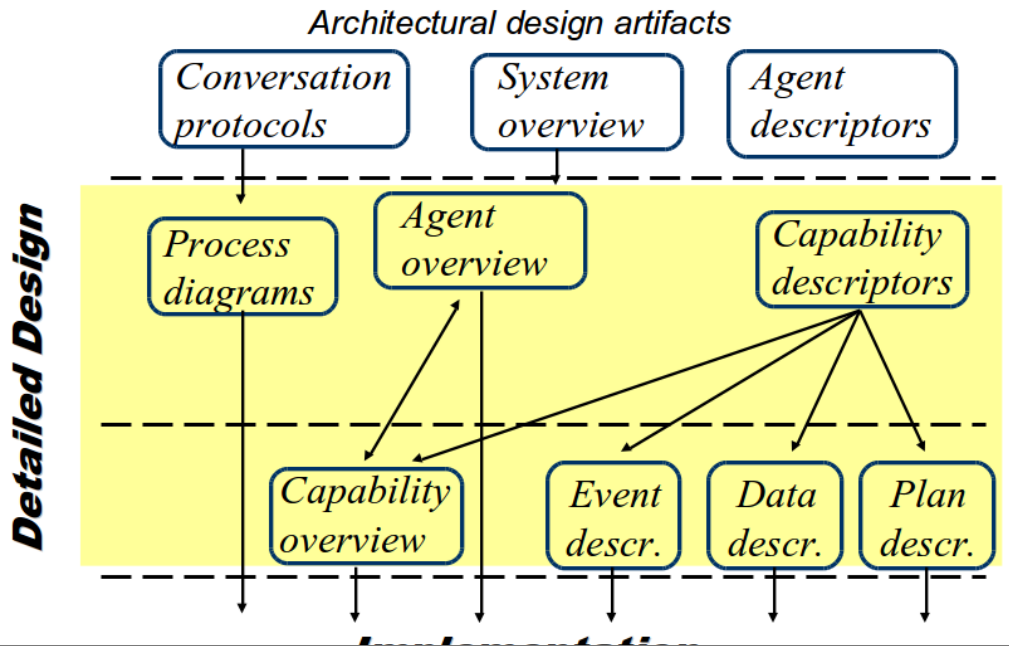
Agente			
Nombre	Agente Reclamador		
Descripción	Se encarga de procesar los libros retrasados		
Cardinalidad mínima	1	**Cardinalidad máxima**	1
Duración	ilimitada	**Inicialización**	Ninguna
En caso de fallo	Nada	**Percepciones**	Inicio de día
Acciones	Enviar mail retraso a usuario	**Usa datos**	Libros prestados a usuarios
Produce datos	Ninguno	**Datos internos**	Por definir
Objetivos	Notificar retraso...	**Roles**	Procesador Fuera plazo
Protocolos	Protocolo Fecha		

Diseño Arquitectónico - Descriptores - Protocolos

Protocolo	
Nombre	Pedir Fecha
Descripción	Obtener la fecha de retorno de un libro
Mensajes	Pedir Fecha Retorno, Enviar Fecha Retorno
Escenarios	Fuera de plazo, Reserva libro
Agentes	Prestador, Reservador, Reclamador
Notas	

Mensaje	
Nombre	Pedir Fecha Retorno
Descripción	Pedir la fecha de retorno de un libro
Distribución	De Reservador a Prestador, De Reclamador a Prestador
Propósito	Obtener la fecha de retorno de un libro
Información contenida	Identificador del usuario y libro





Diseño Detallado

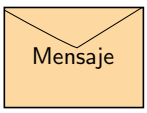
- En esta fase se desarrollan los **elementos internos** de los agentes
  - Se definen en función de capacidades, datos, eventos y planes
  - Se usan diagramas de proceso como un paso intermedio entre protocolos de interacción y planes

Diseño Detallado - Pasos

- Desarrollar la estructura interna de cada agente individual
- Identificar las capacidades de cada agente empezando con sus funcionalidades
- Generar los descriptores de capacidades
- Generar los diagramas de visión general del agente
- Generar los descriptores de los planes
- Generar los descriptores de los eventos
- Generar los descriptores de los mensajes externos e internos
- Generar los descriptores de los datos

- Las **capacidades** se definen como módulos (un trozo de código que hace una cosa en particular)
- Se parte de los roles (funcionalidades) de la primera fase, dividiéndolos hasta obtener capacidades primitivas
- Las funcionalidades de bajo nivel que son comunes entre varios agentes pueden usarse también como capacidades
- Hay que tener presente la reusabilidad al determinarlas

- Las capacidades se descomponen a su vez en **planes**, que son los que llevan a cabo las tareas
- Estos estarán:
  - Conectados a las **percepciones**/eventos
  - Recibirán los **mensajes** generados por los protocolos de interacción entre los agentes
  - Generarán **mensajes** internos para comunicarse con otros planes dentro de la capacidad
  - Realizarán las acciones



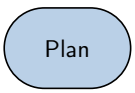
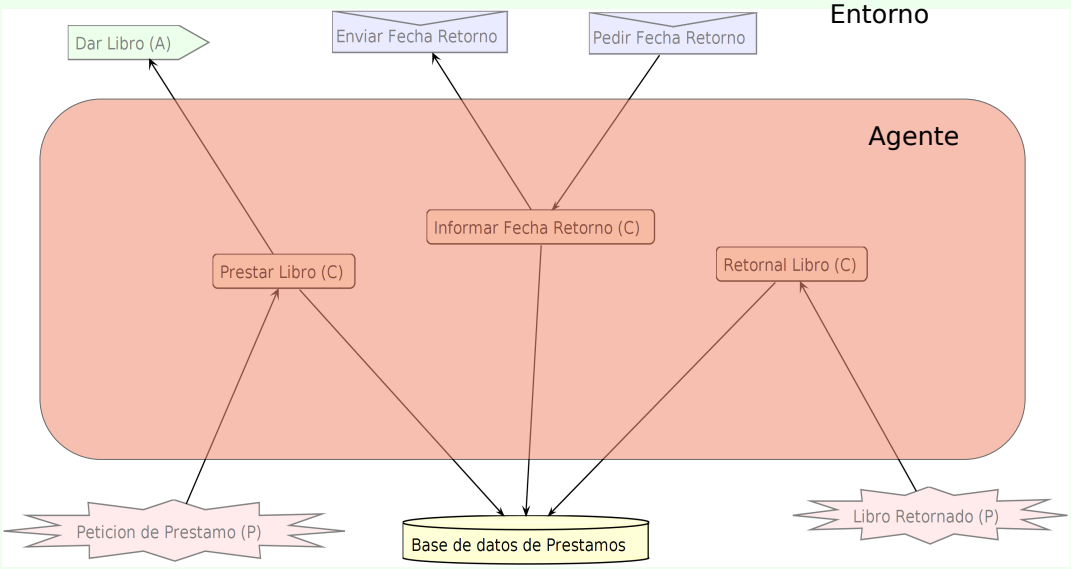
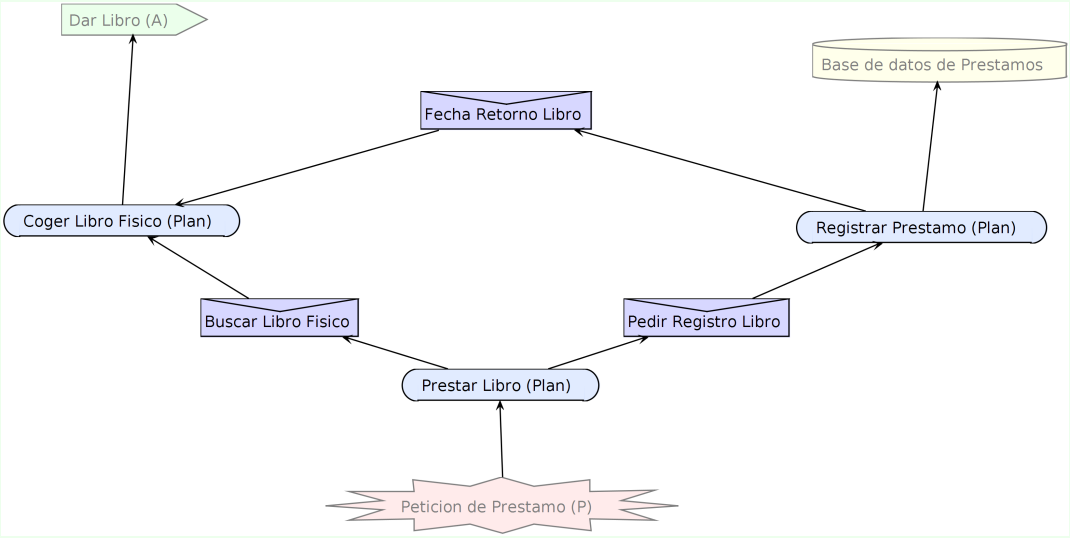


Diagrama del agente **Prestador**



Ejemplo: Diseño Detallado - Diagrama de plan

Planes de la capacidad **Prestar Libro**



Ejemplo: Diseño Detallado - Descriptores - Capacidades

Capacidad	
Nombre	Prestar Libro (C)
Descripción	Obtiene un libro físico, determina la fecha de retorno...
Objetivos	Prestamo...
Protocolos	Ninguno
Mensajes Entrantes	Ninguno
Mensajes Salientes	Ninguno
Mensajes Internos	Pedir registro libro, fecha retorno libro...
Percepciones	Petición de libro (P)
Acciones	Dar Libro (A)
Datos usados	Base de datos usuario
Datos producidos	Fecha de retorno libro, registro de préstamo
Datos Internos	Por definir
Planes incluidos	Prestar Libro, Registrar préstamo...
Capacidades incluidas	Ninguna

Ejemplo: Diseño Detallado - Descriptores - Planes

Plan	
Nombre	Prestar Libro (P)
Descripción	Desarrolla las acciones necesarias para prestar un libro
Iniciador	Percepción de petición de préstamo
Mensajes Entrantes	Ninguno
Mensajes Salientes	Pedir registro libro, Coger libro físico
Datos usados	Base de datos usuario
Datos producidos	Ninguno
Fallo	Falta de acceso a los datos
Recuperación de fallo	Enviar mensaje a mantenimiento
Procedimiento	Código del plan

6

Prometheus: PACMAN

Diseño de sistemas multiagente

Prometheus - PACMAN

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Especificación del sistema

Diseño Arquitectónico

Diseño Detallado

- PACMAN tiene como objetivos comer todas las píldoras que hay en el laberinto sin ser capturado más de n veces.
- Los fantasmas tienen como objetivo capturar a PACMAN más de n veces.
- Hay píldoras especiales que hacen a los fantasmas vulnerables y que puedan ser capturados durante $s1$ segundos, PACMAN puede decidir capturar a los fantasmas tras comer una de estas píldoras
- Un fantasma capturado puede volver al juego en modo normal tras $s2$ segundos.
- Un fantasma en modo vulnerable evita ser capturado

- Elementos adicionales:
  - PACMAN y los fantasmas solo pueden ver hasta el muro más próximo en todas direcciones.
  - PACMAN puede elegir entre comer una píldora o no cuando pasa sobre ella, asumiendo que el detenerse a comerla le toma un tiempo y, por lo tanto, le retrasa.
  - Los fantasmas si ven a PACMAN le persiguen activamente.
  - Los fantasmas pueden comunicarse entre ellos su posición y la posición de PACMAN si lo ven.

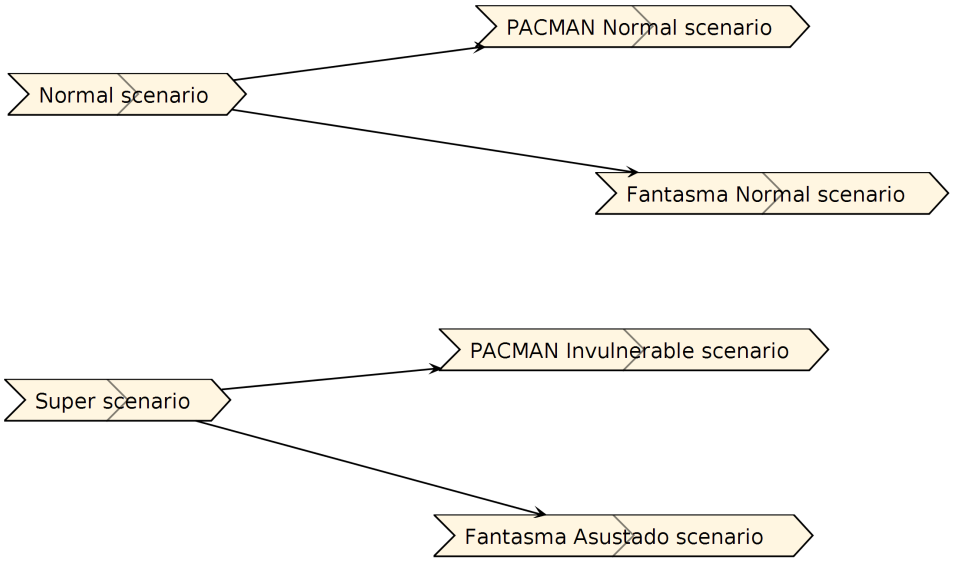
- Otras consideraciones
  - Modelaremos únicamente los agentes que controlan los personajes
  - Asumiremos que el sistema que controla todo el juego es el entorno externo
  - El entorno externo se encargará de enviar las percepciones que reciben los agentes y recibir las acciones

- Podemos distinguir tres actores:
  - El sistema que controla el entorno (que no modelaremos)
  - El PACMAN
  - Los Fantasmas

Especificación- Escenarios

- Podemos distinguir dos escenarios en los que participan los actores
  - Escenario normal (el PACMAN come pildoras y huye de los fantasma, los fantasmas persiguen a PACMAN)
  - Escenario *súper* (el PACMAN también se preocupa de perseguir a los fantasmas y los fantasmas huyen de PACMAN)
- Podemos subdividir cada escenario en la visión que tienen los actores de cada uno de ellos

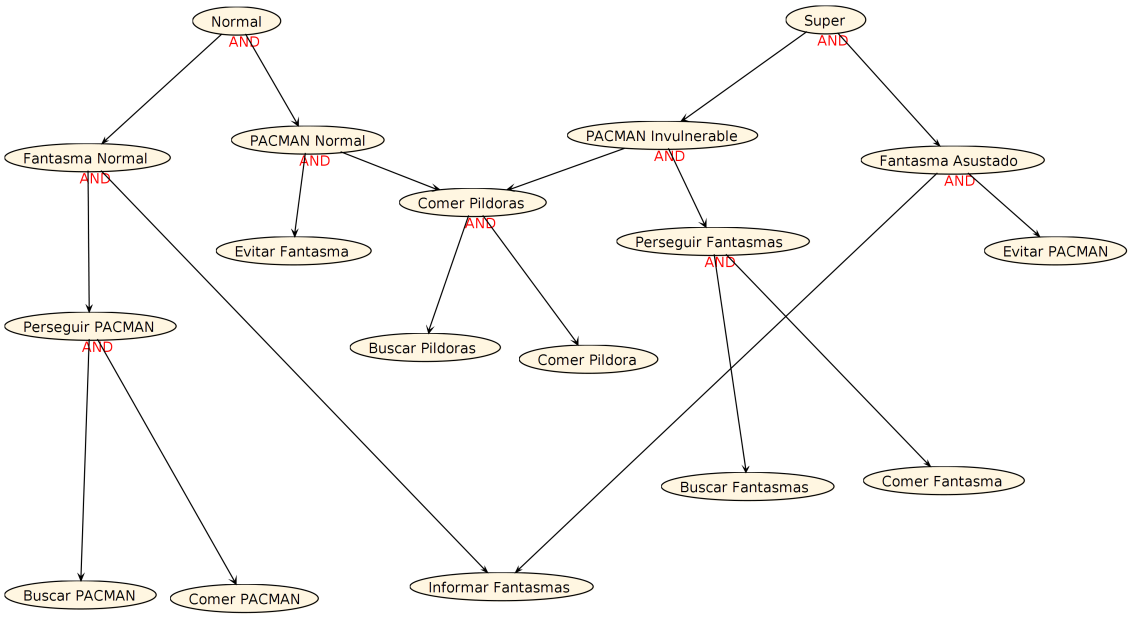
Especificación - Escenarios



Especificación del sistema - Objetivos

- Podemos dividir todos los objetivos del sistema en dos grupos, los que se persiguen en el escenario normal y en el súper
- A su vez los objetivos se organizan según los actores que los persiguen, ya sea PACMAN o los fantasmas
- Los objetivos incluyen: Buscar y comer píldoras, buscar y comer PACMAN y Fantasmas, informarse entre fantasmas

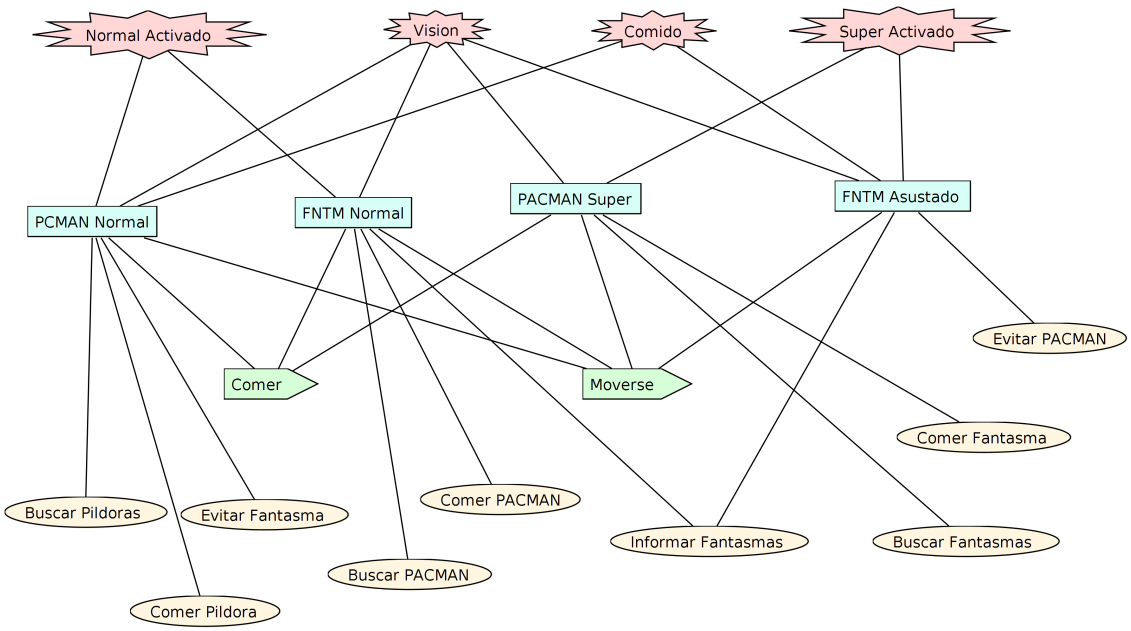
Especificación del sistema - Objetivos



Especificación - Roles/acciones/percepciones

- Podemos distinguir cuatro roles que se corresponden con los diferentes subescenarios identificados
- Como percepciones externas tenemos la activación de los modos normal y super, la visión que reciben PACMAN y Fantasma de su entorno y cuando PACMAN y fantasmas son comidos (esta última percepción podría formar parte de la visión)
- Como acciones podemos identificar el comer píldoras, PACMAN y fantasmas (podrían ser acciones separadas o una genérica) y la acción de movimiento de PACMAN y fantasmas

Especificación - Roles

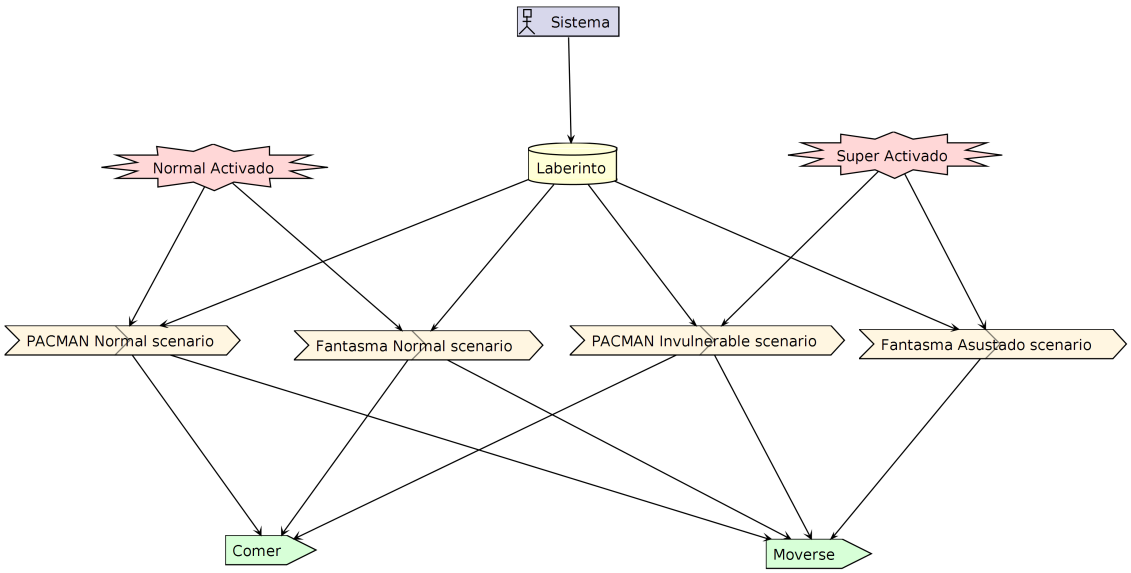


Rol	
Nombre	PACMAN Normal
Descripción	Se encarga de mover a PACMAN por el laberinto comiendo píldoras y huyendo
Evento Iniciador	Modo normal activado
Acciones	Comer píldoras/Movimiento
Información usada	Laberinto
Información Producida	Estado interno
Objetivos	PACMAN normal

Escenario			
Nombre	PACMAN Normal		
Descripción	PACMAN busca píldoras y huye		
Evento	Activación modo normal		
Pasos			
Tipo	**Nombre**	**Rol**	**Datos**
Percepción	Modo normal	Pacman Normal	
Percepción	Visión	Pacman Normal	
Objetivo	Comer píldoras	Pacman Normal	
Objetivo	Evitar Fantasmas	Pacman Normal	
Acción	Comer píldora	Pacman Normal	Laberinto
Acción	Moverse	Pacman Normal	Laberinto

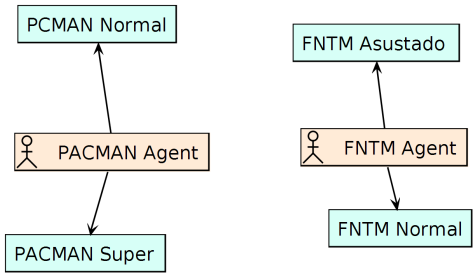
Percepción	
Nombre	Visión
Descripción	Evento que permite obtener información del entorno en la posición actual
Información	Elementos del laberinto visibles
Conocimiento Actualizado	Posición de píldoras/Fantasmas/PACMAN
Fuente	Entorno
Procesamiento	Ninguno
Agentes que responden	Por determinar
Frecuencia	Cada paso de reloj

Acción	
Nombre	Moverse
Descripción	Actualiza la posición de PACMAN/Fantasma en el laberinto
Parámetros	Dirección de movimiento
Duración	Paso de reloj
Fallo	Ninguno
Efectos Laterales	Ninguno



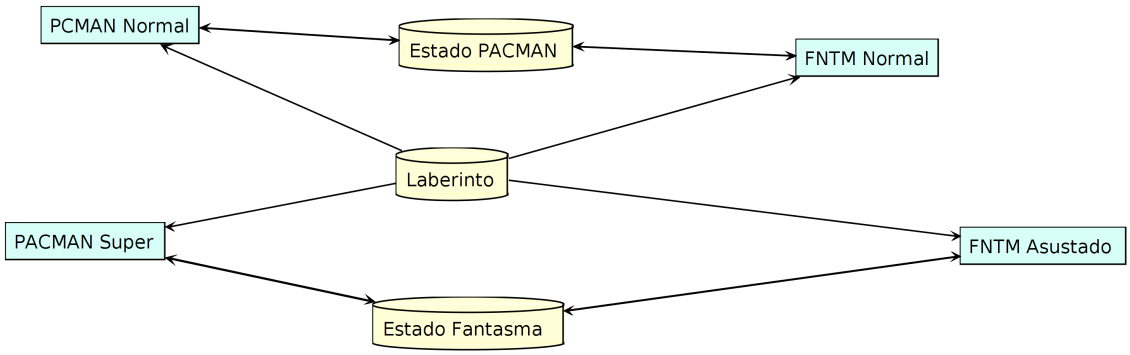
Diseño Arquitectónico - Diagrama roles/agentes

- Identificamos que harán falta dos agentes en el sistema y agrupamos los roles según su coherencia semántica
  - Agente PACMAN: rol PACMAN normal, rol PACMAN súper
  - Agente FANTASMA: rol Fantasma normal, rol Fantasma asustado



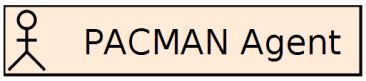
Diseño Arquitectónico - Diagrama acoplamiento datos

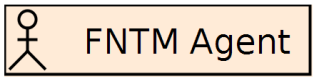
- Las fuentes de datos provienen de:
  - La estructura de datos que guarda la descripción del laberinto
  - La información del estado interno de los agentes que actualizan con las percepciones y las acciones



Diseño Arquitectónico - Diagrama de relacióno

- Los dos tipos de agentes no se comunican en entre sí
- Los fantasmas sí que lo hacen, pero la comunicación entre agentes del mismo tipo no se representa

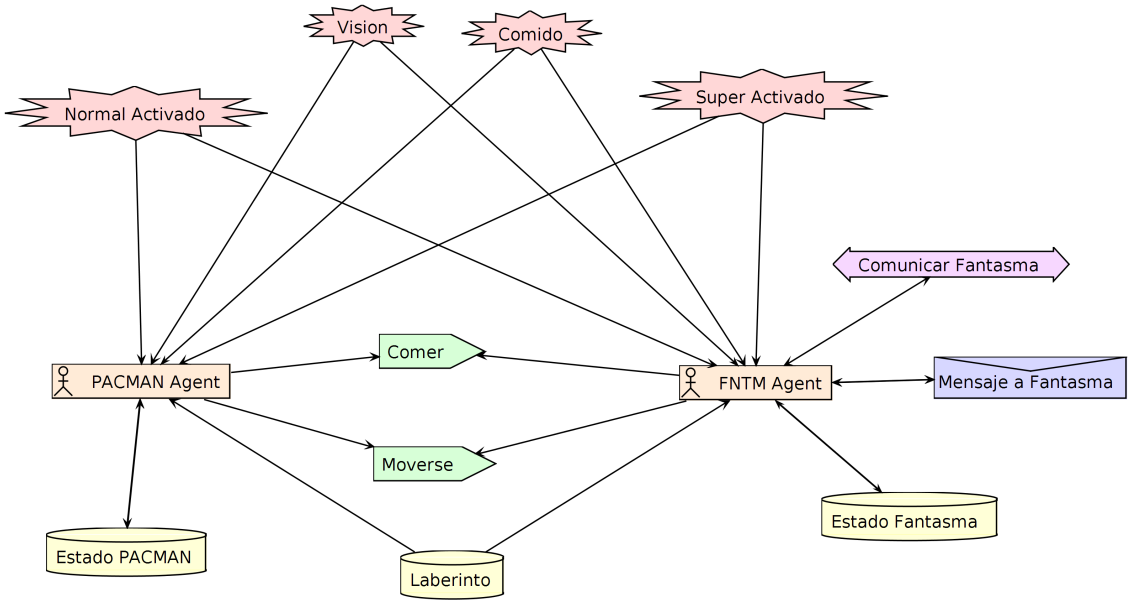




Agente			
Nombre	Fantasma		
Descripción	Se encarga de perseguir a/huir de PACMAN		
Cardinalidad mínima	4	**Cardinalidad máxima**	4
Duración	ilimitada	**Inicialización**	Posición inicial de fantasmas
En caso de fallo	Nada	**Percepciones**	Visión...
Acciones	Moverse/Comer	**Usa datos**	Laberinto, estado interno
Produce datos	Ninguno	**Datos internos**	Por definir
Objetivos	Buscar PACMAN, Comer PACMAN...	**Roles**	Fantasma normal, asustado
Protocolos	Protocolo Informar Fantasmas		

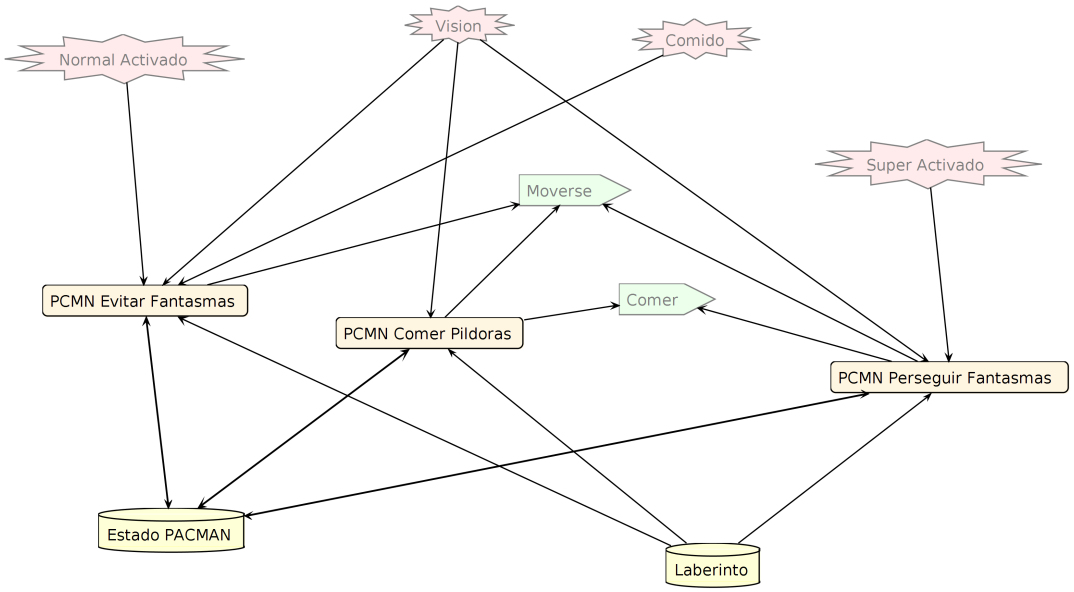
Protocolo	
Nombre	Informar Fantasmas
Descripción	Informa a otros fantasmas de la posición de PACMAN
Mensajes	Enviar Posición
Escenarios	Escenario normal/super
Agentes	Fantasma

Mensaje	
Nombre	Posición PACMAN
Descripción	Informa de la posición de PACMAN
Distribución	De Fantasma a Fantasma
Propósito	Informar de posición PACMAN
Información contenida	Posición actual de PACMAN

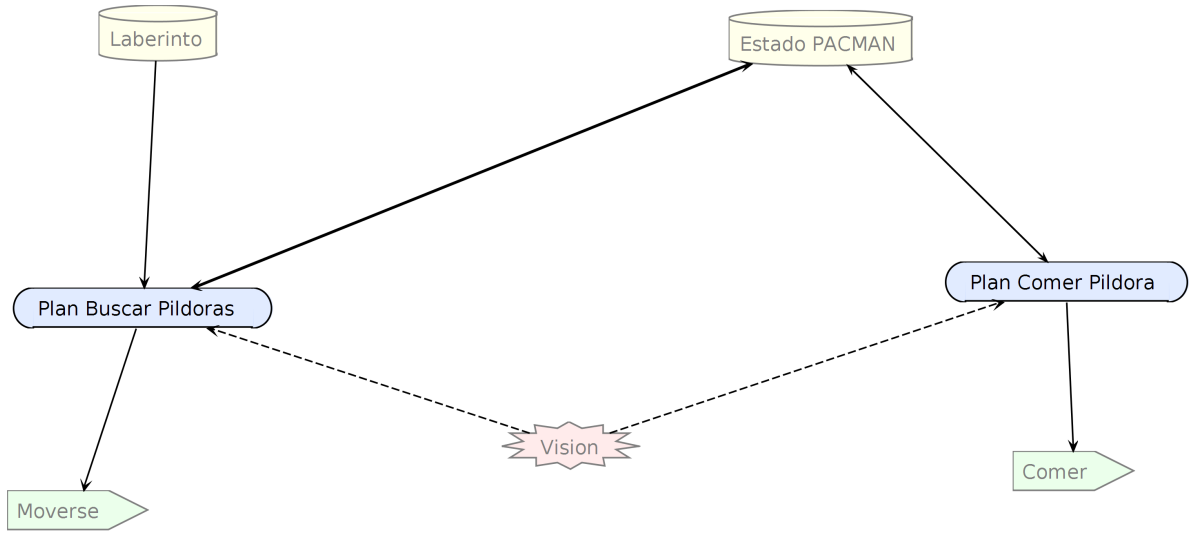


- Identificamos en el agente PACMAN tres capacidades
  1. Evitar Fantasmas
  2. Perseguir
  3. Comer Píldoras
- Detallamos para cada capacidad los planes en los que se descomponen

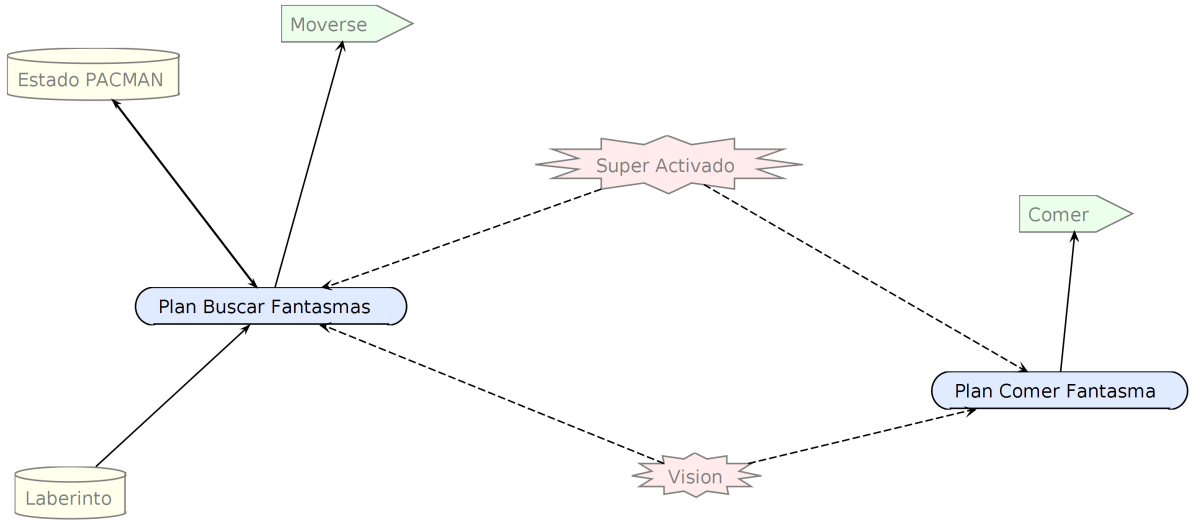
Diseño Detallado - Capacidades PACMAN



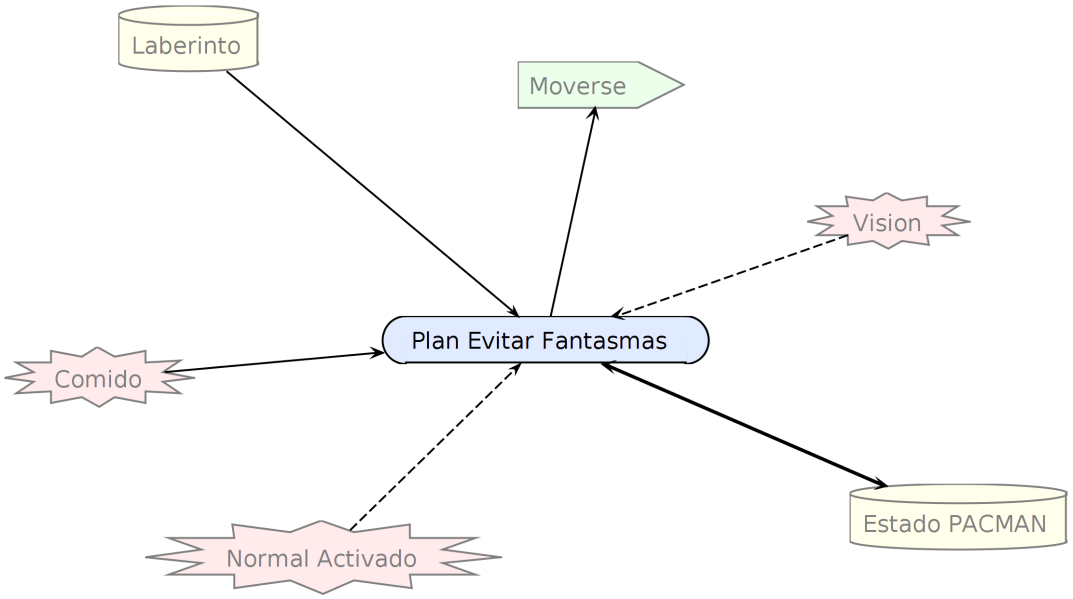
Diseño Detallado - Planes PACMAN - Comer pildoras



Diseño Detallado - Planes PACMAN - Perseguir fantasmas



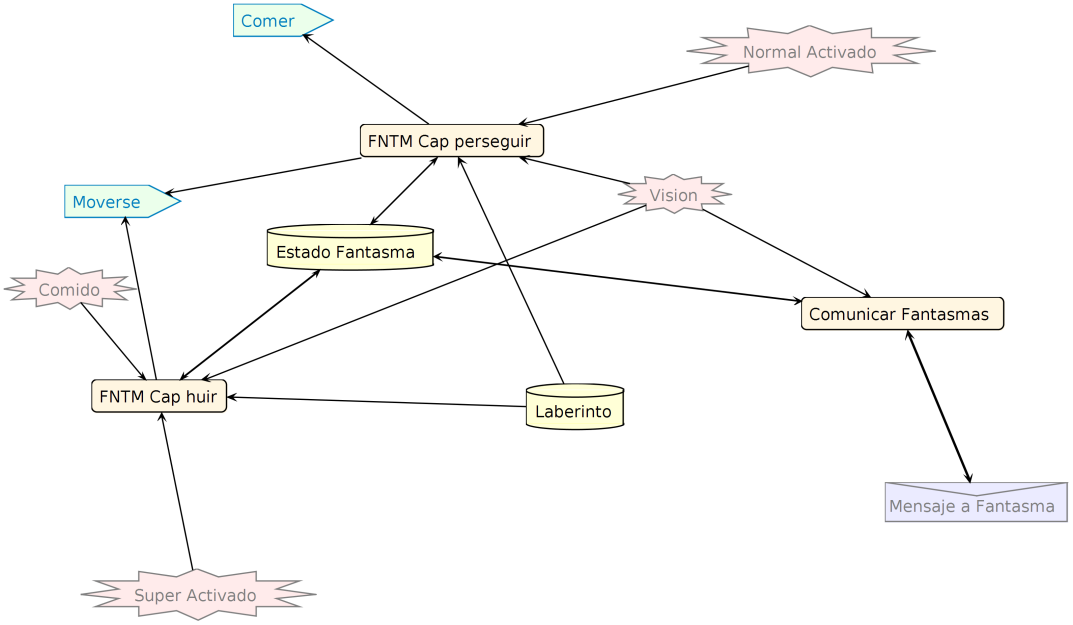
Diseño Detallado - Planes PACMAN - Evitar fantasmas



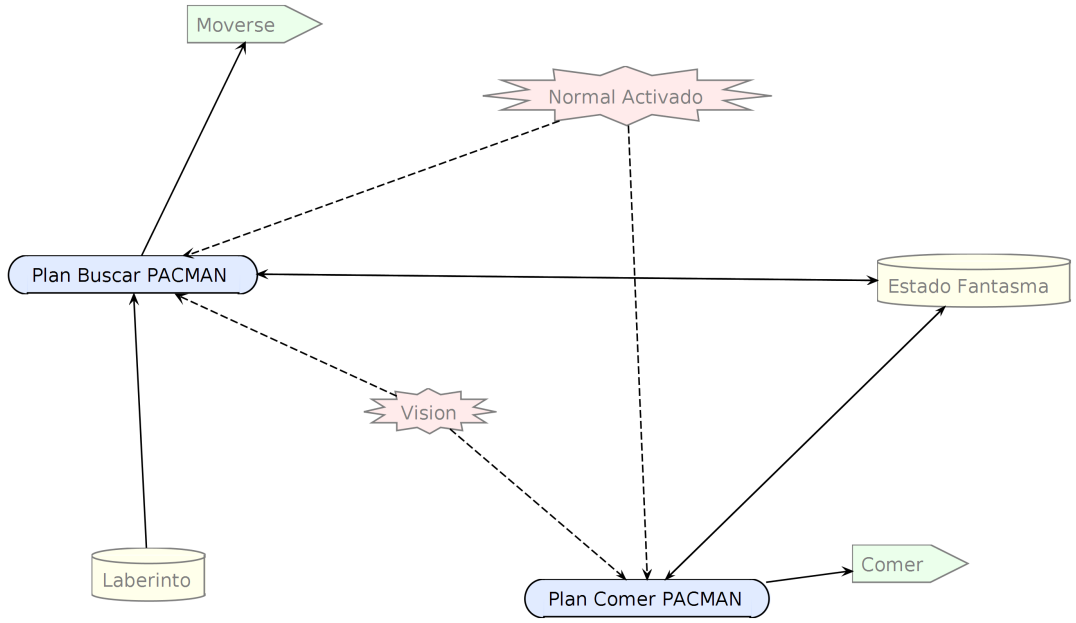
Diseño Detallado - Capacidades Fantasmas

- Identificamos en el agente Fantasma tres capacidades
  1. Perseguir PACMAN
  2. Huir de PACMAN
  3. Comunicarse con otros fantasmas
- Detallamos para cada capacidad los planes en los que se descomponen

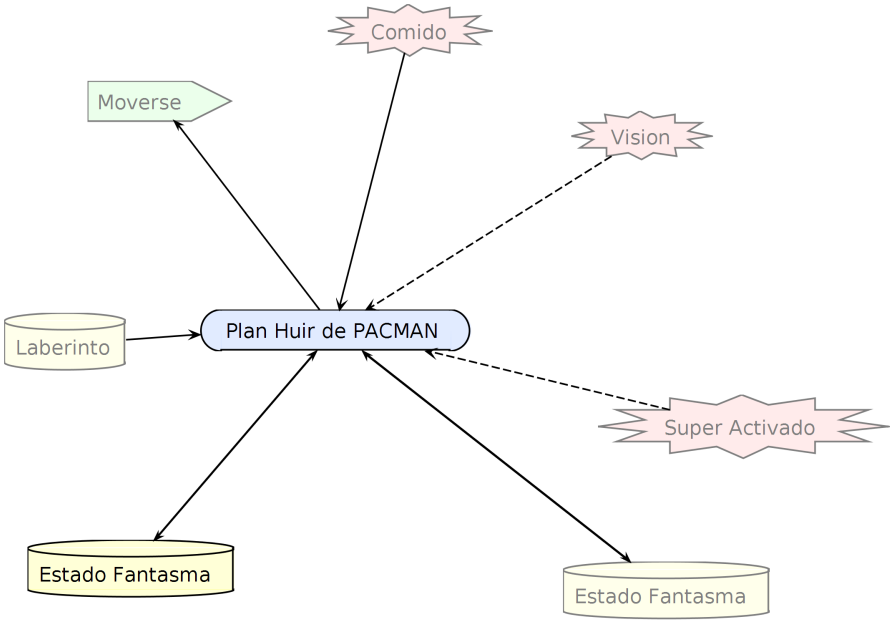
Diseño Detallado - Capacidades Fantasma



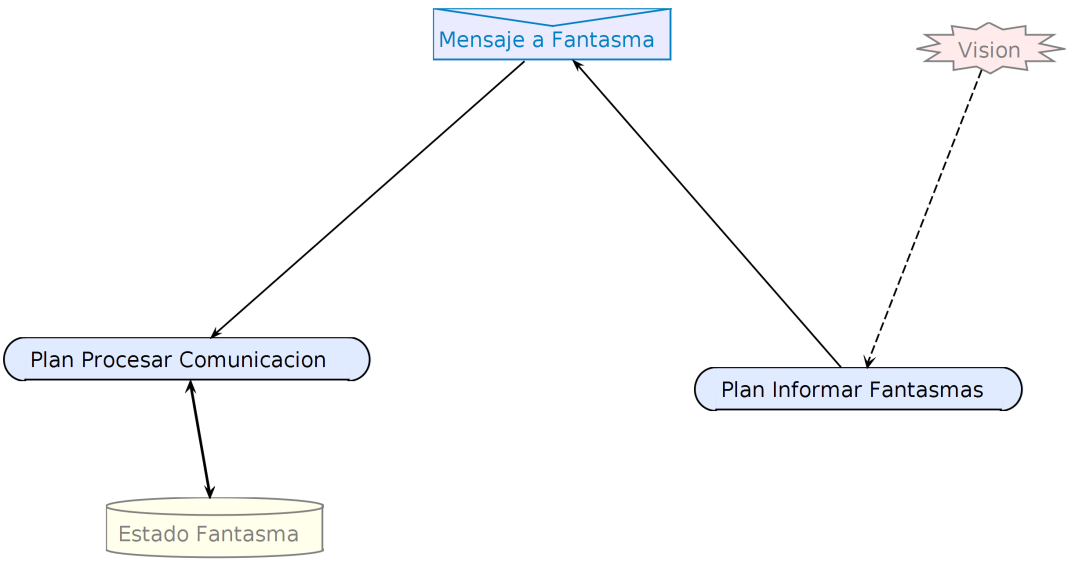
Diseño Detallado - Planes Fantasmas - Normal



Diseño Detallado - Planes Fantasmas - Asustado



Diseño Detallado - Planes Fantasmas - Comunicar fantasmas



Capacidad	
Nombre	Evitar Fantasmas
Descripción	Evita ser comido por los fantasmas
Objetivos	Evitar fantasma
Protocolos	Ninguno
Mensajes Entrantes	Ninguno
Mensajes Salientes	Ninguno
Mensajes Internos	ninguno
Percepciones	visión, comido
Acciones	moverse
Datos usados	Laberinto, estado interno
Datos producidos	ninguno
Datos Internos	Por definir
Planes incluidos	Plan evitar fantasma
Capacidades incluidas	Ninguna

Plan	
Nombre	Plan evitar fantasma
Descripción	Desarrolla las acciones necesarias no ser alcanzado por un fantasma
Iniciador	modo normal
Mensajes Entrantes	Ninguno
Mensajes Salientes	ninguno
Datos usados	Laberinto, estado interno
Datos producidos	Ninguno
Fallo	ninguno
Recuperación de fallo	innecesario
Procedimiento	Código del plan

Datos	
Nombre	Estructura laberinto
Descripción	Representa como es el laberinto
Tipo de datos	grafo
Campos	nodos, conexiones
Persistente	Sí
Externa al sistema	No
Incialización	Laberinto por defecto
Producida por	Sistema externo
Usada por	PACMAN, Fantasmas
Usada cuando	...

7

Comunicación en Sistemas Multiagente

Sociedades de Agentes

Comunicación

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Sistemas Abiertos vs Cerrados
- Comunicación
  - Protocolos, Mensajes, Semántica
- FIPA-ACL
  - Performativas estándar, Protocolos de interacción

Sistemas Abiertos

Comunicación

Sistemas Cerrados

- **Sistema Cerrado**: Todos los estados, transiciones e interacciones del sistema se conocen en tiempo de diseño
- Diseñar un sistema multiagente puede asumir que:
  - los agentes son benevolentes (hacen lo que se les pide)
  - cooperan pasivamente entre ellos para llegar a un objetivo común
- El diseño puede hacerse de arriba a abajo, descomponiendo el problema global en pequeñas tareas
- El nivel social de estos sistemas es débil (o inexistente)

Sistemas Cerrados vs Abiertos

- El paradigma de agentes puede explotar su potencialidad en **entornos realmente abiertos**
- No todos los elementos son conocidos durante el diseño
- Los agentes se desarrollan de manera independiente
- Cada agente proporciona capacidades al resto, los agentes se aglutinan para resolver un problema global
- El problema se resuelve por la interacción entre los agentes (**computation as interaction**)
- El diseño se realiza de abajo hacia arriba

Sistemas Abiertos/Sociedades de Agentes

- Los agentes tienen capacidades utilizables para diferentes propósitos
- Los agentes han de combinar las capacidades de otros agentes para cumplir sus objetivos
- Los agentes han de mantener interacciones con agentes que no son conocidos a priori
- **Problemas:**
  - **Cómo conocer** los servicios ofrecidos por otros agentes
  - **Cómo organizar** las interacciones con otros agentes
  - **Cómo confiar** en agentes que no son conocidos

- **Comunicación entre agentes**
  - Diseño de formas de comunicación de alto nivel entre agentes más allá del paso de parámetros
  - Mensajes, conversaciones/protocolos, lenguajes de contenido

- **Organización dinámica**
  - Modelos y estrategias organizativas que permitan establecer dinámicamente las relaciones entre los agentes
  - Estructuras organizativas, negociación, cooperación, competición

- **Comportamiento social**
  - Modelos que permitan tomar decisiones sobre qué agentes usar en las interacciones (confianza/reputación)
  - Modelos que permitan tomar decisiones sobre como asegurar el comportamiento adecuado de los agentes (convenciones/normas)

Comunicación

Comunicación

- Los agentes han de poder interactuar para resolver problemas
- La interacción no es un simple paso de parámetros
- Una interacción tiene un significado asociado aparte de los valores comunicados
- Los parámetros son evaluados en el contexto del tipo de comunicación

Objetivos de la comunicación

- La comunicación permite que se realicen **labores más complejas** de las que se podrían realizar en solitario
  - Solicitándose la realización de acciones que no son capaces de realizar
  - Solicitando información a otros agentes
  - Compartiendo sus creencias con otros agentes
  - Coordinándose y cooperando para resolver tareas complejas
  - Negociando y compitiendo por recursos

Elementos de la comunicación - Mensajes

- Un **mensaje** estará definido por diferentes elementos que:
  - Determinarán su forma
  - Delimitarán la información que se intercambia
  - Definirán como ha de interpretarse
- Dos elementos necesarios en todo mensaje serán:
  - La identificación del que envía y el que recibe la comunicación
  - El protocolo que se utiliza en el intercambio de mensajes

Elementos de la comunicación - Protocolos

- Niveles de un protocolo de comunicación
  - El nivel más bajo se refiere al medio de interconexión entre los agentes (nivel de transporte) (e.g. `http`)
  - El nivel intermedio se refiere a la sintaxis o el formato en el que el mensaje se envía (lenguaje del contenido, codificación)
  - El nivel superior se refiere a la semántica del mensaje, determinado por el *tipo del mensaje*, el *contenido del mensaje* y la *ontología* a que se refiere el mensaje

Speech Act Theory - Tipos de mensajes

- El estudio de qué tipos de mensajes se pueden intercambiar dos agentes y su semántica entra dentro del ámbito de la lingüística
- Este área es denominada **Teoria de los actos comunicativos** (*Speech Act Theory*)
- Esta teoría ve el lenguaje como acciones (peticiones, sugerencias, acuerdos, respuestas...)
- Por lo tanto, una comunicación supone un cambio sobre el comportamiento y las creencias del que recibe la comunicación

Un acto comunicativo tiene tres aspectos

1. **Locución:** Lo que se dice (la frase, el sonido)
2. **Ilocución:** El significado pretendido de la locución por parte del hablante (la intención del hablante)
3. **Perlocución:** La acción que es resultado de la locución

Por ejemplo: Juan dice a María: "Por favor, cierra la ventana"

- Locución: Sonido
- Ilocución: Para Juan significa una petición
- Perlocución: La ventana es cerrada por María

- Se usa el término **performativa** para denotar la fuerza/intención de una ilocución
- Las podemos clasificar en:
  - **Asertivas/declarativas:** Comunicamos un hecho
  - **Directivas:** Ordenamos algo a alguien (relación cliente-servidor)
  - **Comisivas:** Comunicamos acuerdos
  - **Expresivas:** Expresamos una emoción

- La comunicación entre agentes está basada en la teoría de actos comunicativos
- Los agentes utilizarán un conjunto de performativas establecidas para comunicarse sus intenciones
- La semántica de la performativa permite al agente que recibe el mensaje interpretar su contenido
- Dos estándares de performativas usados
  - **KQML** (Knowledge Query and Manipulation Language)
  - **FIPA-ACL** (Agent Comunication Language)

- Conjunto limitado de performativas (22) con su semántica expresada en lógica de creencias
- Define los elementos de un mensaje:
  - **Tipo de acto comunicativo** (performative)
  - **Participante en la comunicación** (sender, receiver, reply-to)
  - **Contenido del mensaje** (content)
  - **Descripción del contenido** (language, encoding, ontology)
  - **Control de conversación** (protocol, conversation-id, reply-with, in-reply-to, reply-by)

- `request`: Petición de realizar una acción
- `request-when`: Petición de realizar una acción cuando una condición se cumpla
- `request-whenever`: Petición de realizar una acción cada vez que una condición se cumpla
- `propose`: Proposición de realizar una acción cuando ciertas condiciones se cumplan
- `query-if`, `query-ref`: Pregunta al que recibe el mensaje si cree que cierta condición es verdadera o qué elemento cumple esa condición

- `call-for-proposal`: Petición de propuestas de realización de acciones bajo unas precondiciones
- `propagate`: Petición a un agente de que procese y propague un mensaje entre agentes que cumplan una condición
- `proxy`: Petición a un agente de que envíe un mensaje a agentes que cumplan una condición
- `subscribe`: Petición de que un agente informe cuando cierta expresión/objeto cambie su valor

- `inform`: Se informa de que cierta condición es verdadera
- `accept-proposal`, `reject-proposal`: Se acepta o rechaza una propuesta
- `confirm`, `disconfirm`: Comunicamos sobre la veracidad de un hecho a un agente que tiene incertidumbre sobre él
- `agree`: Acuerdo para realizar una acción
- `refuse`: Rechazo a hacer una acción (¿porqué?)
- `cancel`: Cancelación de una acción ya acordada
- `failure`: La acción no se pudo realizar correctamente
- `not-understood`: No te he entendido

- **Inform** y **request** son las performativas primitivas de FIPA, las demás se definen a partir de estas
- Su semántica se define a partir de dos elementos:
  1. Las precondiciones, lo que debe ser cierto para que la comunicación sea efectiva
  2. El efecto racional, lo que el emisor espera como resultado de la comunicación

- El contenido del mensaje es una acción
- Las precondiciones son que el emisor:
  1. Tiene la intención de que el contenido de la acción se realice
  2. Considera que el receptor es capaz de llevar a cabo la acción
  3. No cree que el receptor ya tenga intención de realizar la acción
- El efecto racional es que el receptor pase a tener la intención de realizar la acción

Inform - Semántica

- El contenido del mensaje es una afirmación (statement)
- Las precondiciones son que el emisor:
  1. Considera que el contenido es cierto desde su punto de vista
  2. Considera que el receptor tiene incertidumbre sobre la verdad o falsedad de la afirmación
  3. Tiene la intención de que el receptor también tenga la creencia de que el contenido es cierto
- El efecto racional es que el contenido pase a formar parte de las creencias (estado) del receptor

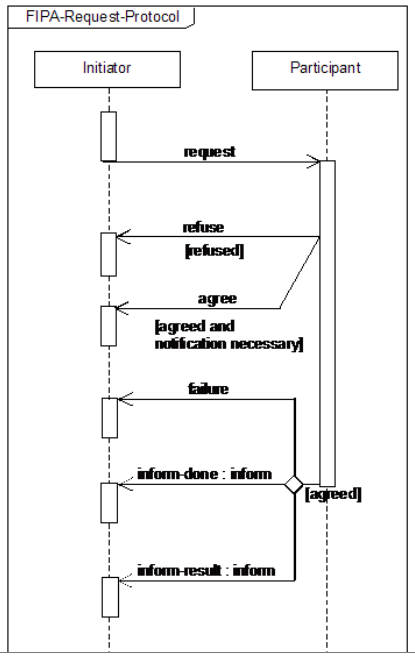
Protocolos de interacción

- La performativas no aparecen aisladas, forman parte de lo que denominaremos protocolos
- Un **protocolo** será una conversación entre agentes que seguirá unas reglas que indican qué performativas se deben intercambiar para cumplir cierto objetivo
- Los protocolos permiten estandarizar las formas de interacción entre agentes, facilitando el diseño

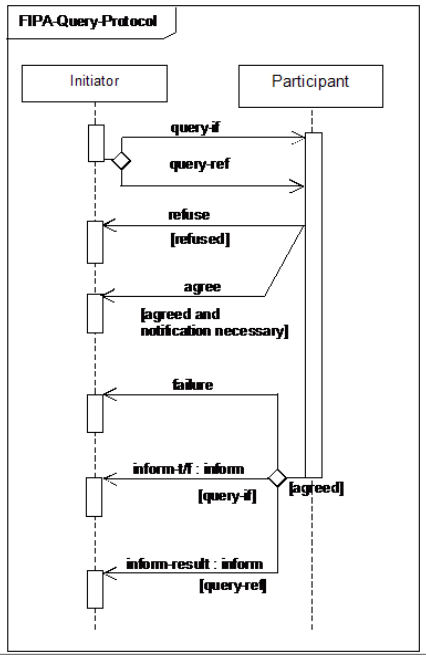
Protocolos de interacción más comunes

- **request:** Conversación en la que se pide a un agente que realice una acción y este nos retorna el resultado si puede
- **request-when:** Conversación en la que pedimos que se realice una acción en ciertas condiciones
- **query:** Preguntamos a un agente si cierta condición es válida y este nos informa, si puede
- **propose:** Proponemos a un agente realizar una acción bajo ciertas condiciones y este acepta o no la propuesta
- **contract-net:** Pedimos a un grupo de agentes propuestas para realizar una tarea y la asignamos al que mejor propuesta nos haga

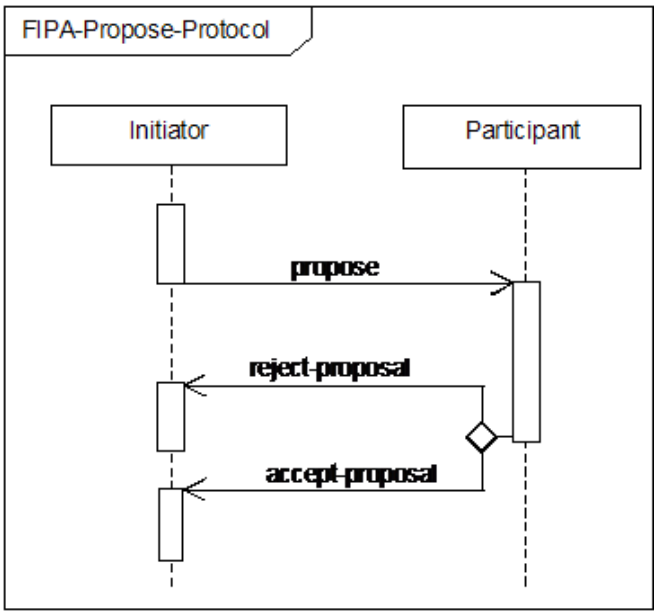
FIPA-Request



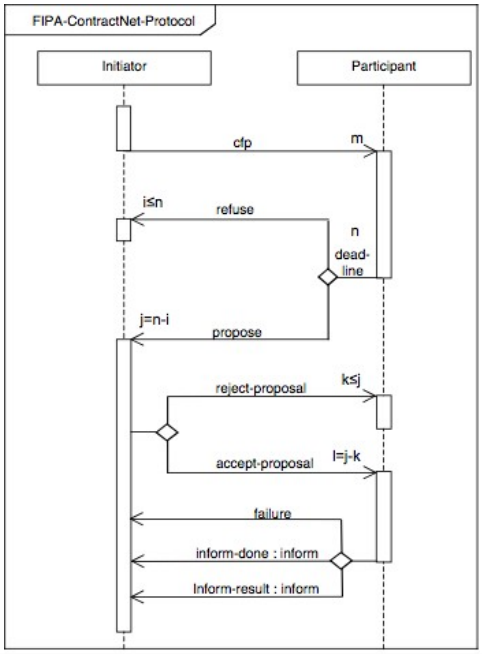
FIPA-Query



FIPA-Propose



FIPA-Contract-Net



Uso de los protocolos

- Estos protocolos los utilizamos explícita o implícitamente en nuestros programas
- Por ejemplo, cualquier llamada a una función es una interacción mediante un protocolo Request, implícitamente asumimos que una función no puede rechazar una petición
- Los protocolos hacen explícitas todas las interacciones que aparecerán cuando las interacciones sean complejas
- Permitirán razonar sobre las interacciones a partir de patrones con una semántica bien definida

Lenguajes de contenido

- Si la performativa nos indica la intención, el contenido nos indicará el objeto de esa intención
- El lenguaje de contenido describe lo que queremos que se haga
- Este lenguaje deberá estar referido a una ontología para saber qué significa cada uno de los términos que aparecen
- Este lenguaje tendrá una sintaxis y una semántica que se deberán establecer aparte
- Un mensaje puede utilizar cualquier tipo de contenido, pero los más habituales incluyen: KIF, RDF y FIPA-SL, también se pueden usar otros como PROLOG, SQL...

8 Ontologías

Ontologías

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Conocimiento en Sistemas Distribuidos
- Ontologías
  - Qué son
  - Porqué son necesarias
  - Tipología
- Proyectos de ontologías

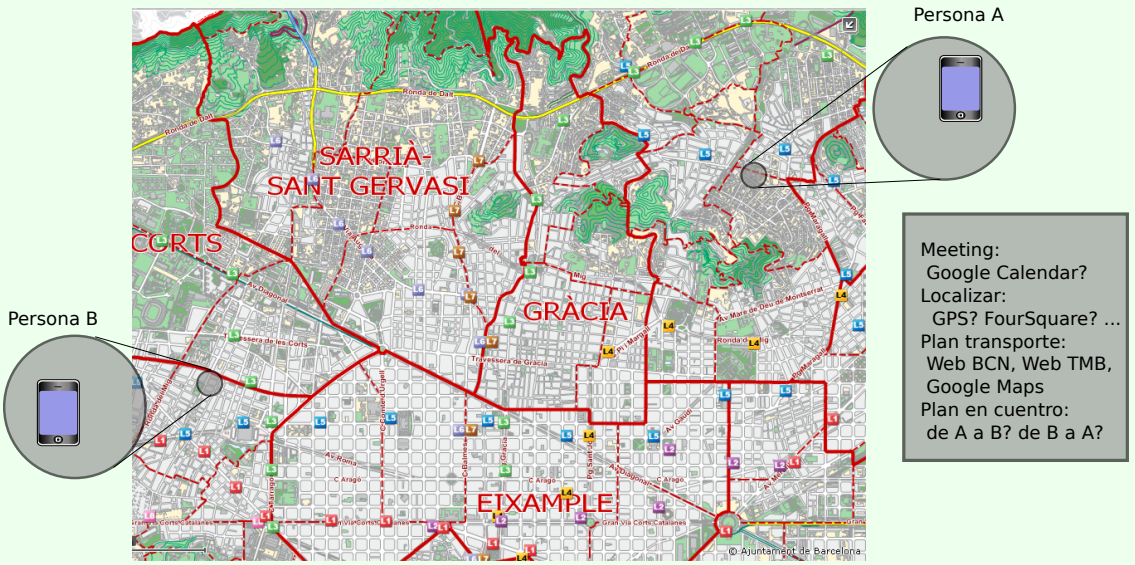
Introducción

Ontologias

Proyectos de Ontologías

- ¿Qué buscamos con una mejor descripción de los servicios?
  - Poder trabajar a un mayor nivel de abstracción
  - Permitir una mejor comunicación entre los servicios (interoperabilidad)
  - Permitir el descubrimiento de servicios (no prefijar con quien nos comunicaremos)
  - Permitir una combinación de servicios a partir de una descripción de objetivos
- Pasamos de servicios web a servicios web semánticos

- Deseamos implementar una aplicación que permita a dos personas encontrarse en una ciudad
- En una aplicación cerrada toda la información para realizar la tarea debería estar disponible
- A partir de lo que debemos hacer:
  - localizar a las personas y el lugar de encuentro
  - decidir cual es la mejor manera de ir
  - decidir el mejor momento del día
- Podríamos utilizar servicios que nos dieran esa información y combinarlos para obtener la solución
- Solo lo podemos hacer si contamos con una descripción formal de la información que debemos usar



- Conocimiento en los servicios:
  - Localizaciones: Coordenada GPS, Dirección, Calle, Ruta, ...
  - Transporte: Medio de transporte, Parada, ...
  - Tiempo, Horario
- Parámetros de los servicios:
  - Qué necesitan (coordenada, dirección, hora, ...)
  - Qué proveen (ruta, parada, horario, ...)
  - Cómo lo proveen (hh:mm:ss, 00^o N, 00^o O, ...)
- Estado en los servicios, Objetivos, Restricciones

- Para establecer relaciones más complejas entre los elementos del estado interno (herencia de propiedades, razonamiento)
- Para representar el estado de otros servicios (estado común, razonamiento sobre otros)
- Para permitir la comunicación con otros servicios más compleja que la llamada a un procedimiento
- Para representar las interacciones entre servicios (organización/cooperación/negociación/delegación/acuerdos)

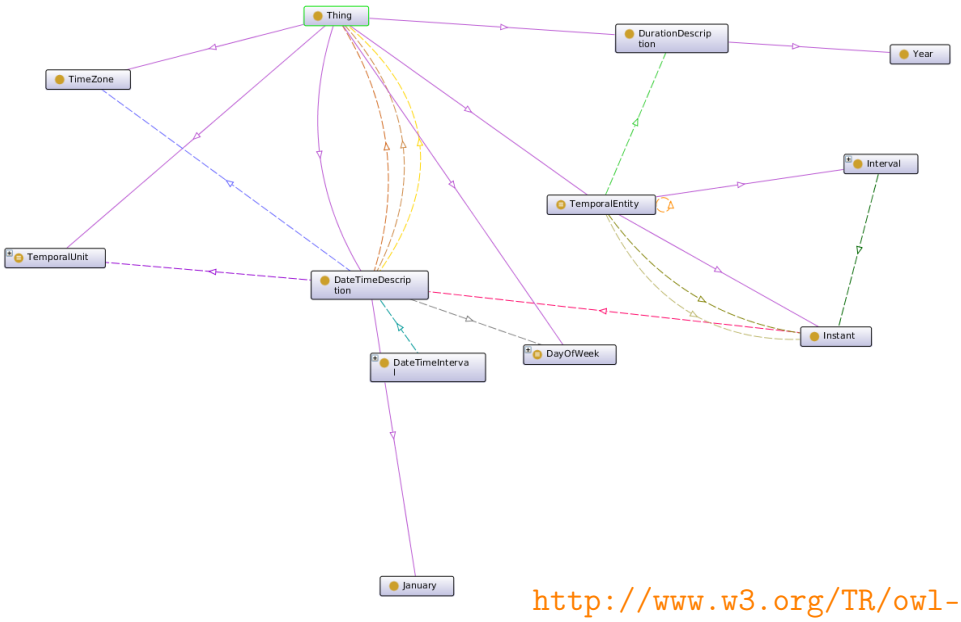
- Las **ontologías** son un esquema de representación que permite esas capacidades
  - Esquema general de representación (basado en lógica)
  - Representación jerárquica del conocimiento
  - Mecanismos complejos de razonamiento
  - Lenguajes estandarizados (interoperabilidad)
  - Acceso a fuentes de conocimiento ya formalizado

- El objeto de la ciencia de la **Ontología** es el estudio de las categorías que existen en un dominio
- El resultado es lo que denominamos una **ontología**

Definición:

"Una ontología es un catálogo de los tipos de cosas que asumimos que existen en un dominio $\mathcal{D}$ desde la perspectiva de alguien que usa un lenguaje $\mathcal{L}$ con el propósito de hablar de $\mathcal{D}$"

- Los elementos de una ontología representan predicados, constantes, conceptos y relaciones pertenecientes a un lenguaje $\mathcal{L}$ cuando se usa para comunicar información sobre $\mathcal{D}$
- Una ontología es pues un vocabulario



http://www.w3.org/TR/owl-time/

- **Separan el conocimiento del dominio del conocimiento operacional**

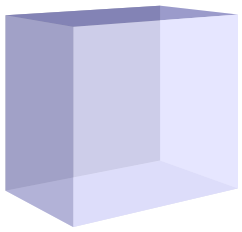
  Permite independizar las técnicas y algoritmos para solucionar un problema del conocimiento concreto del problema

- **Hacen que nuestras suposiciones sobre el dominio se hagan explicitas**

  Facilita replantearse las suposiciones sobre el dominio y ayuda a que otros puedan entender su descripción

¿cubo(x)?

x^3



- **Permiten analizar el conocimiento del dominio**

  Una vez tenemos una especificación del conocimiento podemos analizarlo utilizando métodos formales (para comprobar si es correcto, completo...)

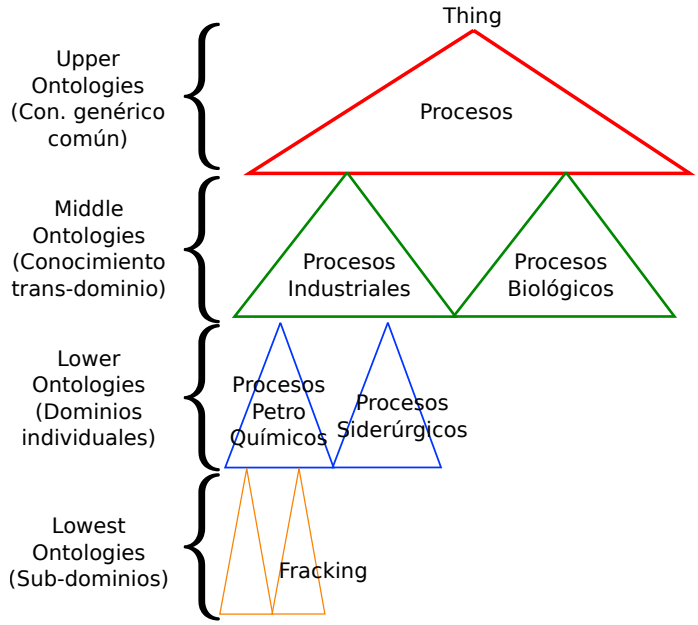
- **Permiten compartir la interpretación de la estructura de la información entre personas/agentes**

  El establecer una ontología sobre un dominio permite que dos agentes puedan entenderse sin ambigüedad y sepan a que se refieren

- **Permiten reusar el conocimiento**

  Hacer una descripción de un dominio permite que esta pueda ser usada por otras aplicaciones que necesiten tratar con ese conocimiento

Tipos de ontologías - Por nivel de abstracción



Tipos de ontologías - Por objetivo

- De dominio:
  - Describen los objetos, relaciones y propiedades que existen en un área de conocimiento específica (ej: Biología, medicina...)
- De tarea:
  - Describen las acciones, objetos, propiedades, relaciones y modificadores que se pueden usar en la resolución de un problema (ej: Jugar al ajedrez, reparar un coche, ...)
- Ask&Tell:
  - Restricción de una ontología para la comunicación entre entidades

Proyectos de ontologías

- Diferentes proyectos de construcción de ontologías con el objetivo de tener representaciones reusables de conocimiento y grandes corpus de información
- Tres tipos de proyectos:
  - Ontologías generales sobre las que desarrollar ontologías de dominio (upper ontologies)
  - Ontologías enciclopédicas que pretenden representar una gran parte del conocimiento humano
  - Ontologías de dominio que representan conocimiento en un ámbito especializado

CYC (1985)/OpenCYC (2003-)

- Ejemplo de ontología enciclopédica
- Objetivo: Formalizar conocimiento de sentido común
- Cientos de miles de conceptos, millones de aserciones
- Definida en diferentes lenguajes de ontologia
- Referenciado a otros corpus de conocimiento (e.g.: DBPedia)
- Subconjuntos especializados (micro teorías)
- cyc.com

WORDNET (1990-)

- Ejemplo de ontología enciclopédica orientada a una aplicación concreta
- Ontología léxica (Organizado según categorías semánticas, etiquetado con categorías sintácticas)
- 95.500 palabras, 70.100 significados
- Redes semánticas
- Inicialmente para inglés, ahora para muchos idiomas
- www.wordvis.com

- Médicas
  - Unified Medical Language System (UMLS)
  - Medical Subject Headings (MeSH)
  - Ontology for Biomedical Investigations (OBI)
- Empresariales
  - TOronto Virtual Enterprise (TOVE)
  - Extensible Business Reporting Language (XBRl)
  - ebXML
- Bibliográficas
  - Dublin Core (DC)
  - Simple Knowledge Organization System (SKOS)
- ...

9

Desarrollo de Ontologías

Desarrollo de Ontologías

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Elementos de una ontología (fundamentos lógicos)
- Fases de una Metodología Sencilla
- Principios de desarrollo de una Ontología

Elementos de un ontología

Metodologías de desarrollo

Principios de desarrollo

- Una ontología es una descripción formal explícita de los **conceptos** (o **clases**) de un dominio
- Los conceptos se describirán a partir de **propiedades** que representarán los atributos, características, relaciones y correspondencias entre los conceptos

- **Relaciones**: Composición de conceptos $R : C_1 \times C_2 \times \cdots \times C_n$

$$Padre\_de : Persona \times Persona$$

- **Funciones**: Correspondencia entre conceptos $F : C_1 \times C_2 \times \cdots \times C_n \rightarrow C_i$

$$preciobillete : Destino \times Fecha \times Fecha \rightarrow Numero$$

- Adicionalmente, estas propiedades tendrán **restricciones** que delimitan diferentes características que las definen (dominio, rango, cardinalidades...)
- Las **instancias** serán elementos identificables que constituirán los individuos concretos que representa la ontología
- Los **axiomas** definen reglas de inferencia que consideramos ciertas para el dominio

- Para poder razonar sobre los elementos de las ontologías necesitamos definir una lógica adecuada
- La lógica de primer orden es demasiado expresiva para este propósito
- **Description Logics** permite razonar sobre clases, subclases, instancias y definiciones
- Es una restricción de lógica de primer orden con una visión conjuntista
- Relacionada con la semántica de la orientación a objetos

- El proceso del desarrollo de una ontología requiere:
  - Definir las **clases** que forman el dominio
  - Organizar las clases en una **jerarquía taxonómica**
  - Definir las **propiedades** de cada clase e indicar las **restricciones** de sus valores
  - Asignar valores a las propiedades para crear **instancias**

- Como cualquier pieza de software, se pueden aplicar metodologías de desarrollo de software a la construcción de ontologías
- Estas metodologías deben ser adaptadas a las particularidades especiales que tienen las ontologías
- **No existe una metodología estándar**
- Diferentes proyectos reales de desarrollo de ontologías han dado lugar a diversas aproximaciones

- *"Ontology Development 101: A Guide to Creating Your First Ontology"*, Noy & McGuinness, (2000)
- Debemos tener en cuenta:
  1. No existe un modo correcto de modelar un dominio. La mejor solución dependerá de la aplicación/problema concreto
  2. El desarrollo de una ontología es un proceso iterativo
  3. Los elementos de la ontología deberían ser cercanos a los conceptos y relaciones que se usan para describir el dominio (generalmente se corresponden a nombres y verbos que aparecen en frases que describen el dominio)

Fase 1

- Determinar el dominio y la cobertura de la ontología
  - ¿Qué dominio cubrirá la ontología?
  - ¿Para qué usaremos la ontología?
  - ¿A que tipos de preguntas ha de poder responder la ontología?
  - ¿Quién usará y mantendrá la ontología?

- Desarrollar una aplicación capaz de recomendar un plan para turistas que quiere pasar unos días en una ciudad
- La ontología debería incluir los diferentes lugares que puede visitar el turista incluyendo actividades culturales y de diversión
- Deberíamos poder conocer los detalles que describen los diferentes lugares como por ejemplo horarios, compra de entradas, precio, abonos, ... y como llegar a ellos
- La ontología podría ser reusada en otras aplicaciones relacionadas

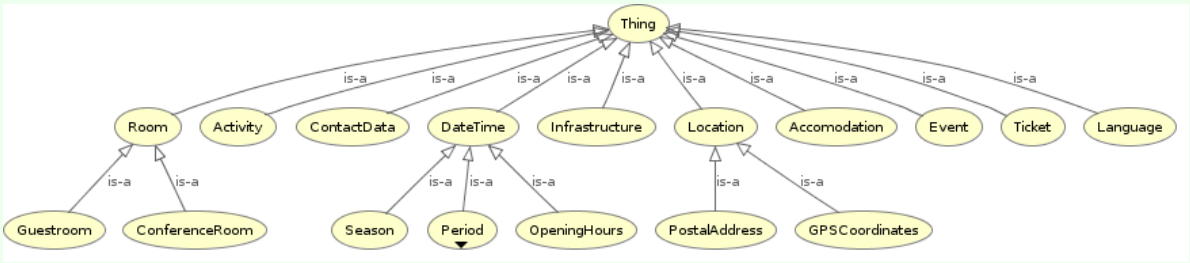
Fase 2

- Considerar la reutilización de ontologías existentes

  Las ontologías se construyen para comunicar conocimiento en dominios, por lo que se desarrollan con la idea de compartición.

  No es necesario rehacer un trabajo que ya está hecho, si existe una ontología sobre el dominio en el que trabajamos, podemos incorporarla

- Buscando en Google podemos encontrar varias ontologías que incluyen el término turismo, analizamos cinco de ellas
- Dos son demasiado generales: Viajes (TravelOntology.owl), elementos de una ciudad (OTN.owl)
- Una demasiado especifica e incompleta (tourism.owl)
- De las dos más adecuadas, una es muy compleja (ETP-tourism.owl) y otra tiene elementos que no nos interesan (e-tourism.owl)
- Podemos aprovechar algunos conceptos de estas dos últimas



Fase 3

- Enumerar los términos importantes en la ontología

  Escribir una lista de términos que podemos usar para referirnos a nuestro dominio, elaborando frases que podríamos utilizar para preguntarnos cosas sobre él o para explicar a alguien información sobre él.

  - ¿Que propiedades tiene esos términos?
  - ¿Que nos gustaría decir sobre ellos?

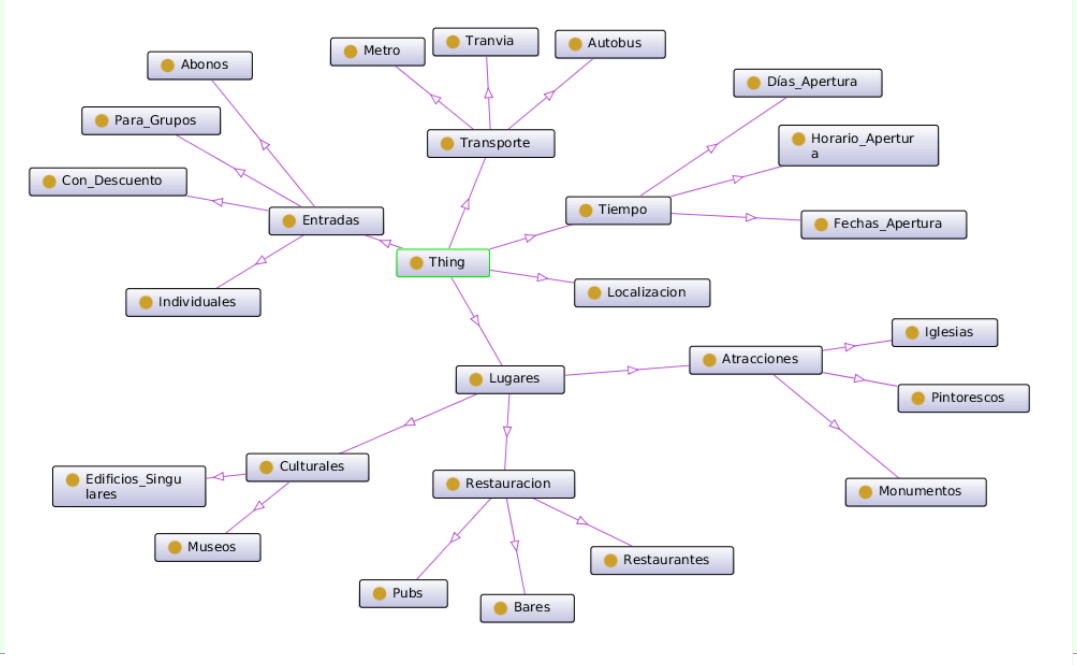
- En nuestra aplicación hablaremos de:
  - Atracciones, servicios, lugares, eventos...
  - Entradas, pagos, descuentos...
  - Horarios, localizaciones, transporte...
- Nos gustaría decir/saber
  - ¿Que horario de visita tiene un lugar, una atracción, un servicio...?
  - ¿Dónde está un lugar, atracción...?
  - ¿Es necesario pagar (¿cómo?) para ir/entrar en un lugar, atracción?
  - ¿Cuál es el tipo de un servicio?
  - ¿Cómo se va a un lugar, atracción...?
  - ...

Fase 4

- Definir las clases y su jerarquía, aproximaciones:
  - De arriba a abajo: Definimos los conceptos más generales y vamos especializándolos
  - De abajo a arriba: Definimos las clases más especificas y la agrupamos según propiedades comunes generalizando
  - Combinación de ambas: Definimos los conceptos más importantes y especializamos y generalizamos para completar la ontología

Ninguno de estos métodos es esencialmente mejor y depende del dominio

- Escogemos generar la jerarquía de arriba a abajo
- Decidimos los conceptos más generales y los especializamos:
  - Lugares (culturales, entretenimiento, atracciones, eventos...)
  - Localizaciones
  - Tiempo (horarios, fechas, días)
  - Transporte (autobús, metro, tranvía, taxi)
  - Entradas (abonos, individuales, con descuento...)



Fase 5

- Definir las propiedades de las clases
  - Debemos describir la estructura interna de las clases
  - Determinaremos una lista de características y en que clases debemos tenerlas
  - Podemos tener muchos tipos de propiedades
    - Propiedades descriptivas, cualidades
    - Propiedades identificadoras, nombres
    - Partes
    - Relaciones con instancias de otras clases

- Propiedades identificadoras: Nombre de lugar, línea de transporte...
- Propiedades descriptivas: hora, calle, descripción de lugar...
- Relaciones:
  - Un transporte y un lugar tienen un horario de apertura
  - Un lugar tiene una localización
  - Museos, edificios singulares e iglesias tienen unos días/fechas de apertura
  - Los lugares culturales necesitan una entrada
  - Los lugares tienen asociados medios de transporte
  - ...

Fase 6

- Definir las características de las propiedades
  - Cardinalidad (número de valores permitidos)
  - Tipo, valores
  - Valores por defecto
  - Obligatoriedad
  - Si es una relación definir la cardinalidad y el rango

- Un lugar tiene solo una localización y un horario de apertura
- Un monumento puede o no necesitar una entrada
- Una entrada tiene un precio con valor real (podría ser 0)
- Una entrada abono tiene un tiempo de validez expresado con un número (días)

Fase 7

- Crear instancias

  Si es necesario (en algún momento lo será) crear las instancias que formarán parte de la ontología

  Esto incluye las instancias estáticas del dominio (constantes y objetos)

- **Claridad y objetividad**
  - Cada concepto de la ontología debe ser definido objetivamente y con una semántica lo más cercana posible a su representación en el lenguaje
- **Completitud**
  - Los términos se deben definir a partir de condiciones suficientes y necesarias
- **Consistencia**
  - Se ha de asegurar que las deducciones obtenidas a partir de la ontología son consistentes con las definiciones de los conceptos

- **Extensibilidad monótona**
  - Incluir nuevas especializaciones y generalizaciones en la ontología no implica revisar conceptos ya definidos
- **Principio de distinción ontológica**
  - Las clases de la ontología son disjuntas
- **Diversificación**
  - Diversificación de las jerarquías incluidas para aprovechar la potencia de la herencia múltiple

- **Modularidad**
  - Reducir la interacción entre módulos semánticamente disjuntos
- **Estandarización de nombres**
  - Definir y seguir un estándar para nomenclatura de los conceptos
- **Minimización de la distancia semántica**
  - Minimizar la distancia semántica entre conceptos fuertemente relacionados. Conceptos similares deben estar agrupados y representados usando los mismos elementos primitivos

Consejos para el desarrollo de una ontología (1)

- Definir una forma común para denominar las clases (por ejemplo, no usar nombres en singular y plural)
  - e.g. Sería confuso tener una clase *personas* y otra *animal*, el plural podría tomarse como un concepto para grupos y el singular para individuos
- Los nombres no son las clases, debemos distinguir la clase del nombre que le damos. Podemos tener sinónimos, pero todos representan a la misma clase
  - e.g. podemos tener una clase llamada *personas* y otra *humanos*, pero sabiendo que se refieren a las mismas entidades

Consejos para el desarrollo de una ontología (2)

- Observar las relaciones de transitividad y comprobar si son correctas (evitarlas si podemos)
  - e.g.: podemos tener una relación *ancestro* que sea transitiva, pero una consulta sobre todos los ancestros de una persona puede llevar a una explosión combinatoria
- Asegurarnos de que la jerarquía está correctamente construida (por ejemplo, sin ciclos)

Consejos para el desarrollo de una ontología (3)

- Todas las subclases de una clase deben estar al mismo nivel de generalidad
  - e.g. podemos tener una clase *persona* que se especialice en *niño*, *joven*, *adulto* y *anciano*, pero no tendría sentido añadir una especialización que fuera *joven europeo*
- No hay un criterio respecto al número de clases, la experiencia dice que un número entre dos y doce es habitual, más clases indicaría que debemos estructurarlas añadiendo más niveles

- **¿Cuando introducir nuevas clases?**

  Suele ser incómodo navegar por jerarquías o muy planas o muy profundas, se debería elegir un punto intermedio:

  - Las nuevas clases tienen propiedades adicionales que no tiene la superclase
  - Tienen restricciones diferentes
  - Participan en relaciones diferentes

  e.g. en un dominio médico puede no tener sentido especializar por el estado civil de una persona, pero en uno sobre impuestos sí

- Decidir si hemos de usar una propiedad o crear una clase

  A veces un atributo es suficientemente importante como para considerar que sus valores diferentes corresponden a objetos diferentes

  e.g. en un dominio médico ser un niño o un adulto lleva a decisiones diferentes, por lo que se les puede considerar entidades distintas

- Decidir donde está el nivel de las instancias

  Pensar cual es nivel mínimo de granularidad que necesitamos en nuestro dominio

- Limitar el ámbito de la ontología
  - La ontología no necesita incluir todas las clases posibles del dominio, solo las necesarias para la aplicación que se desarrolla
  - Tampoco necesitamos incluir todos los atributos, restricciones, relaciones posibles

10

Lenguajes de Ontologías y Web Semántica

Lenguajes de Ontologías

Web Semántica

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- La Web Semántica
- Recursos como información, XML como lenguaje de representación
- RDF, RDFS - Tripletas - Grafos Semánticos
- OWL como lenguaje de ontologías en la Web
- Linked Data, Triplestores, SPARQL

Información y la WWW

La Web Semántica

Web Semántica - RDF

Web Semántica - RDFS

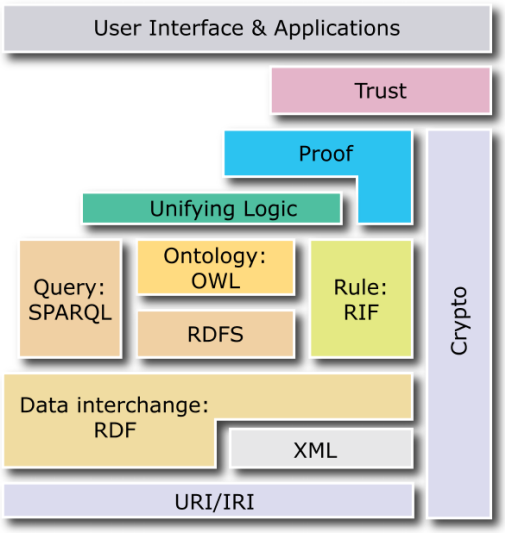
Web Semántica - OWL

Linked Data

- La WWW es una inmensa fuente de información
- **Problema**: está pensada para ser utilizada por personas
  - Lenguaje orientado a la presentación (HTML)
- Asume que los usuarios finales pueden:
  - Reconocer el significado del contenido y sacar conclusiones
  - Inferir nuevo conocimiento utilizando el contexto
  - Entender el conocimiento relacionado
- La evolución de la Web 2.0 a la Web 3.0 pretende que las máquinas aprovechen también esa información

- El que servicios/agentes compartan/adquieran información requiere lenguajes para expresarla
- Este lenguaje:
  - Debe permitir representar el conocimiento ontológico de manera sencilla
  - Debe tener una semántica axiomatizable (eg. DL)
  - Debe permitir el razonamiento automático
  - El coste computacional del razonamiento debe ser razonable

- El poder utilizar la información de la web necesita de un lenguaje de representación uniforme
- El proyecto **W3C Semantic Web** pretende establecer el conjunto de estándares necesarios para hacerlo realidad
- Todo el conocimiento en la web podrá ser descrito usando este esquema
- Utilizados para la representación, consumo y compartición de información entre servicios y agentes
- El objetivo es construir los servicios web semánticos sobre esta tecnología



- Un primer paso para poder representar de manera uniforme la información en la web es poder referenciarla
- En terminología de la web semántica, todo lo que se puede describir en la web es un recurso
- Todo recurso está identificado mediante un **URI** (Uniform Resource Identifier)

Referencia uniforme

- Un URI junta dos conceptos:
  - URN (Uniform Resource Name): Identificador único que permite referenciar un recurso (pero no dice donde esta) (p.ej.: un ISBN de un libro)
  - URL (Uniform Resource Locator): Un identificador único que permite indicar como acceder al recurso

Elementos de un URI

- URI, sintaxis

  `scheme:[//authority][/path][?query][#fragid]`
- Elementos:
  - `scheme`, tipo de URI (http, https:, mailto, imap...)
  - `authority`, habitualmente un servidor
  - `path`, ruta de acceso dentro del servidor
  - `query`, parámetros adicionales
  - `fragid`, una parte constituyente del recurso

Representación uniforme: XML

- XML es un lenguaje para definir lenguajes
- Estándar para interoperación/representación (W3C)
- Mecanismo de almacenamiento de información uniforme
- Lenguaje de intercambio entre aplicaciones (eg: servicios/agentes)
- Puede definir esquemas de representación del conocimiento (lenguajes de ontologías)
- Permite su traducción a otros esquemas (XSLT)

- XML se ha extendido para referenciar definiciones
- Permite construir repositorios de definiciones reutilizables (namespaces)
- Se pueden tomar como vocabularios para dominios concretos

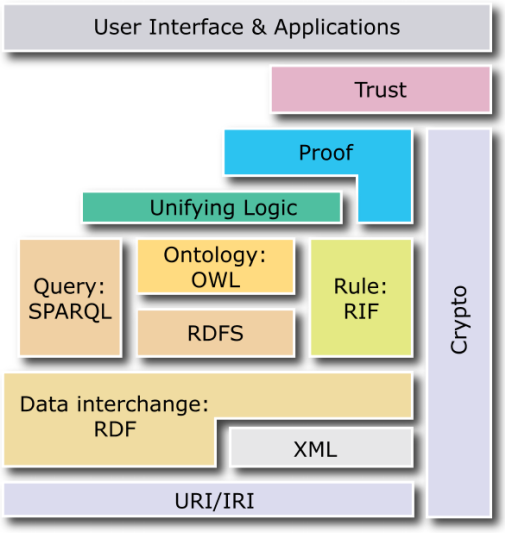
Ejemplo

```
<direccion xmlns="http://definicion_de_direccion">
  <nombre> Juan </nombre>
  ...
</direccion>
```


- Extension de XML usado como lenguaje de definición tipos de datos
- Basado en un conjunto de tipos primitivos (XML Schema Datatypes, XSD)
- Podemos definir nuevos tipos de datos a partir de estos

ejemplo.xml

```
<xsd:complexType name="direccion" >
<xsd:sequence>
<xsd:element name="nombre"   type="xsd:string"/>
<xsd:element name="calle" type="xsd:string"/>
<xsd:element name="ciudad"   type="xsd:string"/>
...
</xsd:sequence>
</xsd:complexType>
```



- XML puede servir como base para un lenguaje de representación del conocimiento
- Los **namespaces** permiten la posibilidad de definiciones compartibles
- Necesitamos además una semántica asociada a esas definiciones (separar la estructura de los datos de su significado)
- RDF es un estándar del W3C definido sobre XML que permite representar información sobre recursos

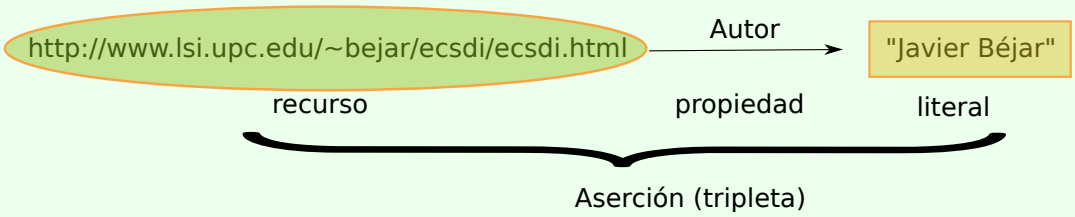
- El elemento principal de RDF es la afirmación (statement)
- Una afirmación define lo que conocemos sobre un recurso:
  - Relacionándolo con otro recurso
  - Declarando los valores de las propiedades de un recurso
- Una afirmación recibe también el nombre de **tripleta** (triplet)

- Una tripleta está formada por:
  - **Sujeto**, un recurso identificado por su URI
  - **Predicado**, una propiedad (también denotada por una URI)
  - **Objeto**, un recurso o literal con el que se define la relación

- **Recursos** (resources): Cualquier cosa que se pueda referenciar (tiene un URI)
- **Propiedades** (properties): Características, atributos o relaciones que se pueden usar para describir recursos, formarán parte de un vocabulario identificado por una URI
- **Literales**: Valores que pertenecen a un tipo de datos primitivo (habitualmente uno de los definidos en xsd)

- **Contenedores/colecciones**: permiten referenciar grupos de recursos
- **Nodos Blancos**: recursos sin identificador usados para agrupar información o como variables en ciertas expresiones

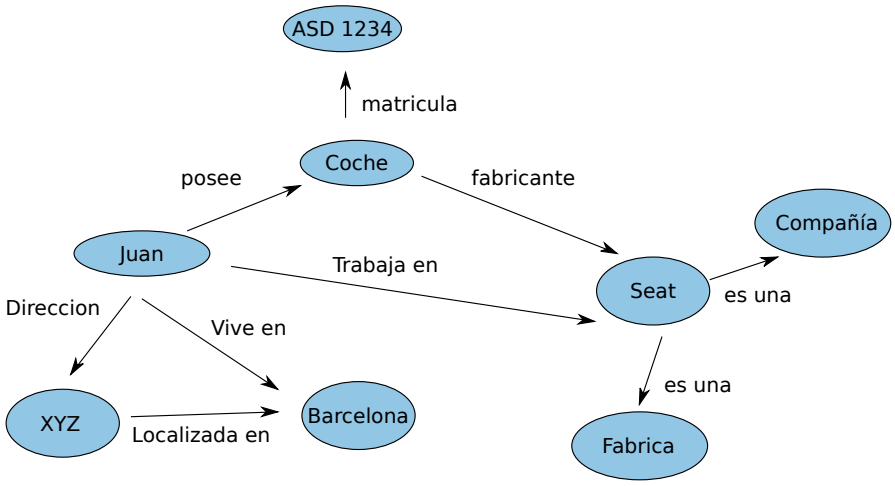
Gráficamente:



en RDF:

```
<rdf:RDF xmlns:s="URI definiciones (propiedad Autor)">
 <rdf:Description about="http:\\ ...\ecsdi.html">
  <s:Autor> Javier Bejar</s:Autor>
 </rdf:Description>
</rdf:RDF>
```


- Las tripletas representan el conocimiento como un grafo
- Los diferentes conceptos representados están interconectados mediante sus relaciones



- Tipos y propiedades serán recursos que tendrán un URI
- Indicamos el tipo de un recurso usando la etiqueta `rdf:type`

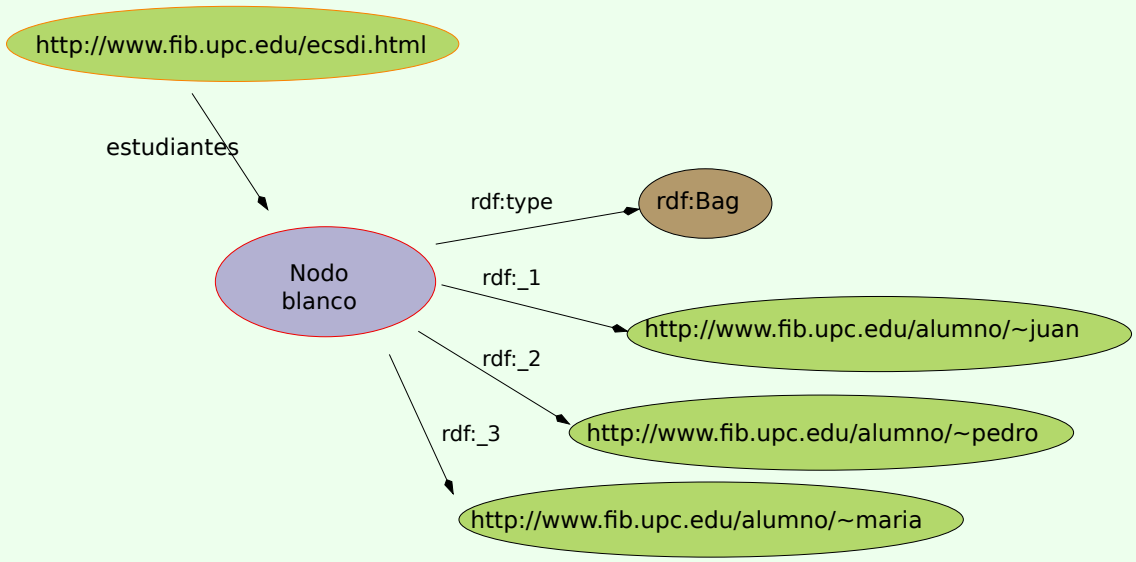
```
juan rdf:type estudiante.
```

- Podemos indicar que algo es una propiedad con `rdf:property` (que usaremos para enlazar recursos/literales)

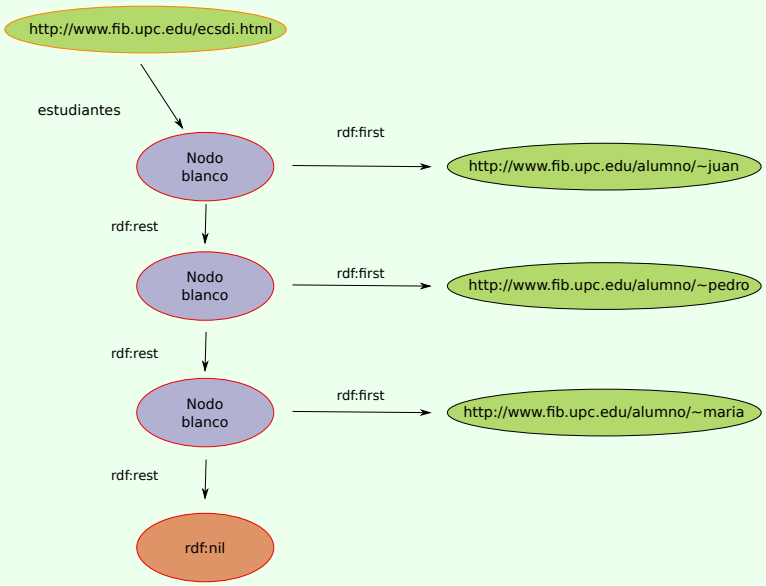
```
vive_en rdf:type rdf:property.
juan vive_en Barcelona.
```

- RDF tiene además la capacidad de definir contenedores y colecciones (grupos de recursos)
- **Contenedores** (no cerrados)
  - `rdf:Bag`: Conjunto no ordenado de recursos o literales
  - `rdf:Sequ`: lista ordenada de recursos o literales
  - `rdf:Alt`: Alternativas para el valor de una propiedad
- **Colecciones** (cerradas)
  - `rdf:List`: Listas enlazadas y cerradas de recursos (`rdf:first`, `rdf:next`, `rdf:nil`)

RDF - ejemplo de contenedores



RDF - ejemplo de coleccion



Vocabulario de RDF

- El estándar W3C de RDF define su vocabulario

Clases:	`rdf:Property`, `rdf:Statement`, `rdf:XMLLiteral`, `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:List`
Propiedades:	`rdf:type`, `rdf:subject`, `rdf:predicate`, `rdf:object`, `rdf:first`, `rdf:rest`, `rdf:_n`, `rdf:value`
Recursos:	`rdf:nil`

https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/

Serialización de RDF

- La forma de serializar RDF es habitualmente XML, pero hay otras alternativas que lo hacen más legible
- En la práctica se ha de leer código en RDF y obviamente no es fácil leerlo directamente en XML
- Varias notaciones alternativas:
  - **TURTLE**(Terse RDF Triple Language)
  - N-triples, JSON-LD, RDFa. TriG, N-quads

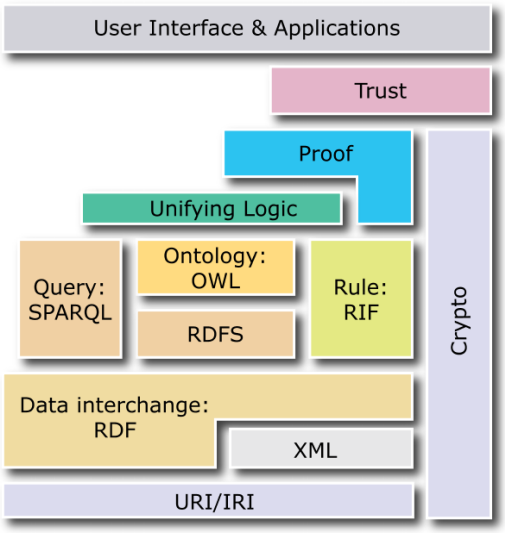
TURTLE

- Utiliza definición de prefijos para aumentar la legibilidad
  `@prefix rdf:<http://www.w3.org/...`
- Permite definir una base común para todos los recursos
  `@base rec:<http://mis.recursos.org/>`
- Substitute `rdf:type` por `a`
- Permite encadenar aserciones sobre un mismo sujeto (`;`)
- Posee una sintaxis simple para los `xsd` y la denotación del idioma de las etiquetas (`@`)

TURTLE - Ejemplo

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix pers:<http://personas.org>.
@prefix org:<http://organizacion.org>.

pers:juan a pers:persona.

pers:juan pers:edad "33"^^xsd:integer;
          pers:nombre "Juan"^^xsd:string.

org:UPC a org:universidad;
     rdf:label "Technical University of Catalonia"@en.

pers:juan org:estudia_en org:UPC.
```


RDFS



- RDF solo nos permite establecer afirmaciones sobre la información
- No permite definir la estructura de la información
- Por ejemplo, podemos decir:
  `pers:juan rdf:type pers:estudiante.`
- Pero no decimos qué es un `estudiante`
- Asumimos implícitamente que es una clase

- RDF Schema (RDFS) es una extensión de RDF que permite declarar clases, propiedades de clases y relaciones entre clases
- RDFS es una colección de recursos RDF que permite describir propiedades de otros recursos RDF (espacio de nombres **rdfs**)
- Es un sistema de clases a partir de las cuales se pueden definir otras clases vía instanciación
- Podemos extenderlo, añadiendo nuevas definiciones
- Estándar W3C: http://www.w3.org/TR/rdf-schema/

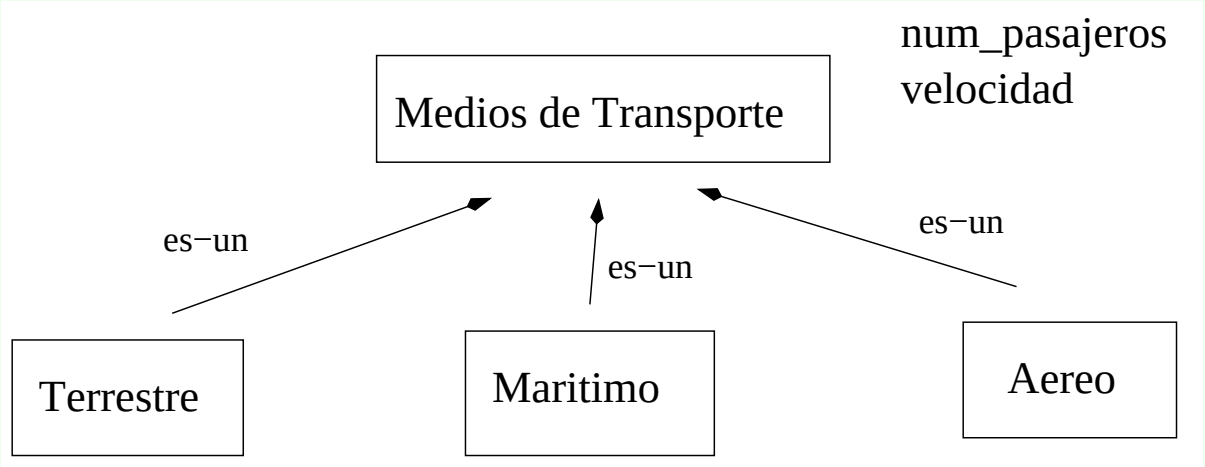
- `rdfs:Resource`: Todo lo que se expresa en RDF es de esta clase (clase padre de la jerarquía)
- `rdfs:Class`: Define conjuntos de recursos
- `rdfs:Literal`: Toda expresión que pertenece a un tipo primitivo (números, strings...)

- `rdfs:Datatype`: Clase de los tipos de datos primitivos
- `rdfs:Container`: Clase de los contenedores
- `rdf:Property`: Toda propiedad de una clase es una instancia de esta clase
- `rdf:type`: Indica que un recurso es un miembro de una clase (instancia-de)

- `rdfs:subClassOf`: Es una propiedad que permite definir la relacion clase/subclase. Su rango es siempre una clase. Es transitiva. Permite herencia de propiedades
- `rdfs:Subproperty`: Indica que una propiedad es una especialización de otra (instancia de `rdf:Property`)
- `rdfs:domain`: Dominio de una propiedad
- `rdfs:range`: Rango de una propiedad

- Metadatos
  - `rdfs:label`: Etiqueta asignada a un recurso
  - `rdfs:comment`: Comentario descriptivo
  - `rdfs:SeeAlso`: Información adicional sobre el recurso

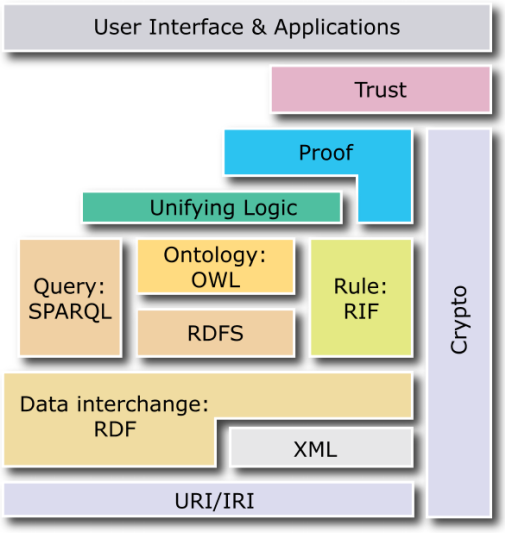
RDFS - Ejemplo - Clase Medios de transporte (I)



RDFS - Ejemplo - Clase Medios de transporte (II)

```
@prefix rdf:<RDF-vocabulario>.
@prefix rdf:<RDFS-vocabulario>.
@prefix xsd:<XML-Datatype>.
@base mt:<http://transporte.org/>.

mt:Medios_de_Transporte a rdfs:Class.

mt:Maritimo rdfs:SubClassOf mt:Medios_de_Transporte.
mt:Terrestre rdfs:SubClassOf mt:Medios_de_Transporte.
...
mt:num_pasajeros a rdf:Property.

mt:num_pasajeros rdfs:domain mt:Medios_de_Transporte;
                 rdfs:range xsd:integer.
```

OWL



Limitaciones de RDF y RDFS

- RDF+RDFS aún no tienen la potencia expresiva necesaria
- No hay semántica para contenedores, colecciones
- No permite hacer comprobaciones para las propiedades de dominio y rango
- Solo se pueden expresar relaciones binarias

Limitaciones de RDF y RDFS

- No permite definir las características de las propiedades (transitiva, reflexiva...)
- No permite restricciones de cardinalidad
- No permite definir clases a partir de otras (unión, intersección) o indicar restricciones sobre clases (disjuntas)

OWL 2

- Diferentes esfuerzos para definir un lenguaje de ontologías sobre RDFS llevan hasta **OWL 2**
- Extienden los elementos de RDFS para:
  - Tener primitivas de frames y Description Logic
  - Tener una semántica definida (DL/Primer Orden)
  - Poder construir demostradores para soportar razonamiento automático
- Define diferentes sintaxis para representar las ontologías
- Estandar W3C: http://www.w3.org/TR/owl2-overview/

OWL 2 - Características

- La ontología pasa a ser un objeto de primer orden ⇒ un recurso
- Las ontologías son importables y extensibles
- Las ontologías se pueden anotar con metadatos
- Podemos establecer restricciones sobre clases y propiedades
- Podemos construir clases a partir de otras clases
- Podemos declarar axiomas de deducción sobre la ontología

OWL 2 - Características

- La extensión esta definida en un nuevo espacio de nombres (`owl`)
- Las ontologias se declaran como objetos de tipo ontología: `owl:Ontology`
- Podemos importar otras ontologias: `owl:imports`
- Podemos utilizar las definiciones de tipos en XML-Schema-Datatype (namespace `xsd`)
- Se definen propiedades para anotar ontologías

OWL 2 - Clases

- Definición de clases: `owl:Class`
- Objeto inicial de la jerarquía `owl:Thing`
- Restricciones sobre clases:
  - `owl:ComplementOf`: Complementario de otra clase
  - `owl:DisjointWith`: Declaración de clases disjuntas
  - `owl:UnionOf`: Clase declarada como unión de clases
  - `owl:IntersectionOf`: Clase declarada como intersección de clases
  - `owl:SameClassAs`: Sinónimo de otra clase

OWL 2 - Clases - Ejemplo

```
Animales a owl:Ontology.

Animal a owl:Class.

ATerrestre rdfs:SubClassOf Animal.
AAcuatico rdfs:SubClassOf Animal.
AAereo rdfs:SubClassOf Animal.
AAereo owl:DisjointWith ATerrestre.

AAnfibio rdfs:SubClassOf Animal.
AAnfibio owl:IntersectionOf [rdfs:first ATerrestre;
                             rdfs:rest [rdfs:first AAcuatico;
                             rdf:rest rdfs:nil]].
```

OWL 2 - Propiedades

- Definicion de relaciones: `owl:ObjectProperty`
- Definicion de atributos: `owl:DataTypeProperty`
- Características de las relaciones/propiedades/instancias:
  - `owl:UniqueProperty`: Cardinalidad 1
  - `owl:TransitiveProperty`, `owl:SymmetricProperty`, `owl:InverseOf`: Transitividad, simetría, inversa
  - `owl:restriction`: Restricciones (p. ej: de cardinalidad `owl:Cardinality`)
  - `owl:sameAs`, `owl:differentFrom`: Individuos iguales/diferentes
  - ...

OWL 2 - Propiedades - Ejemplos

```
Hombre a owl:Class.
Mujer a owl:Class.
Persona a owl:Class;  owl:UnionOf (Hombre Mujer).

Nombre a rdfs:DataProperty; rdfs:domain Persona;
        rdfs:range xsd:string.
Edad a rdfs:DataProperty; rdfs:domain Persona;
        rdfs:range xsd:int.

Progenitor a owl:ObjectProperty; rdfs:domain Persona;
            rdfs:range Persona.

CardProg2 a owl:Restriction;
          owl:onProperty Progenitor;
          owl:cardinality 2.

Hijo_de a owl:ObjectProperty; owl:InverseOf Progenitor.
```

OWL 2 - Instancias

Las instancias se construyen a partir de clases y propiedades

```
juan a Hombre;
     rdfs:comment "Juan es el padre de Luisa";
     Edad 38;
     Nombre "Juan"
     Progenitor luisa.

luisa a Mujer;
     Edad 12;
     Nombre "Luisa"
     Hijo_de juan.
```

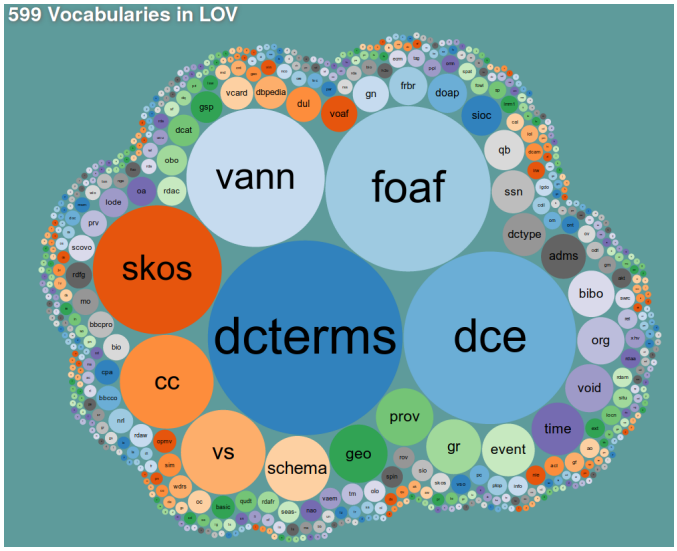
Vocabularios RDFS/OWL

- La extensión del uso de la web semántica ha llevado a popularizar (y a veces a estandarizar) vocabularios
- Ejemplos:
  - SKOS (Simple Knowledge Organization System) (W3C)
  - FOAF (Friend of a Friend)
  - Dublin Core
  - Good Relations
  - Organization Ontology
  - DOAP (Description of a Project)
  - Basic Geo Vocabulary

Repositorios de ontologías RDFS/OWL

- DAML (http://www.daml.org/ontologies/)
  - 282 Ontologías públicas escritas en DAML+OIL/OWL
  - academic department, Actors, address book, airport, Bibliography, Biology, Chemistry, Clothing, Weather, ...
- AberOWL ontology repository (http://aber-owl.net/)
- BioPortal (http://bioportal.bioontology.org/)
  - Más de 300 ontologías en biología
- Ontohub (http://ontohub.org/ontologies)
  - Cerca de 22.000 ontologías
- http://schema.org (Google, Microsoft, Yahoo, ...)

Linked Open Vocabularies

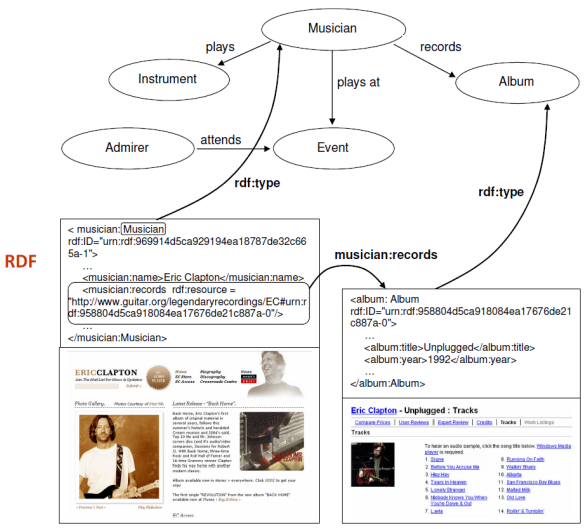


https://lov.linkeddata.es/dataset/lov/

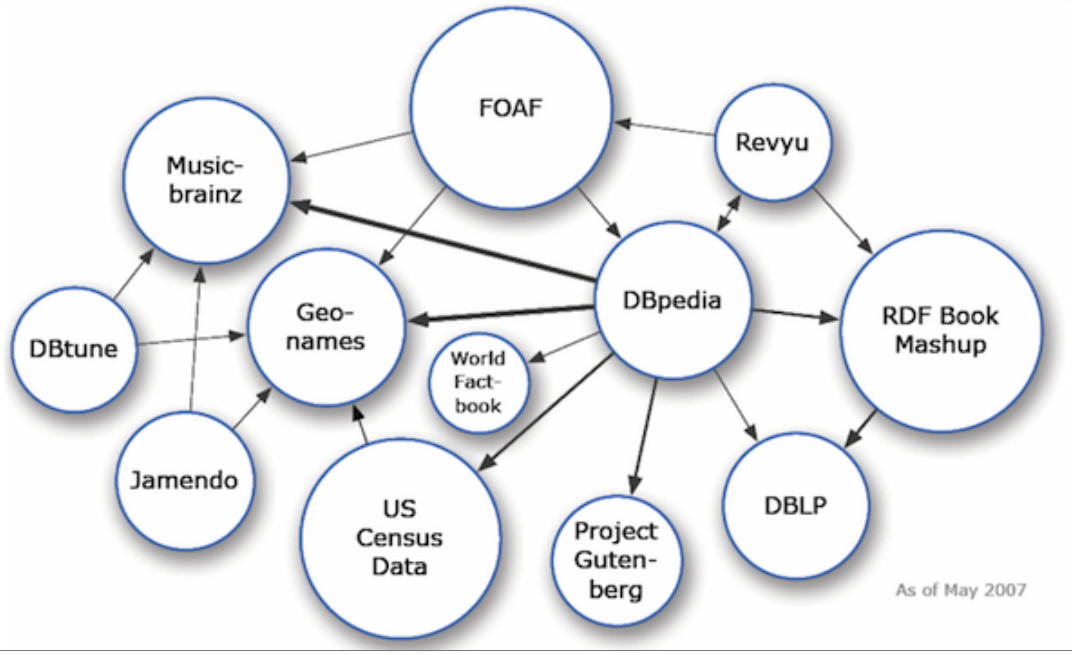
Ejemplos de vocabularios

- Friend of a Friend vocabulary
- Description of a Project (DOAP)
- European Skills, Competences, qualifications and Occupations (ESCO)
- Air Traffic Management Vocabulary
- Drammar: a comprehensive ontology of drama
- domOS Common Ontology (IoT)

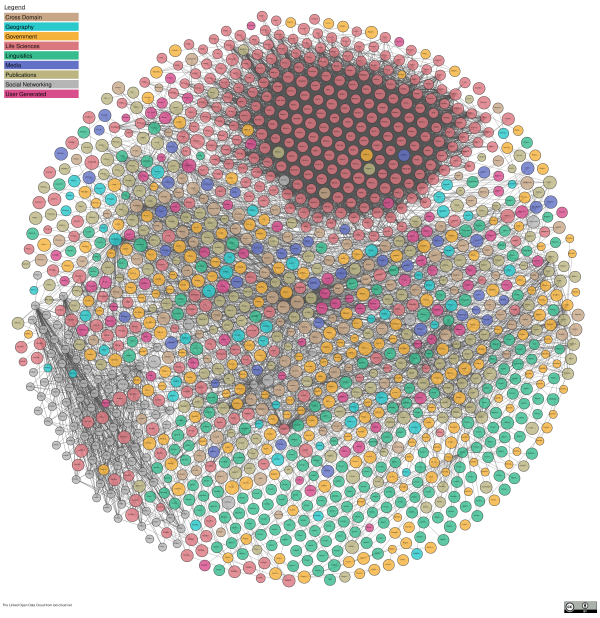
- El uso de ontologías/vocabularios comunes permite publicar datos de manera abierta
- Todo tipo de conocimiento puede ser descrito para su acceso
- Diferentes organizaciones pueden describir su información y enlazarla usando las mismas clases y atributos
- Disponer de esta información permite usarla en nuevas aplicaciones y de nuevas formas



Linked Data 2007



Linked Data 2021 - http://lod-cloud.net/



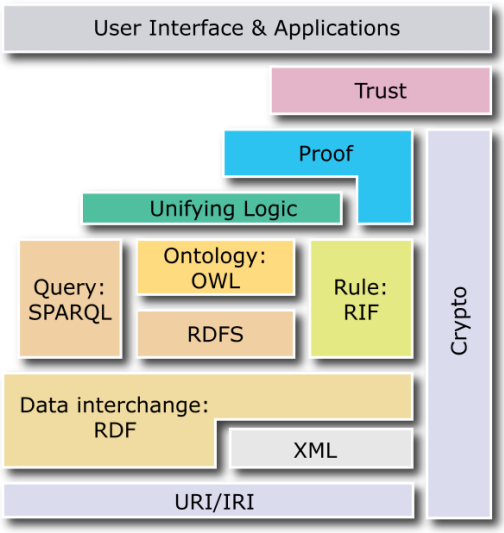
Internet como Base de Datos - SPARQL

- Los datos forman una gigantesca base de datos distribuida
- Cada elemento (recurso) esta descrito mediante un URI que le sirve de identificador
- Las propiedades/aserciones enlazan los recursos entre si
- El lenguaje de consulta **SPARQL** (similar a SQL) permite acceder a esta información
- Los puntos de conexión para lanzar las consultas se denominan **SPARQL endpoints**
  - eg.: DBPedia (`http://dbpedia.org/sparql`)

Triple Stores

- Una **Triple Store** equivale a un RDBMS para linked data
- Almacena la información representada en grafos RDF
- Las operaciones que se pueden hacer sobre una TS son similares a las de una BD relacional
- El lenguaje de manipulación (**SPARQL**) está definido como un estándar por el W3C (version 1.1)
- Estándar W3C: http://www.w3.org/TR/sparql11-query/

SPARQL



SPARQL

- **SPARQL** es un lenguaje de consultas para grafos RDF
- Su sintaxis es similar a SQL
- No requiere que haya un esquema tabular para los datos
- Hace más sencilla la consulta a los datos a través sus relaciones
- Las consultas se resuelven mediante coincidencia de patrones (pattern matching) en el grafo

SPARQL - Consultas

- SPARQL define cuatro tipos de consultas:
  - `SELECT`: permite obtener una lista de tripletas que coinciden con unas propiedades
  - `ASK`: permite saber si existe alguna instancia que cumpla unas propiedades
  - `DESCRIBE`: retorna algunas propiedades del URI que corresponde a la consulta
  - `CONSTRUCT`: permite construir un grafo RDF a partir de los resultados de la consulta

SPARQL - Actualizacion

- SPARQL define cuatro tipos de operaciones de modificación:
  - `INSERT DATA`: Permite insertar nuevas tripletas
  - `INSERT`: Permite insertar nuevas tripletas o mover tripletas de un grafo RDF a otro
  - `DELETE DATA`: Permite borrar tripletas
  - `DELETE`: Permite borrar tripletas según un patrón
- Existen otras operaciones, pero el estándar no obliga a implementarlas

SPARQL - INSERT DATA - Ejemplo

Añadimos una persona a un grafo con algunas propiedades

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ejemplo: <http://ejemplo.org/>
INSERT DATA
         {
          ejemplo:maria a foaf:person;
            foaf:name "Maria";
            foaf:age 27 .
}
```

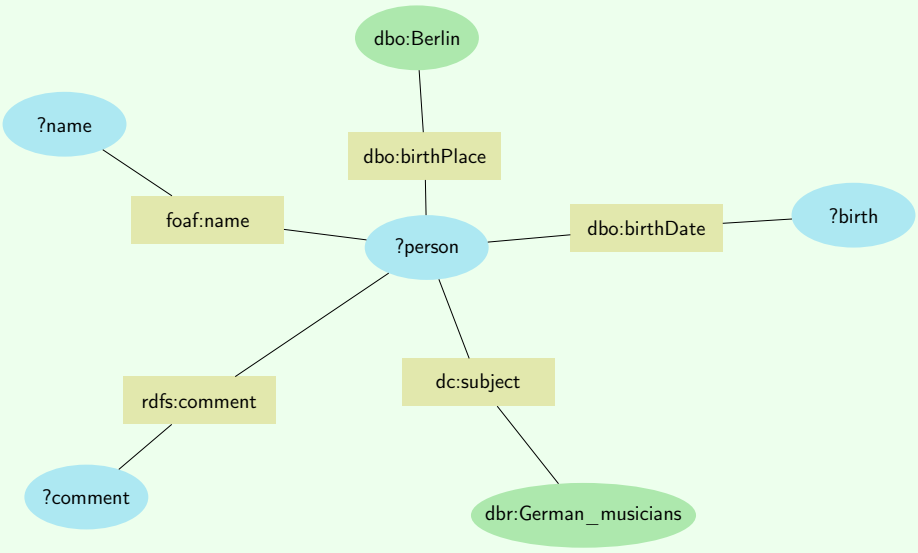
- Una consulta SELECT esta compuesta por:
  - BASE: URI de la TS sobre la que hacer la consulta
  - PREFIX: Lista de prefijos a usar para simplificar la referencia a los espacios de nombre
  - SELECT: descripción del resultado de la consulta (qué variables) (podemos usar el modificador DISTINCT)
  - FROM: Grafo en el que hacer la consulta
  - WHERE: Patrón de la consulta
  - Modificadores: ORDER BY, GROUP BY, LIMIT, OFFSET, ...

SELECT - Ejemplo - Músicos alemanes nacidos en Berlín

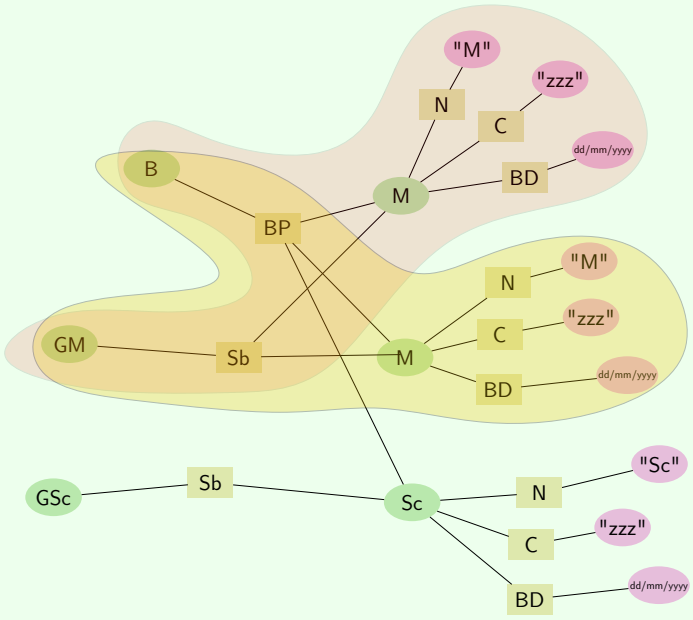
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?birth ?city ?person ?name ?description
WHERE {
   ?person dct:subject dbc:German_musicians.
   ?person  dbo:birthDate ?birth.
   ?person dbo:birthPlace ?city.
    ?person foaf:name ?name.
    ?person rdfs:comment ?description .
}
```

http://factforge.net/sparql

SELECT - Músicos alemanes nacidos en Berlín - Query



SELECT - Músicos alemanes nacidos en Berlín - Matching



SPARQL - SELECT

- En la cláusula se pueden especificar otros elementos:
  - `optional`: Indicando que una parte de la consulta es opcional (queremos el resultado aunque no se cumpla)
  - `filter`: queremos aplicar un filtro a los valores de las variables de la consulta mediante una condición sobre su valor o una expresión regular
  - `union`: queremos que la consulta coincida con alguno de los patrones que indicamos

SPARQL - SELECT - Ejemplo

Personas con su nombre, correo (si es un .com) y fecha de nacimiento (si esta)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX myont: <http://my.ontology.org/my-ont#>
SELECT DISTINCT *
FROM <http://mi.grafo.org/personas.rdf>
WHERE {
        ?p myont:nombre ?n .
        ?p foaf:mbox ?mail.
        optional {
            ?p myont:fnacim ?fn.
        }
        filter (regexp(str(?mail),".com"))
}
```

SPARQL - SELECT - Ejemplo

100 Personas con su nombre, que tengan correo y/o teléfono y fecha de nacimiento posterior a 1/1/1990

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX myont: <http://my.ontology.org/my-ont#>
SELECT DISTINCT *
FROM <http://mi.grafo.org/personas.rdf>
WHERE {
   ?p myont:nombre ?n .
   { {?p foaf:mbox ?mail.}
     union
     {?p myont:telefono ?tf.}}
   ?p myont:fnacim ?fn.
   filter (xsd:date(str(?fn)) > "1990-1-1"^^xsd:date).}
LIMIT 100
```

SPARQL - SELECT - Ejemplo

Personas con su nombre y ordenadas por edad

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX myont: <http://my.ontology.org/my-ont#>
SELECT  ?n ?e
FROM <http://mi.grafo.org/personas.rdf>
WHERE {
       ?p myont:nombre ?n .
       ?p myont:edad ?e.
       filter (?e > 18).
}
ORDER BY ?e
```


SPARQL - Construct

- La sentencia `CONSTRUCT` permite añadir nuevas tripletas a partir de una consulta
- La cláusula `construct` permite indicar qué aserciones se han de construir a partir de las variables instanciadas en la consulta
- Esto permite definir reglas de deducción arbitrarias que permiten explicitar nueva información a partir de la consulta

SPARQL - CONSTRUCT - Ejemplo

Transformar datos de FOAF a mi ontología

```
PREFIX myont: <http://my.ontology.org/my-ont>
CONSTRUCT {
    ?p myont:nombre ?n.
    ?p myont:correo ?m.
}
WHERE {
    ?p foaf:name ?n.
    ?p foaf:mbox ?m.
}
```


SPARQL - CONSTRUCT - Ejemplo

Dos personas que viven en la misma dirección son vecinos

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX myont: <http://my.ontology.org/my-ont>
CONSTRUCT {
    ?p1 myont:vecino ?p2.
}
WHERE {
    ?p1 myont:direccion ?d.
    ?p2 miont:direccion ?d.
}
```


SPARQL y deducción

- Los motores de SPARQL pueden utilizar deducción al realizar las consultas
- Hay definidos diferentes niveles de deducción:
  - RDF
  - RDFS
  - OWL DL, Full
  - OWL2 RL, QL y EL
  - Rule Interchange Format (escribimos nuestras propias reglas de deducción)

11

Organización en Sistemas Multiagente

Estructura Social/Organización

Organización en SMA

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Sistemas Cerrados vs Sistemas Abiertos
- Coordinacion: Cooperación vs Negociación
- Técnicas de resolución cooperativa
- Competición = Negociación
  - Teoría de Juegos
  - Técnicas Heurísticas

Organización

- Los agentes/servicios autónomos deben tomar decisiones en tiempo de ejecución y ser capaces de una coordinación dinámica
- Necesitarán compartir
  - Tareas
  - Información
- Cuando los agentes no están diseñados por las mismas personas, pueden tener objetivos distintos
- **Agentes benevolentes** vs **agentes egoístas**

Organización - Sistemas cerrados

- Si los agentes son benevolentes tenemos un sistema cerrado (o casi)
- Asumimos que todos los agentes tienen el mismo objetivo
- Siempre hacen lo que se les pide
- Podemos usar técnicas simples para organizar y repartir tareas entre los agentes
- Esto simplifica mucho el diseño del sistema
- **Resolución de problemas distribuidos cooperativa**

Organización - Sistemas abiertos

- En sistemas abiertos tenemos agentes que representan a individuos u organizaciones (cada uno con sus intereses)
- Se asume que los agentes actuarán según esos intereses (incluso a expensas de los otros)
- Se ha de asumir que pueden aparecer conflictos
- Esto complica mucho el diseño del sistema
- Es necesario un **comportamiento estratégico** (Teoría de juegos)

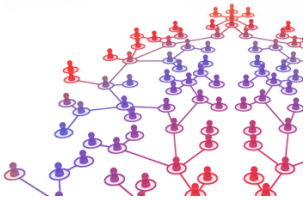
- Podemos definir **Coherencia** como:

  La medida en la que un sistema se comporta como una unidad según algún criterio

- Diferentes medidas: calidad de solución, eficiencia en recursos, ...

- Podemos definir **Coordinación** como:

  El grado en el cual los agentes pueden evitar comportamientos indeseados sincronizando y alineando sus actividades



- Relacionada con otros dos conceptos:
  - Cooperación: Coordinación entre agentes cuyos objetivos son comunes
  - Negociación: Coordinación entre agentes competitivos (hay conflicto de objetivos y debe llegarse a un acuerdo)

- Obtener coherencia global, maximizando la capacidad de trabajo en común de los agentes (eficiencia)
- Repartir tareas, recursos e información
- Reconocer y resolver disparidades y conflictos en objetivos, hechos, creencias, acciones
- Establecer la estructura organizativa de un grupo (reparto de roles)
- Evitar deadlocks (nadie puede actuar por un conflicto) y livelocks (las acciones no llevan a los objetivos)

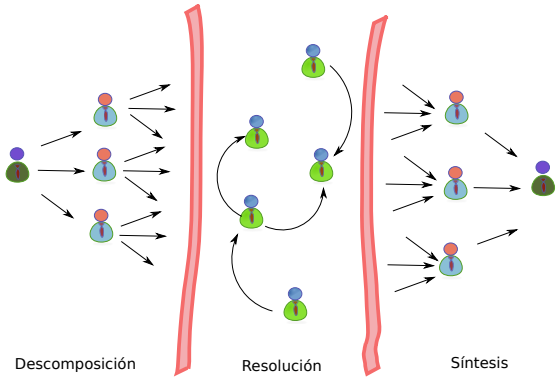
Organización

Cooperación

Negociación

Resolución de problemas cooperativa

- Fases en resolución cooperativa de problemas distribuidos
  1. Descomposición del problema
  2. Resolución de los subproblemas
  3. Síntesis de la solución



Descomposición del problema

- El problema debe dividirse en problemas más sencillos
- Habitualmente se hace de manera recursiva/jerárquica
- La división acaba cuando se obtienen problemas que puede resolver un solo agente

Descomposición del problema - Decisiones

- ¿Quién hace la división? ¿Está centralizada?
- ¿Cómo se hace la división?
- ¿Hay agentes que conocen la estructura global del problema?
- ¿Qué agentes resuelven los subproblemas?

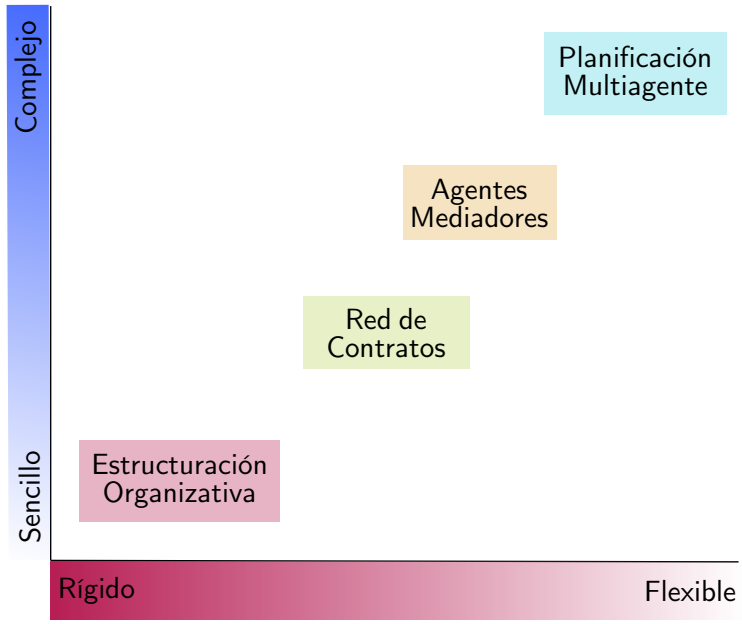
Resolución de subproblemas

- Cada agente resuelve la tarea que se le ha asignado
- Durante la resolución los agentes pueden compartir información entre ellos
- La resolución puede involucrar la sicronización entre agentes (si hay dependencias entre las tareas)

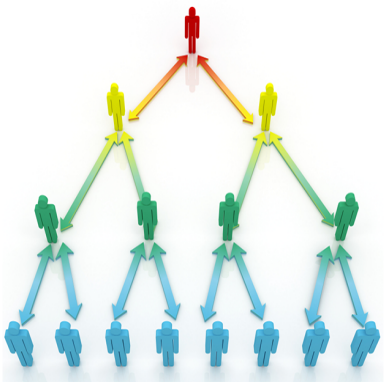
Síntesis de la solución

- Las soluciones de cada subproblema son informadas e integradas en la solución global
- Esta síntesis también será jerárquica

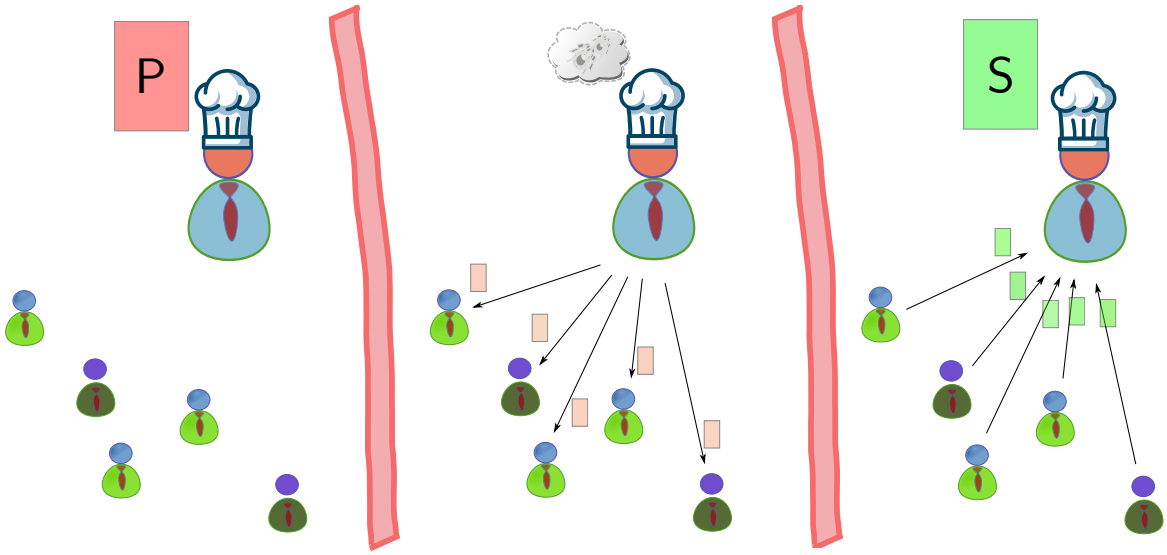
- Estructura organizativa: Dar una estructura a priori al conjunto de agentes (asignando roles específicos)
- Red de Contratos: Crear un mercado para distribuir las tareas
- Planificación multiagente: Crear planes para cumplir un objetivo y distribuir las tareas
- Agentes mediadores: Agentes intermediarios entre los que necesitan resolver una tarea y los que pueden resolverla



- El problema a resolver define una organización a priori
- Esta organización establece responsabilidades, capacidades, conectividad y flujo de control
- La asignación de estos elementos no tiene por que ser fija en el tiempo
- La más común asume una estructura jerárquica con un agente que tiene una visión global del problema

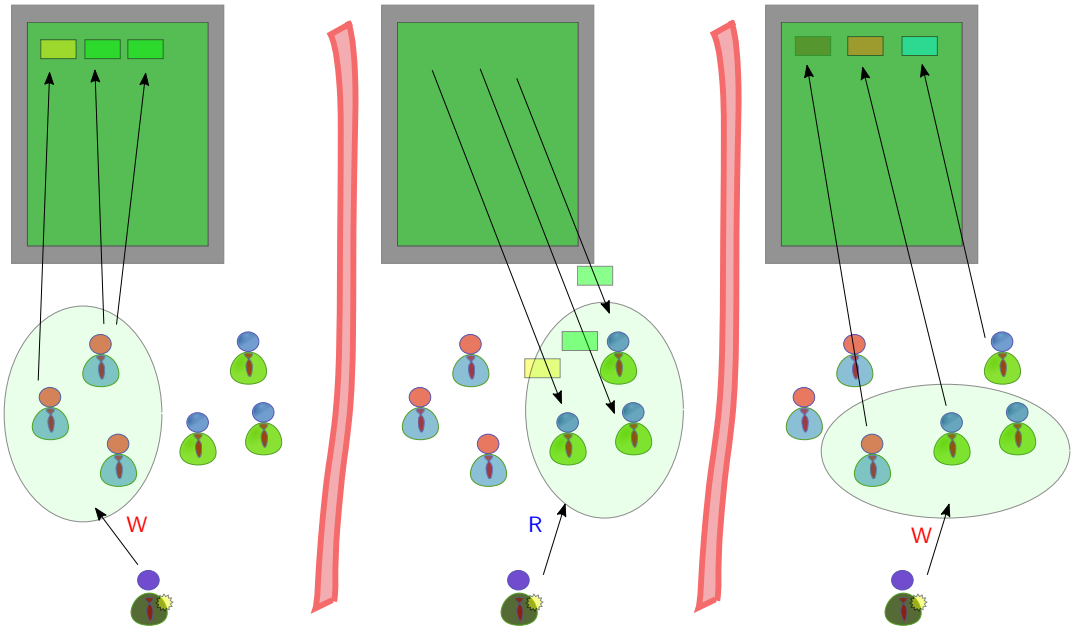


- El agente central planifica y distribuye las tareas entre agentes secundarios
- Los agentes secundarios pueden o no comunicarse
- Una vez finalizadas las tareas el agente central recibe e integra los resultados



- Un recurso común de lectura/escritura es usado como sistema de coordinación
- Un agente organiza las lecturas/escrituras según la estructura de la organización

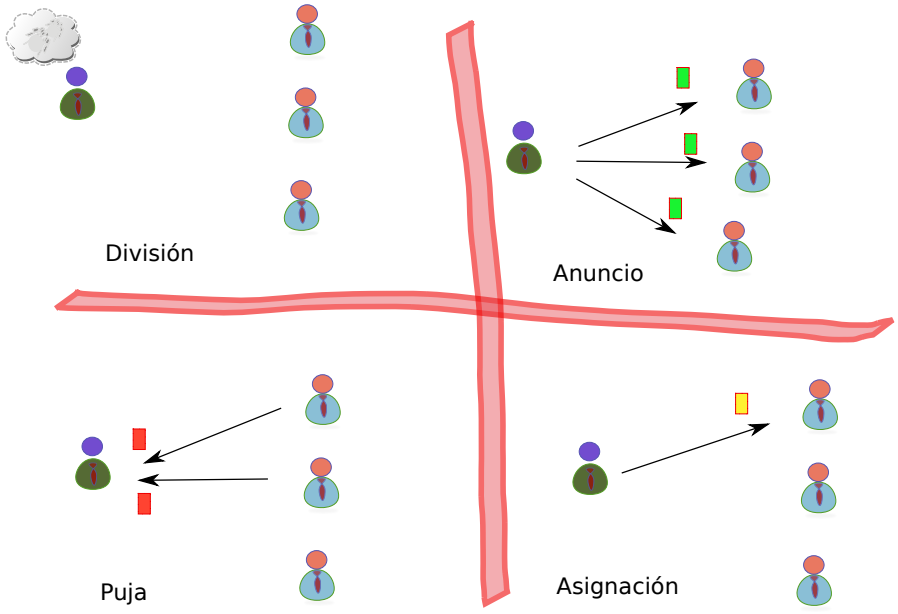
Estructura organizativa - Basadas en arquitecturas de pizarra



Coordinación - Red de Contratos

- Un agente coordinador (contratador) parte la tarea
- Las tareas son anunciadas a todos los agentes disponibles
- Los agentes evalúan la tarea respecto a sus capacidades/disponibilidad y responden con una oferta
- El agente contratador procesa las respuestas y asigna las tareas (contratos)
- Los agentes contratados pueden iterar el proceso
- El agente coordinador espera e integra las respuestas

Coordinación - Red de contratos



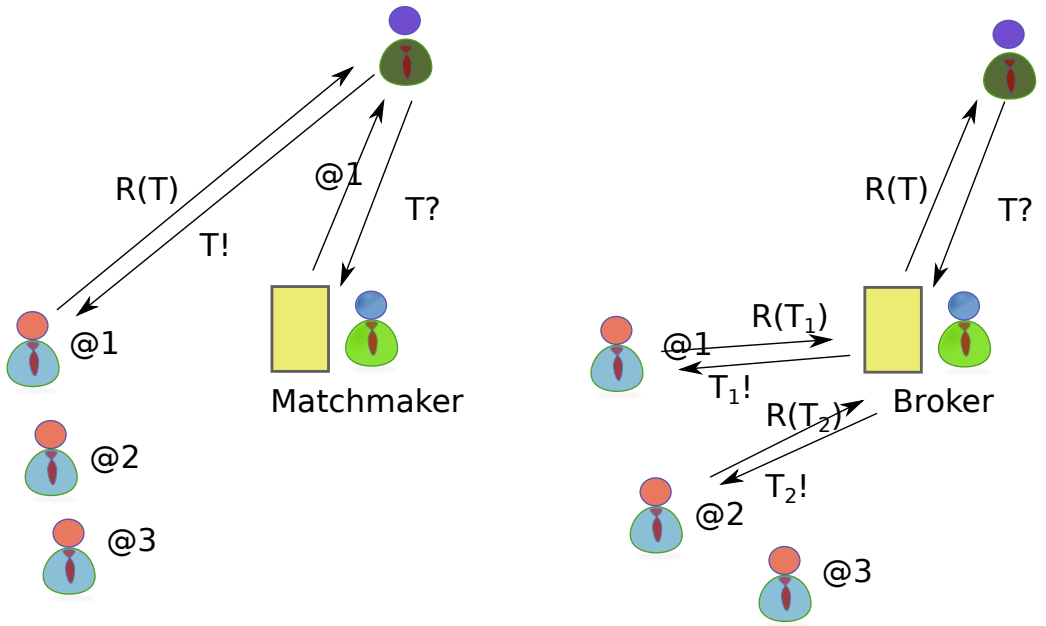
- ¿Cuándo?
  - Existe una estructura jerárquica en el problema
  - La tarea a resolver se puede descomponer en tareas más simples
  - No hay muchas interacciones entre tareas
- Problemas
  - Presupone que no habrá conflictos
  - Asume agentes pasivos, benévolos y no antagonistas
  - Requiere mucha comunicación

- Los agentes pueden planificar la tarea que les toca resolver (acciones a realizar)
- Los planes obtenidos se comunican para identificar conflictos
- Los conflictos se resuelven mediante replanificación
- Dos aproximaciones:
  - Planificación centralizada
  - Planificación distribuida
- Inconvenientes: Se ha de procesar y comunicar una gran cantidad de información

- Un agente coordinador recibe los planes del resto de agentes
- Este los analiza detectando inconsistencias y conflictos
- Se combinan los planes en un plan global y se redistribuyen a los agentes

- Cada agente tienen un modelo de los planes del resto
- Se comunican los conflictos y las actualizaciones de los planes

- **Matchmaking**:
  El agente mediador conoce las capacidades de los agentes, resuelve las peticiones y comunica a los solicitantes la dirección del agente más adecuado
- **Brokering:**
  El agente mediador conoce las capacidades de los agentes, pero es él el que reparte las tareas (uno o más agentes), recolecta los resultados y los reenvía al solicitante

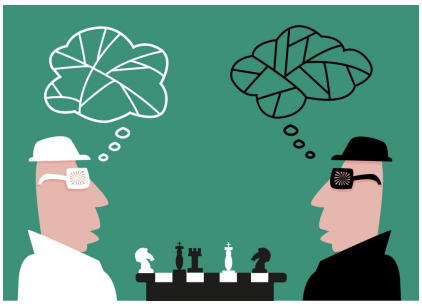


- **Entornos abiertos:** se pueden entrar en conflicto
  - Por el uso de recursos
  - Por el orden de realización de las tareas
- No existen estrategias definidas de negociación, solo protocolos de interacción
- Las estrategias a usar en las interacciones se resuelven
  - Mediante técnicas de teoría de juegos
  - Mediante heurísticas basadas en negociación entre personas

- **Teoría de juegos:** modeliza la toma de decisiones entre agentes racionales
- Un juego es cualquier interacción entre dos o más agentes con un objetivo
- Cada agente tiene una medida de satisfacción de sus acciones/resultados en el juego
- Cada agente tiene un conjunto de decisiones posibles que afectan al estado del juego
- La exploración del espacio de posibilidades puede dar lugar a estrategias de actuación

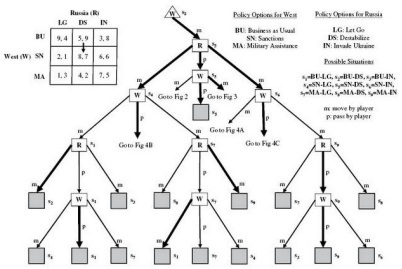
- **Espacio de acuerdos**, conjunto de combinaciones de acuerdos/acciones que pueden realizarse
- **Una función de utilidad**, cada agente modela el beneficio/perjuicio que obtiene con cada posible acción (propia y del contrario)
- **Estrategia**, secuencia de acciones que permite maximizar la utilidad
- **Protocolo**, forma en la que se comunican las ofertas/contraofertas

- Comportamiento racional por parte de los jugadores (la estrategia de un jugador afecta a la del resto)
- Todos los jugadores buscan maximizar su beneficio
- Todos los jugadores tienen las mismas capacidades



- La teoría de juegos permite analizar matemáticamente las decisiones posibles entre agentes
- Existen diferente tipos de juegos dependiendo de:
  - Número de jugadores
  - Información disponible
  - Efecto de las acciones (determinista/no determinista)
  - Resultado (un ganador, varios ganadores)
- Por ejemplo, los juegos de suma cero (un solo ganador) modelan situaciones totalmente competitivas

- El objetivo del análisis de un juego es encontrar una política de actuación estable para los jugadores
- Esta política permite maximizar la ganancia global de los jugadores
- Cualquier desviación de la política supone una pérdida



- **Arbitraje**: Los agentes revelan sus necesidades y un agente coordinador (árbitro) toma una decisión maximizando la utilidad global
- **Regateo**: Dos agentes se intercambian posibles acuerdos según sus preferencias hasta que las dos partes aceptan o rompen la negociación

- **Subastas**: Se puede cuantificar económicamente llegar o no a un acuerdo, un agente coordinador recibe las ofertas y el mejor postor es el que gana el juego (suma cero)
  - eg. Subasta inglesa (alza), holandesa (baja), sobre cerrado...
  - Aplicaciones: Mercados financieros, Google AdSense/AdWord, Facebook/Instagram

12

Composición de Servicios

Composición de servicios

Composición estática

Javier Béjar

ECSDI - 2023/2024 2Q

Objetivos del tema

- Programación como composición de servicios
- Descripción de servicios
- Estrategias de composición de servicios
  - Orquestación
  - Coreografía
- Lenguajes de composición
- Limitaciones

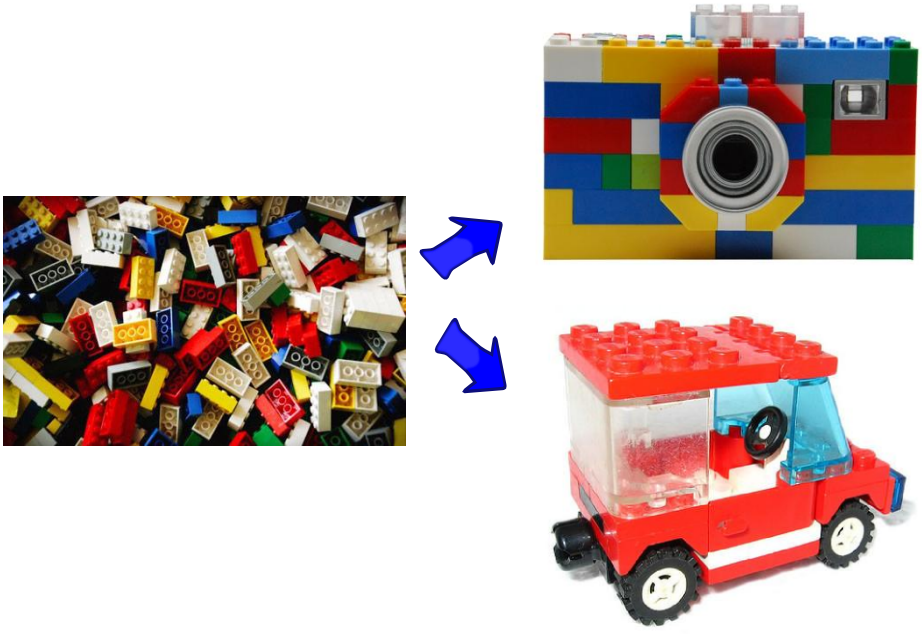
Servicios

- El número de servicios web disponibles en internet crece cada día
  - `www.programmableweb.com` tiene ∼20.000 WS APIs
- El desarrollo de aplicaciones en entornos abiertos permite aprovechar otros servicios accesibles
- Podemos desarrollar nuevas aplicaciones componiendo servicios
- El esfuerzo de desarrollo de las aplicaciones se ve reducido
- La filosofía de análisis y diseño de aplicaciones debe adaptarse

Programación con servicios

- El sentido de la programación basada en servicios (o agentes) es poder usarlos como componentes
- Cada servicio individual realiza una (o varias) tarea concreta
- La unión de servicios siguiendo diferentes flujos permite implementar diferentes soluciones
- **Composición:** unión de servicios (o agentes) para obtener una solución/aplicación

Programación con servicios



Descripción de servicios

- Diferentes tecnologías de composición de servicios
  - Ligadas a arquitecturas SOA (eg. SCA)
  - Como lenguajes de modelado para el diseño (SOAML)
  - Como lenguajes de documentacion del APIs web/Microservicios (Swagger/OpenAPI/API Blue-print)
- Están en diferentes procesos de estandarización y maduración
- Están apoyadas por diferentes compañías
- Están dirigidas a diferentes nichos de aplicación

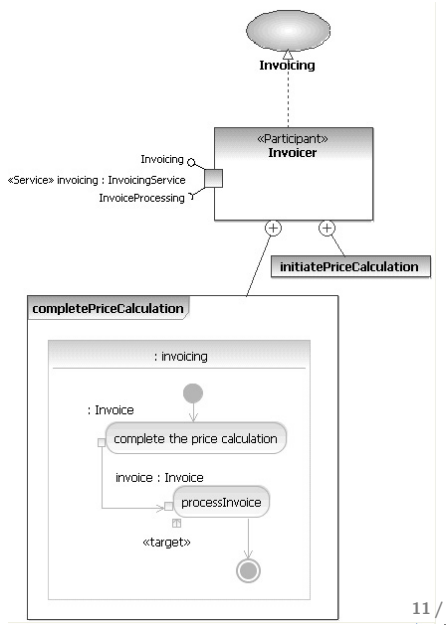
Service Component Architecture (SCA)

- La unidad es el componente
- Un componente
  - es accesible mediante una interfaz
  - puede referenciar a otros componentes
  - tiene elementos configurables que modifican su comportamiento
  - puede estar formado a su vez por otros componentes
- El flujo del proceso a componer dicta como se han de conectar

- Desacopla la composición de los detalles de invocación
- Es agnóstico al lenguaje/tecnología (C++, java, XML, BPEL)
- Puede utilizar diferentes mecanismos de comunicación
- Puede integrar elementos basados en otras tecnologías (RMI, RPC, CORBA...)
- Varias implementaciones Open Source y propietarias (IBM, Oracle, Red Hat)
- Estándar OASIS (open SCA)

SOAML

- Service Oriented Architecture Modelling Language (SOAML) es un perfil UML y un metamodelo para la definición de arquitecturas orientadas a servicios
- Extiende UML para poder definir todos los elementos que necesita una arquitectura orientada a servicios



SOAML

- Como identificar servicios (qué proveen, dependencias)
- Como especificar servicios (capacidades funcionales, protocolos, intercambio de información)
- Definición de proveedores y consumidores
- Políticas de uso y provisión de servicios
- Requerimientos de los servicios y su uso
- Esquemas de clasificación de servicios/organización/restricciones
- Enlace con otros metamodelos UML

- El elemento principal es el participante
- Los participantes pueden ser proveedores o consumidores
- Los servicios se definen a través de puertos
- Estos definen puntos de provisión y consumo de servicios
- El acceso a los servicios se define a partir de:
  - Una interfaz UML (unidireccional)
  - Una interfaz de servicio (bidireccional)
  - Un contrato de servicio (bidireccional)
- Fases iniciales de implantación (primera versión 2012)

- Estas tecnologías permiten definir cómo son los componentes y cómo se organizarán
- Permiten definir el proceso de negocio especificando:
  - El orden potencial de ejecución de la composición
  - Los datos compartidos por los servicios
  - Qué servicios estarán involucrados y cómo (interacciones/relación)
  - Cómo se tratarán las excepciones (quién/cómo)

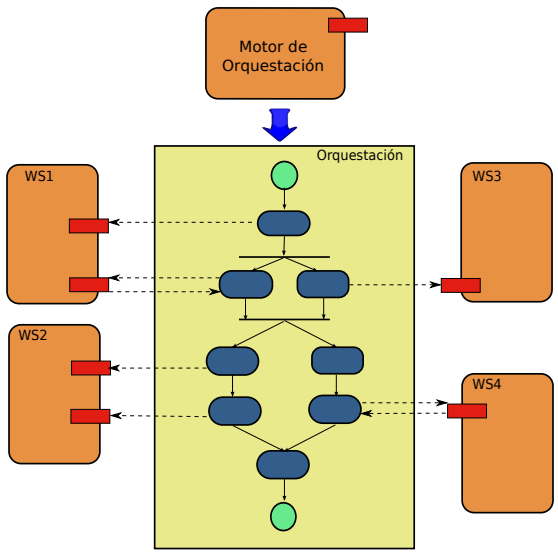
Estrategias de composición

- Existen diferentes posibilidades de establecer como la composición se implementará al ejecutarla
  - Para el flujo de ejecución:
    - Prefijado (estático)
    - Generado automáticamente a partir de la tarea a resolver (dinámico)
  - Para los servicios involucrados
    - Servicios prefijados (estáticos)
    - Servicios provistos por un servicio de directorio (coincidencia sintáctica/semántica) (dinámicos)

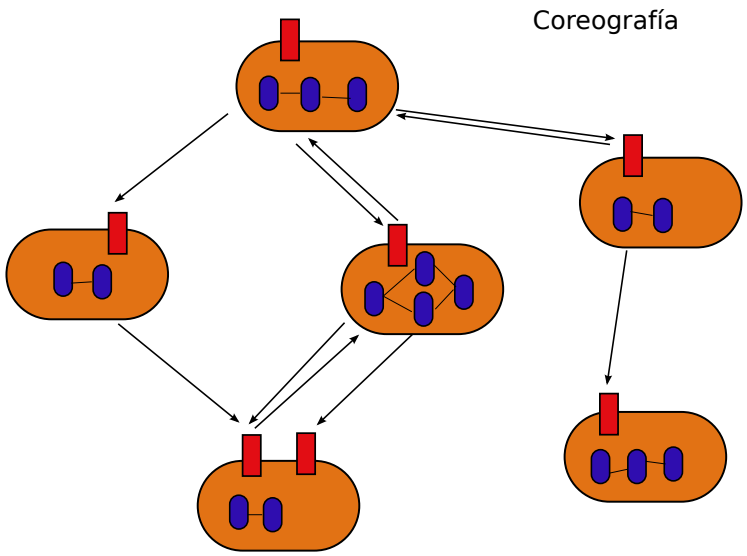
Estrategias de composición - Orquestación

- Supone que un ente organizador ejecutará la composición (motor de orquestación)
- Define el flujo desde la perspectiva de un único integrante (orquestador)
- Una orquestación define:
  - La lógica de ejecución
  - El orden de las interacciones
  - Cómo los componentes interactuarán al nivel de mensaje

Estrategias de composición - Orquestación



- El control está distribuido entre los participantes
- Define el flujo desde la perspectiva de todas las partes
- Cada participante conoce el comportamiento de los otros (con los que colabora)
- Hay una visión común de los elementos del estado
- La coreografía describe la secuencia de mensajes que intercambian múltiples participantes
- Cada participante describe el rol que tiene en la interacción

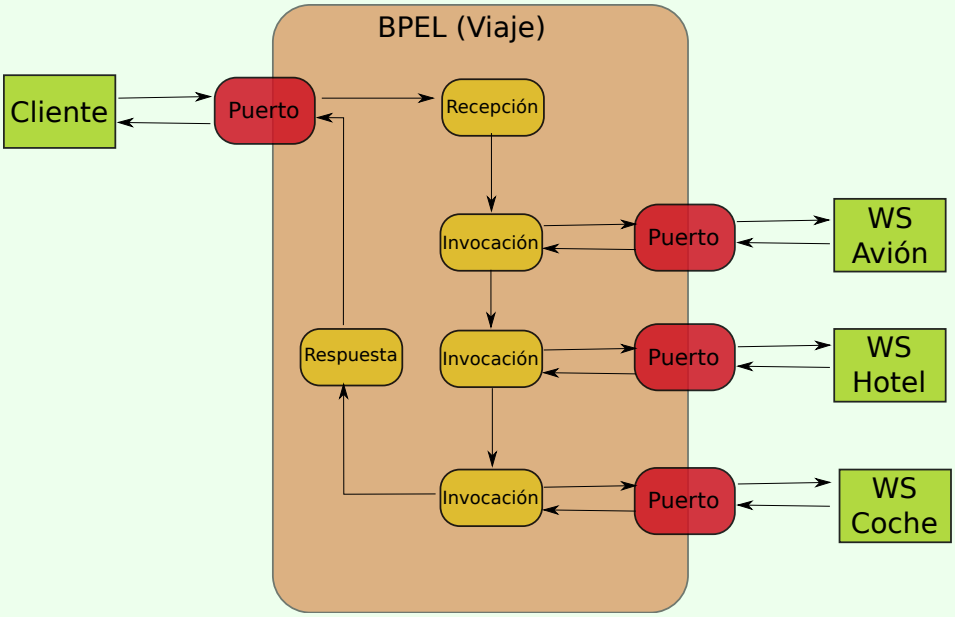


- La implementación puede ser programática
- Es más ventajoso utilizar **lenguajes de composición**:
  - Independencia del lenguaje de los servicios
  - Adaptación al cambio en el flujo de ejecución del servicio compuesto
- Lenguajes utilizados:
  - WS-BPEL (Web Services Business Process Execution Language) (OASIS)
  - WS-CDL (Web Services Choreography Description Language) (W3C)

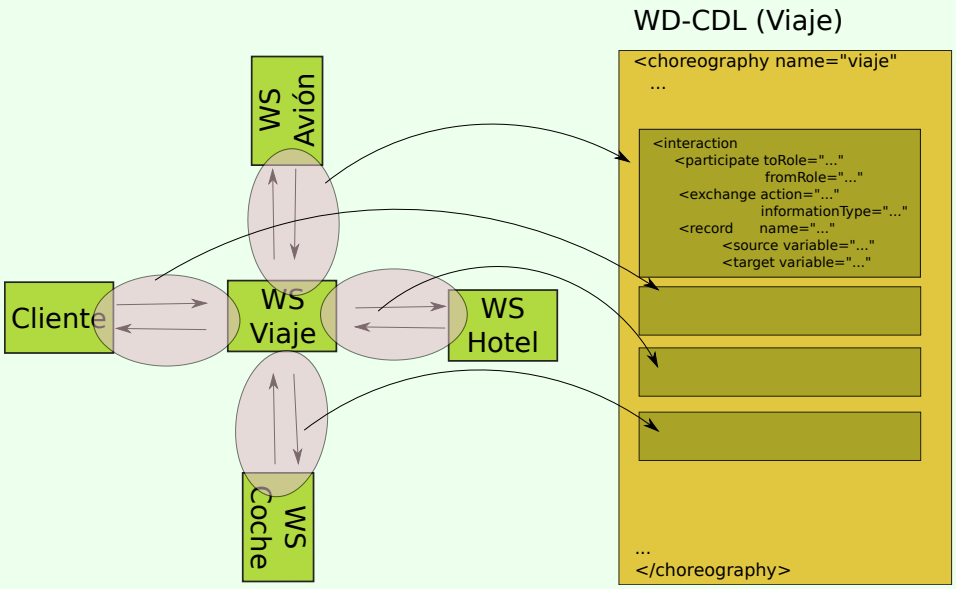
- Descripción mediante WSDL (Web Services Description Language)
- Permite definir:
  - **Procesos ejecutables**: Especificación de los detalles del flujo de ejecución
  - **Procesos abstractos**: Procesos generales que permiten dejar elementos por especificar (a instanciar)
- WS-BPEL define un lenguaje de programación de procesos: invocación, recepción, envío de mensajes, excepciones, secuencias, flujos paralelos, bucles, alternativas...

- Definición del acceso al servicio compuesto desde el exterior mediante WSDL
  - Definición de puertos, operaciones, mensajes...
- El proceso es ejecutado por el motor de orquestación según su definición en WS-BPEL
- Este proceso define
  - cómo y cuándo se ejecutarán los servicios
  - cómo se guardará el estado
  - excepciones, fallos...

1. Una petición inicia el proceso
2. Invocación al servicio de reserva de vuelos (envió/recepción mensaje)
3. Guardar respuesta vuelo
4. Invocación al servicio de reserva de alojamiento (envió/recepción mensaje)
5. Guardar respuesta alojamiento
6. Invocación al servicio de reserva de coche (envió/recepción mensaje)
7. Guardar respuesta coche
8. Generar y enviar mensaje respuesta a la petición



- Se basa en la descripción de los servicios mediante WSDL
- No permite definir procesos ejecutables, es un lenguaje de definición
- Permite definir las interacciones entre los diferentes participantes
  - Cada interacción puede implementarse a partir de mecanismos diferentes (por ejemplo WS-BPEL)
- Permite la interoperabilidad entre servicios al ser independiente de la implementación de las interacciones



- Tienden a obtener composiciones estáticas
- Son sensibles al fallo de los servicios (el fallo de un solo servicio acaba el proceso)
- La complejidad de los procesos los hacen difíciles de diseñar y depurar
- Los servicios son modelados al mismo nivel de abstracción y granularidad independientemente de su complejidad
- Suele haber poco conocimiento del contexto de las interacciones (normas/regulaciones)

- El manejo del flujo de ejecución en entornos abiertos no es trivial
  - No todos los servicios pertenecen a la misma organización
  - No podemos asumir que todas las partes cumplirán su cometido correctamente
  - ¿Debería ser necesario un acuerdo previo entre las partes?
- ¿Tratamiento de servicios esenciales de terceras partes que dejan de funcionar?
- ¿Tratamiento de la evolución de aplicaciones y procesos?

- Los objetivos y procesos de las aplicaciones evolucionan con el tiempo
  - **Diseño**: No podemos prever todas las interacciones
  - **Implantación**: Asumimos una composición dinámica
  - **Gestión**: Cambio/adaptación/evolución deberían ser un elemento natural
- Las tecnologías a usar deberían
  - Permitir adaptarse a un entorno cambiante
  - Permitir la reconfiguración sencilla de los procesos
  - Permitir la adaptación dinámica a las necesidades

13

Composición Dinámica de Servicios

Composición Dinámica

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Composición dinámica

- Orquestación y coreografía asumen un conjunto prefijado de servicios
- El desarrollar aplicaciones en entornos abiertos permite que esto no tenga que ser así
- En el momento de la ejecución podemos decidir qué instancias de servicio serán utilizadas
- Se necesitan elementos intermediarios que gestionen la elección de servicios
- Se necesita hacer una descripción de entradas, salidas y efectos de los servicios

- Descubrimiento automático de servicios
  - La decisión del servicio específico a usar se realiza en tiempo de ejecución (búsqueda)
- Invocación automática de servicios
  - La invocación en tiempo de ejecución se obtiene a partir de la descripción declarativa del servicio

- Composición automática e interoperación
  - El flujo de ejecución se obtiene por la selección de los servicios adecuados y la generación de su composición
  - La conexión entre los servicios (interoperabilidad) se obtiene a partir de sus descripciones (entradas/salidas)
- Monitorización automática
  - La detección de excepciones/fallos se determina a partir de sus descripciones (objetivos/precondiciones/estado)
  - El tratamiento de las excepciones/fallos (recuperación) se realiza automáticamente

Descubrimiento de servicios

- Previo o durante el proceso de ejecución de un flujo de negocio se eligen los servicios a usar
- Aparece la figura del **matchmaker**
  - Recibe características que los servicios deben cumplir
  - Busca servicios que cumplan esas características
  - Elige entre los servicios disponibles
  - Esa elección puede atender a características de calidad de servicio (QoS)

Servicios de directorio

- Otro elemento fundamental del descubrimiento es el servicio de directorio (Páginas Amarillas)
- Este servicio permite el registro de la descripción de los proveedores de servicios
- Cada servicio indica sus características de manera que puedan ser encajadas con necesidades de otros servicios
- De la complejidad de esta descripción depende la flexibilidad de la composición

Composición/Descubrimiento - Posibilidades

- Flujo de ejecución fijo y descubrimiento sintáctico
  - Resolución siempre igual
  - Sólo entradas y salidas como parámetros, coincidencia sintáctica
- Flujo de ejecución fijo y descubrimiento semántico
  - Resolución siempre igual
  - Entradas/salidas descritas a partir de una ontología, coincidencia semántica
- Flujo de ejecución dinámico y descubrimiento semántico
  - Diferentes alternativas
  - Entradas/salidas/precondiciones/efectos descritos a partir de una ontología

- **UDDI** (Universal Description Discovery and Integration)
- Estándar para la publicación y descubrimiento de servicios
  - Clasificación, catálogo y manejo de servicios web
  - Búsqueda de servicios a partir de criterios
  - Parámetros de invocación, protocolos de transporte y seguridad
  - Tratamiento de errores y cambios en los servicios
- Tres componentes:
  - Páginas blancas: dirección, contacto e identificadores
  - Páginas amarillas: categorización de los servicios a partir de una taxonomía estándar
  - Páginas verdes: información técnica del servicio

- Diferentes necesidades de Micro Servicios y Cloud Computing han dado lugar a sistemas de registro con diferentes capacidades
  - Netflix Eureka (sobre servicios en AWS)
  - Apache Zookeeper (sobre contenedores)
  - Consul (descubrimiento/configuracion) (BD clave/valor)
  - Etcd (descubrimiento/configuracion) (BD clave/valor)
  - SkyDNS, SmartStack, Serf...

- La **búsqueda sintáctica** depende de:
  - Una buena clasificación de los servicios (detallada, uso de estándares)
  - Una descripción detallada y completa de los parámetros
- Limitaciones
  - La coincidencia de los parámetros ha de ser exacta (literal)
  - La coincidencia de los parámetros no asegura la semántica del servicio (efectos, precondiciones)

Métodos semánticos

- Una descripción más precisa de los servicios lleva a una mayor efectividad en la búsqueda
- Se ha de considerar que un proceso ejecutado por un conjunto de servicios tiene un estado
- Cada servicio modifica el estado para conseguir objetivos
- Cada objetivo tiene una serie de precondiciones
- La ejecución de las operaciones modifican el estado y generan como objetivos precondiciones de otros servicios

Descubrimiento Semántico

- El descubrimiento semántico de servicios pretende ir más allá de la coincidencia sintáctica
- La descripción de servicio usando conceptos de una ontología para entradas, salidas, condiciones y efectos permite el uso de deducción automática
- También se pueden utilizar las relaciones clase/subclase y de equivalencia para la coincidencia
- La labor de deducción la debe realizar el matchmaker
  - Para cada entrada, salida, condición y efecto en la consulta debe haber una coincidencia en el servicio

Descripción de servicios

- Poder hacer una búsqueda semántica de servicios depende de la expresividad del lenguaje de descripción
- Se han desarrollado diferentes alternativas para la descripción semántica de servicios:
  - Semantic Annotations for WSDL (SAWSDL) `https://www.w3.org/TR/sawsdl/`
  - OWL-S (OWL-Services) `https://www.w3.org/Submission/OWL-S/`
  - WSMO (WS Modelling Ontology) + WSML (WS Modelling Language)

SAWSDL

- Permite complementar la búsqueda por coincidencia literal entre las entradas y salidas de los servicios
- Las descripciones WSDL incluyen anotaciones que referencian ontologías
- La ontología indica el significado del parámetro
- Se puede razonar sobre los significados de los parámetros
  - Para calcular la coincidencia entre entradas y salidas
  - Para obtener una transformación que permita la interoperabilidad

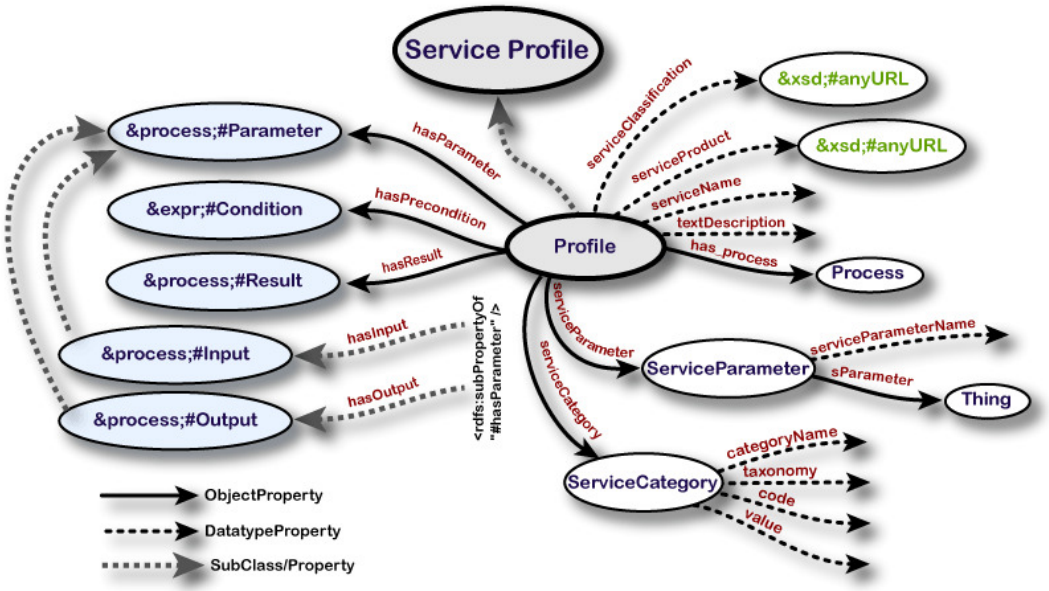
OWL-Services

- Ontología de servicios desarrollada sobre OWL
- Un servicio se describe a partir de tres elementos:
  - **Perfil del servicio** (Service Profile): Qué requiere de sus usuarios y qué provee
  - **Modelo del servicio** (Service Model): Cómo funciona el servicio
  - **Uso del servicio** (Service Grounding): Cómo se usa el servicio

OWL-S - Service Profile

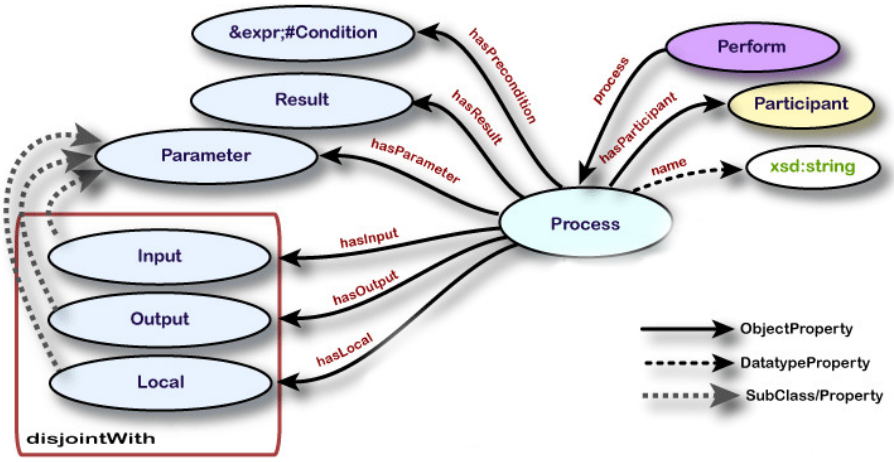
- Permite representar la información que el matchmaker necesita para encontrar servicios
- Se compone de tres elementos:
  - Qué organización provee el servicio (info de contacto)
  - Qué función es computada por el servicio (entradas, salidas, precondiciones y efectos)
  - Características del servicio (clasificación, calidad)
- La descripción del perfil debería ser consistente con la descripción del modelo de servicio

OWL-S - Service Profile

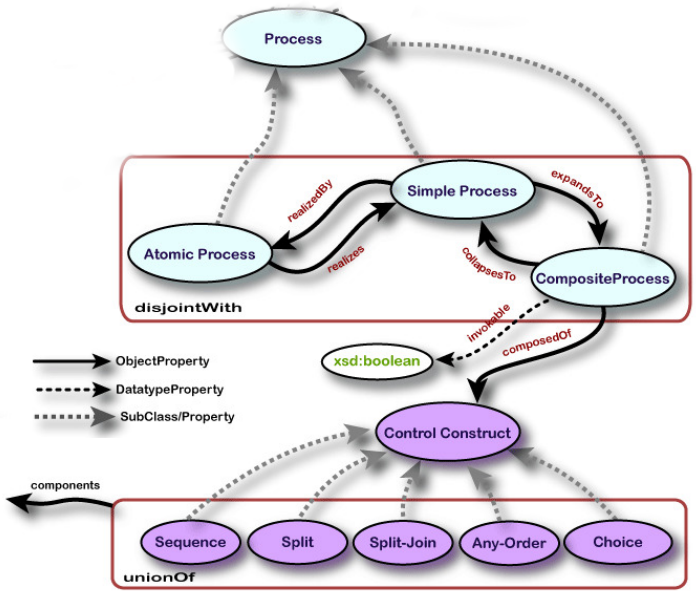


OWL-S - Service Model

- OWL-S describe el modelo del servicio como un proceso
- Ontología de procesos para describir su funcionamiento



OWL-S - Service Model



- **Entradas y salidas**:
  - Provistas por/enviadas a servicios externos u otros procesos
- **Precondiciones y efectos**
  - Las precondiciones son condiciones que se deben cumplir en el estado para que el proceso se pueda ejecutar
  - Los efectos son las condiciones que cambian en el estado
- **Condiciones sobre los efectos y salidas**
  - Efectos y salidas dependen de la invocación específica (pueden suceder según las circunstancias)

- La descripción e implementación del servicio ha de conectarse transformando los elementos descritos en el proceso a invocaciones
  - Correspondencia entre las entradas y salidas de los procesos atómicos (abstracción del proceso) y la implementación
  - Identificación de protocolos, formatos de mensajes, serialización, transporte y direcciones
- En este proceso los procesos atómicos se vinculan con la descripción WSDL de los servicios

Composición Dinámica

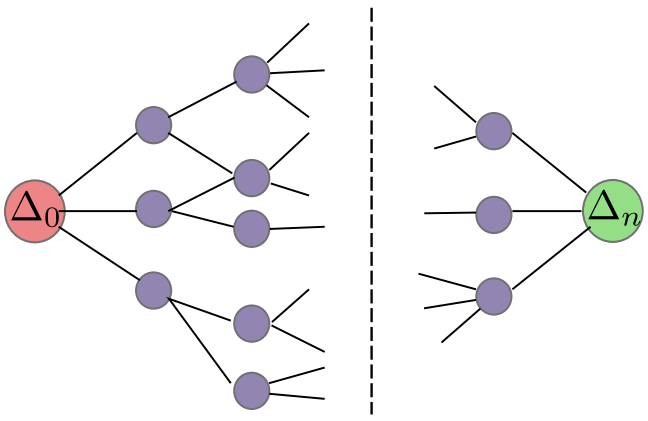
- Hasta ahora hemos supuesto que el flujo de ejecución de los servicios ya existía
- Esto limita la composición a solamente elegir los servicios específicos a usar en cada paso
- Incluir en la descripción del servicio precondiciones y efectos permite generar automáticamente el flujo de ejecución
- Permite usar más flexiblemente los servicios descubiertos
- Es necesario un sistema capaz de planificar la composición

Planificación automática

- La planificación automática es una técnica de resolución de problemas de Inteligencia Artificial
- Es un método de programación automática
- Un problema se describe a partir de:
  - La representación del objetivo a alcanzar
  - La representación de las acciones que se pueden realizar
  - La representación de los elementos estado
- Un planificador determina la secuencia de acciones que obtiene el objetivo a partir de las acciones
- En nuestro caso las acciones son los servicios/agentes

Planificación automática

- La planificación se plantea como búsqueda de caminos
- Obtener un plan significa encontrar un camino entre el estado inicial y los objetivos finales



- Un problema de planificación se define a partir de
  - Un conjunto de acciones aplicables fijo

$$Ac = \{\alpha_1, ..., \alpha_n\}$$

  - Un estado inicial Δ que define las condiciones iniciales del problema
  - Un objetivo Ω define las características (totales o parciales) que debe cumplir el estado solución

- $< P_\alpha, D_\alpha, A_\alpha >$ es un descriptor para una acción $\alpha \in Ac$
  - P_α es un conjunto de formulas lógicas que caracterizan la precondición de la acción α
  - D_α es un conjunto de fórmulas lógicas que caracterizan aquellos hechos que se vuelven falsos por la ejecución de α (*delete list*)
  - A_α es un conjunto de fórmulas lógicas que caracterizan aquellos hechos que se vuelven ciertos por la ejecución de α (*add list*)

- Dada una tripleta $< \Delta, Ac, \Omega >$, un plan $\pi = \{\alpha_1, \ldots, \alpha_n\}$ determina una secuencia de estados $\Delta_0, \ldots, \Delta_n$
- Donde $\Delta_0 = \Delta$ y

$$\Delta_i = (\Delta_{i-1} \ - D_{\alpha_i}) \cup A_{\alpha_i} \ \ para \ \ 1 \leq i \leq n$$

- Un plan π es aceptable ssi $\Delta_{i-1} \vdash P_{\alpha_i}$ para $1 \leq i \leq n$
- Un plan π es correcto ssi es aceptable y $\Delta_n \vdash \Omega$

- Cada elemento de un problema de planificación (estados, acciones, precondiciones) se representa a partir de fórmulas lógicas
- Dependiendo de la expresividad de la lógica empleada se pueden representar diferentes complejidades de problemas
- Existe un lenguaje estandarizado para los sistemas de planificación automática (PDDL)

- **Representación de estados**: Los planificadores describen el dominio a partir de fórmulas lógicas, representando un **estado** como una conjunción de literales positivos:
  - Proposiciones:

$$Pobre \wedge Desconocido$$

  - Literales de 1er orden:

$$En(Avion1, Melbourne) \wedge En(Avion2, Sydney)$$

- **Representación de objetivos**: Un objetivo es un estado parcialmente especificado
- Un estado S satisface un objetivo O si S contiene todos los átomos de O (y posiblemente algunos más)

$$O \subseteq S \equiv S \vdash O$$

El estado:

$Rico \wedge Famoso \wedge Miserable$

satisface el objetivo (se deduce)

$Rico \wedge Famoso$

- **Representación de acciones**: Las acciones se especifican en términos de las precondiciones que se han de cumplir antes de que se puedan ejecutar y de los efectos que producen una vez se han ejecutado

`volar(av, orig, dest)`	
PRECOND:	En(av, orig) $\wedge$ Avion(av) $\wedge$ Aeropuerto(orig) $\wedge$ Aeropuerto(dest)
EFECTO:	$\neg$ En(av, orig) $\wedge$ En(av, dest)

- La precondición es una conjunción de literales que especifica qué debe de ser verdadero en un estado antes de que la acción se ejecute
- El efecto es una conjunción de literales describiendo como cambia el estado cuando la acción se ejecuta
  - Qué pasa a ser cierto en el nuevo estado
  - Qué deja de ser cierto en el nuevo estado

- Una acción es aplicable en cualquier estado que satisfaga la precondición
- Si es necesario, se unifican sus variables en la precondición

El estado

$$\underline{Avion(A1)} \wedge \underline{En(A1, JFK)} \wedge \\ \underline{Avion(A2)} \wedge \underline{En(A2, SFO)} \wedge \\ \underline{Aeropuerto(JFK)} \wedge \underline{Aeropuerto(SFO)}$$

satisface la precondición de la acción $volar(av, orig, dest)$

$$av = A1 \;\; orig = JFK \;\; dest = SFO$$

- El resultado de ejecutar la acción en un estado S es un estado S' al que:
  - se añaden los literales positivos del efecto
  - se eliminan los literales negativos

El efecto de la acción volar sobre el estado anterior:

$$En(A1, SFO) \wedge Avion(A1) \wedge$$
$$En(A2, SFO) \wedge Avion(A2) \wedge$$
$$Aeropuerto(JFK) \wedge Aeropuerto(SFO)$$

Se eliminó: $En(A1, JFK)$
Se añadió: $En(A1, SFO)$

- Dos cargas (C1 y C2) estan en 2 aeropuertos (SFO, JFK)
- Tenemos dos aviones (A1 y A2) para transportar las cargas, uno en cada aeropuerto
- Describimos el estado inicial así:

$$Inicio(En(C1, SFO) \wedge En(C2, JFK) \wedge En(A1, SFO) \wedge En(A2, JFK) \wedge Carga(C1) \wedge Carga(C2) \wedge Avion(A1) \wedge Avion(A2) \wedge Aeropuerto(SFO) \wedge Aeropuerto(JFK))$$

- El objetivo es que C1 acabe en JFK y C2 en SFO
- Describimos el objetivo así:

$$Objetivo(En(C1, JFK) \wedge En(C2, SFO))$$

`carga(c, av, aerop)`	
PRECOND:	En(c, aerop) ∧ En(av, aerop) ∧ Carga(c) ∧ Avion(av) ∧ Aeropuerto(aerop)
EFECTO:	¬En(c, aerop) ∧ Dentro(c, av)

`descarga(c, av, aerop)`	
PRECOND:	Dentro(c, av) ∧ En(av, aerop) ∧ Carga(c) ∧ Avion(av) ∧ Aeropuerto(aerop)
EFECTO:	En(c, aerop) ∧ ¬Dentro(c, av)

`volar(av, orig, dest)`	
PRECOND:	En(av, orig) ∧ Avion(av) ∧ Aeropuerto(orig) ∧ Aeropuerto(dest)
EFECTO:	¬En(av, orig) ∧ En(av, dest)

Solucion 1: dos aviones para hacer el traslado

$[carga(C1, A1, SFO), vuela(A1, SFO, JFK),$
$descarga(C1, A1, JFK), carga(C2, A2, JFK),$
$vuela(A2, JFK, SFO), descarga(C2, A2, SFO)]$

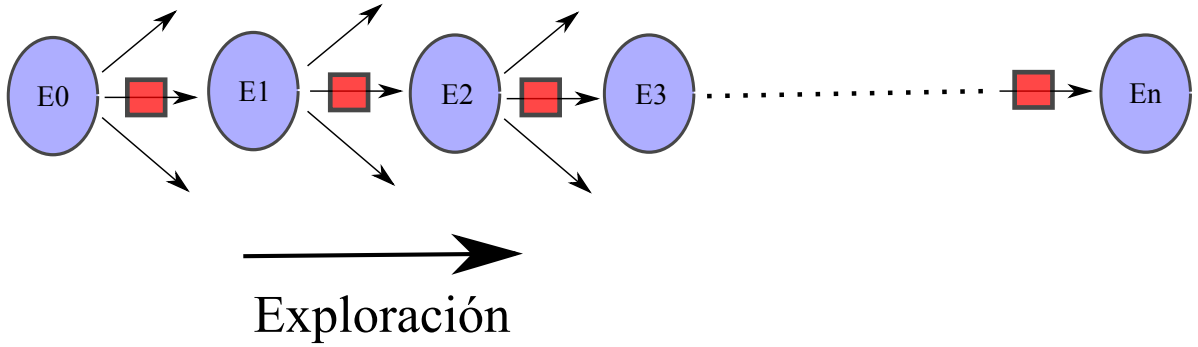
Solucion 2: usamos solo un avión

$[carga(C1, A1, SFO), vuela(A1, SFO, JFK),$
$descarga(C1, A1, JFK), carga(C2, A1, JFK),$
$vuela(A1, JFK, SFO), descarga(C2, A1, SFO)]$

- Existen tres estrategias para la resolución de un problema de planificación (Calcular la secuencia de acciones)
  - Planificación en espacio de estados (State-Space Planning)
  - Planificación en el espacio de planes (Plan-Space Planning)
  - Planificación jerárquica (Hierarchical Task Network Planning)
- Cada una de ellas se puede plantear mediante diferentes algoritmos y heurísticos

Espacio de estados (State-Space Planning)

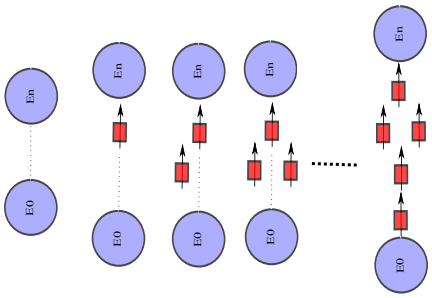
- El problema se define como una secuencia de acciones que va completando los objetivos del problema
- Se explora el espacio de caminos que conecta el estado inicial y final



Exploración

Espacio de planes (Plan-Space Planning)

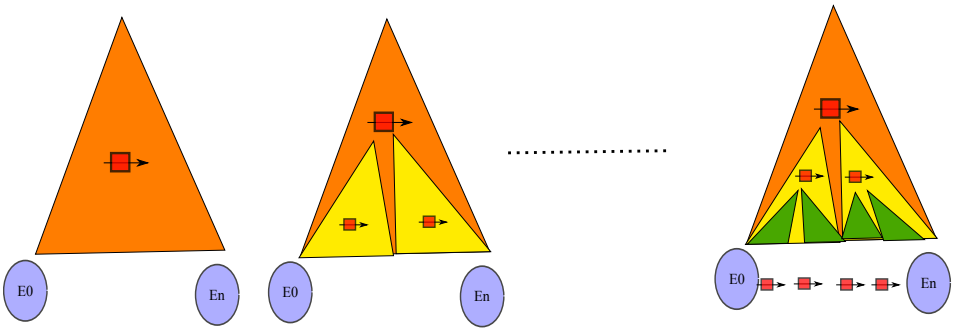
- El problema se define como una secuencia incompleta de acciones
- Se explora el espacio de planes parciales añadiendo acciones que completan el plan

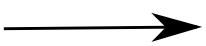


Exploración

Espacio de descomposiciones (Hierarchical Task Network Planning)

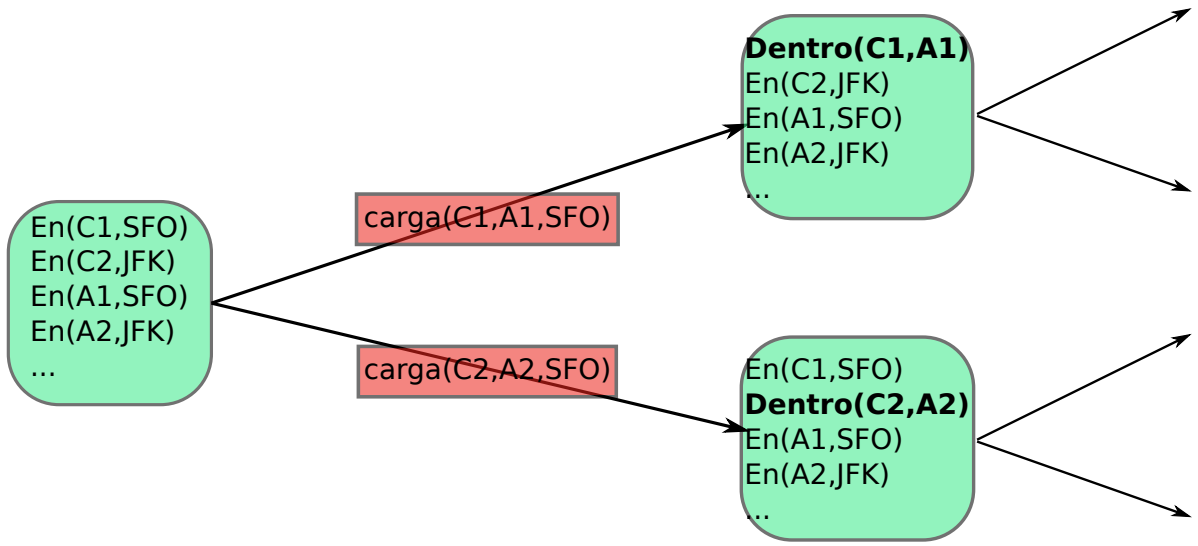
- El problema se plantea como una descomposición jerárquica de problemas
- Se explora el espacio de descomposiciones hasta obtener una secuencia de problemas primitivos





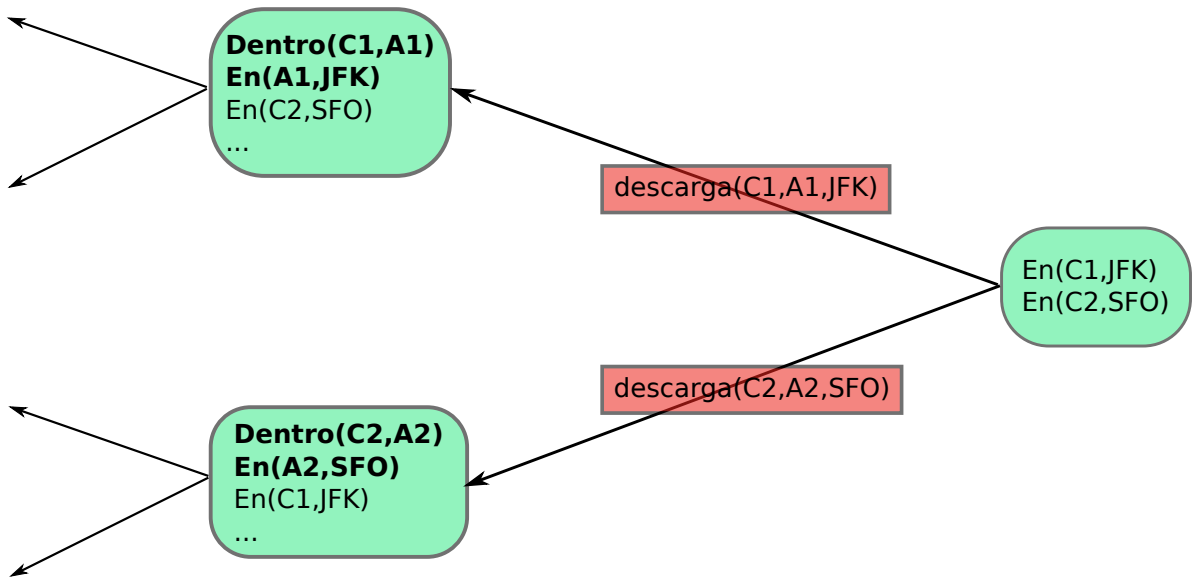
Exploración

- Se parte del estado inicial
- Mientras el estado no es el objetivo:
  1. Se intentan unificar las precondiciones de las acciones
  2. Se elige una acción que unifique
  3. Se aplican los efectos de la acción al estado
     - nuevos predicados son ciertos
     - algunos predicados dejan de ser ciertos



- Algoritmos de búsqueda hacia adelante:
  - Anchura prioritaria
  - Profundidad prioritaria
  - Best first (A*, IDA*)
  - Greedy Best first
- Heurísticas generales para reducir el espacio de búsqueda
- Problema: Búsqueda no dirigida por el objetivo lleva a una explosión combinatoria, se puede reducir mediante heurísticas dependientes del dominio

- La búsqueda se inicia desde el estado final del problema
- Mientras haya objetivos pendientes:
  1. Se unifican los efectos de las acciones
  2. Se elige una acción que cubra subobjetivos
  3. Se eliminan subobjetivos que son efectos positivos de la acción
  4. Se añaden al estado sus precondiciones (nuevos subobjetivos)

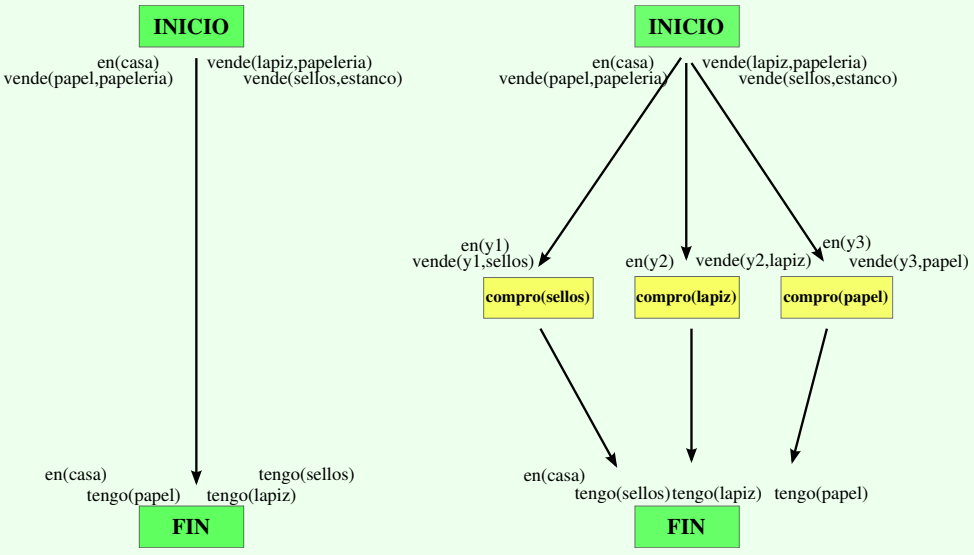


- Permite focalizar mejor la búsqueda (solo precondiciones que llevan al objetivo del problema)
- El espacio de búsqueda es aún bastante grande
- Son necesarias heurísticas generales y específicas para reducir el espacio de búsqueda

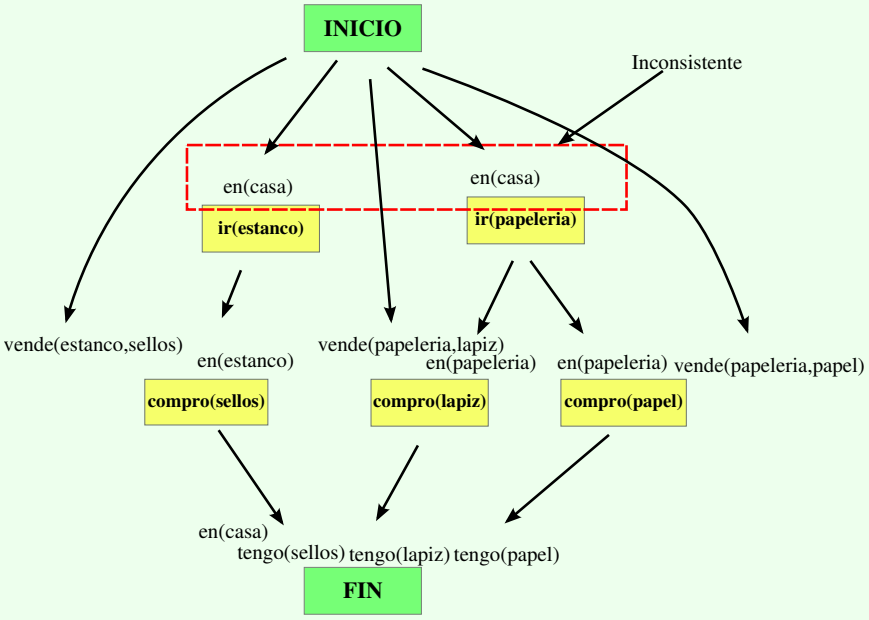
- Búsqueda **hacia atrás** desde el objetivo
- Cada nodo del espacio de búsqueda es un plan parcial que incluye
  - un conjunto de operadores parcialmente instanciados
  - un conjunto de restricciones sobre los operadores

- Descomponer conjuntos de objetivos en objetivos individuales
- Planificar cada subobjetivo por separado
- Detectar y corregir conflictos entre objetivos imponiendo restricciones
- Aplicar estrategia de mínimo compromiso (mínima modificación que hace válido el plan)
- Se acaba cuando se obtiene un plan parcialmente ordenado

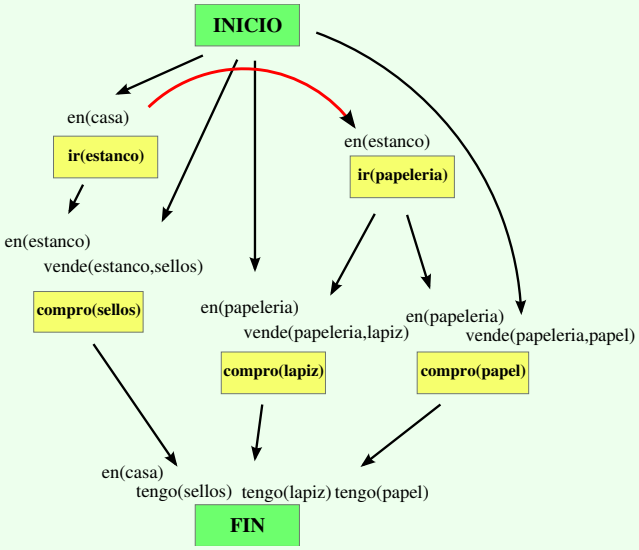
- **Problema:** escribir una carta
- Condiciones iniciales: `en(casa)`, `vende(papel, papelería)`, `vende(lápiz, papelería)`, `vende(sellos,estanco)`
- Condiciones finales: `en(casa)` `tengo(papel)`, `tengo(sellos)`, `tengo(lápiz)`
- Operadores:
  - `ir(x)` $\rightarrow$ prec: `[en(y)]`, efec: `[en(x),¬en(y)]`
  - `comprar(x)` $\rightarrow$ prec: `[en(y), vende(y,x)]`, efec: `[tengo(x)]`



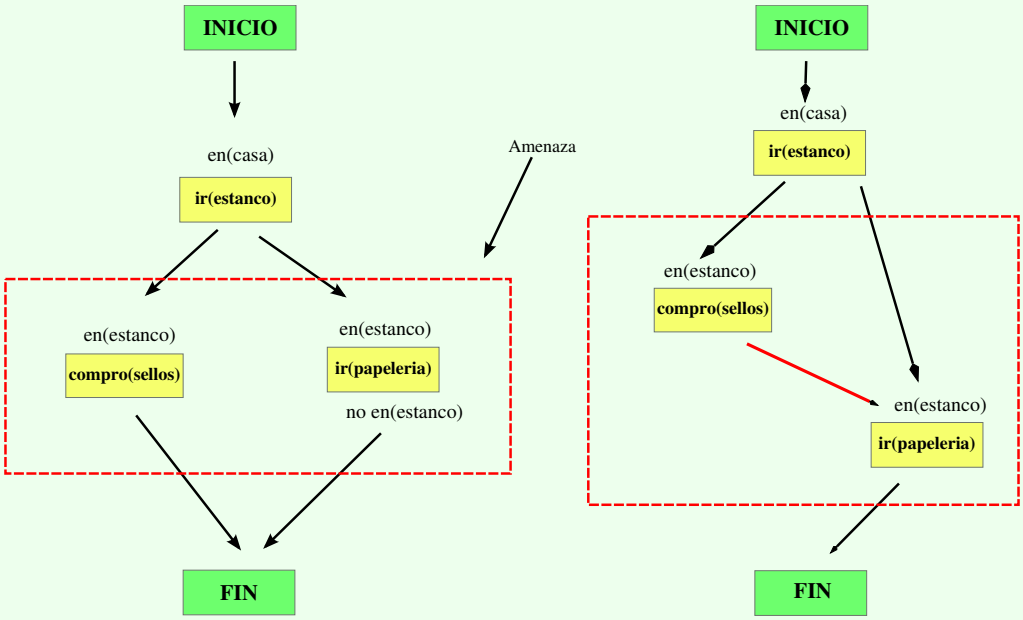
Planificación en espacio de planes - Ejemplo



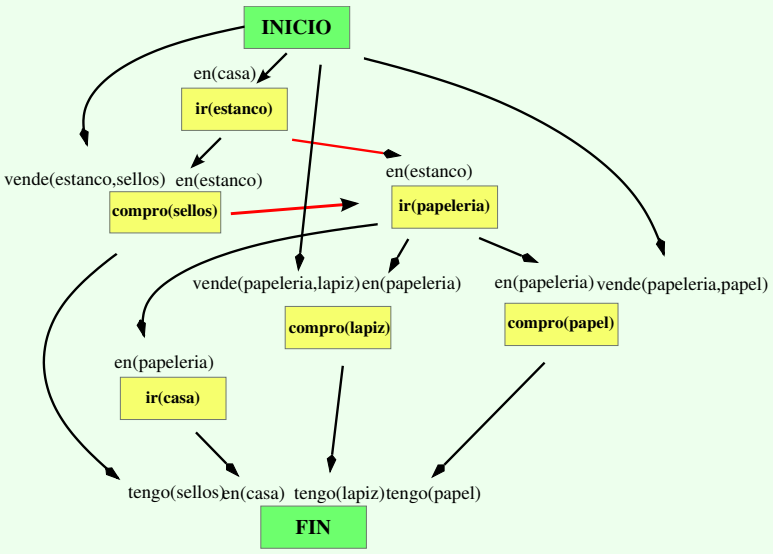
Planificación en espacio de planes - Ejemplo



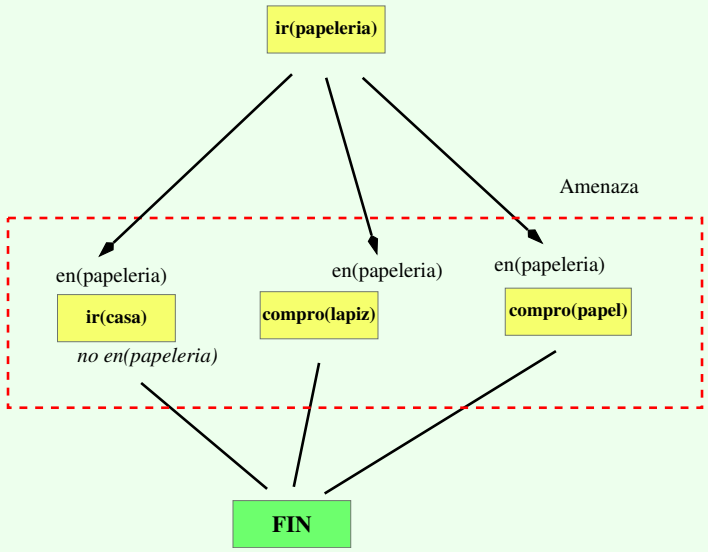
Planificación en espacio de planes - Ejemplo



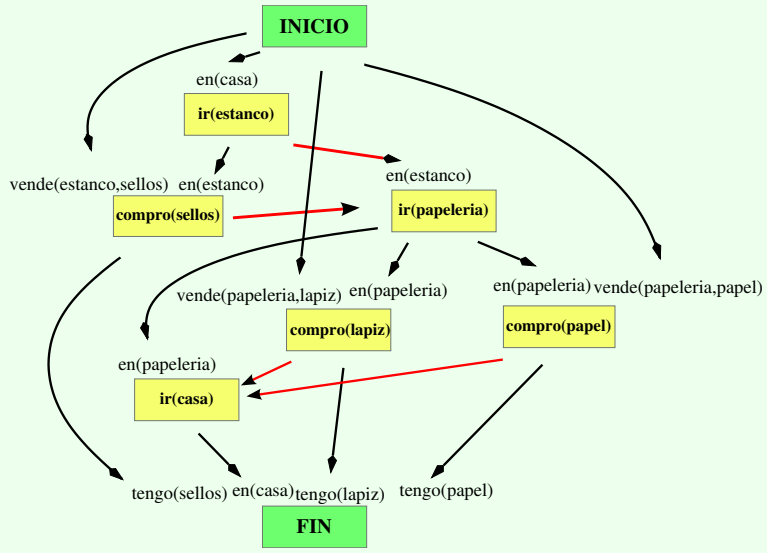
Planificación en espacio de planes - Ejemplo



Planificación en espacio de planes - Ejemplo

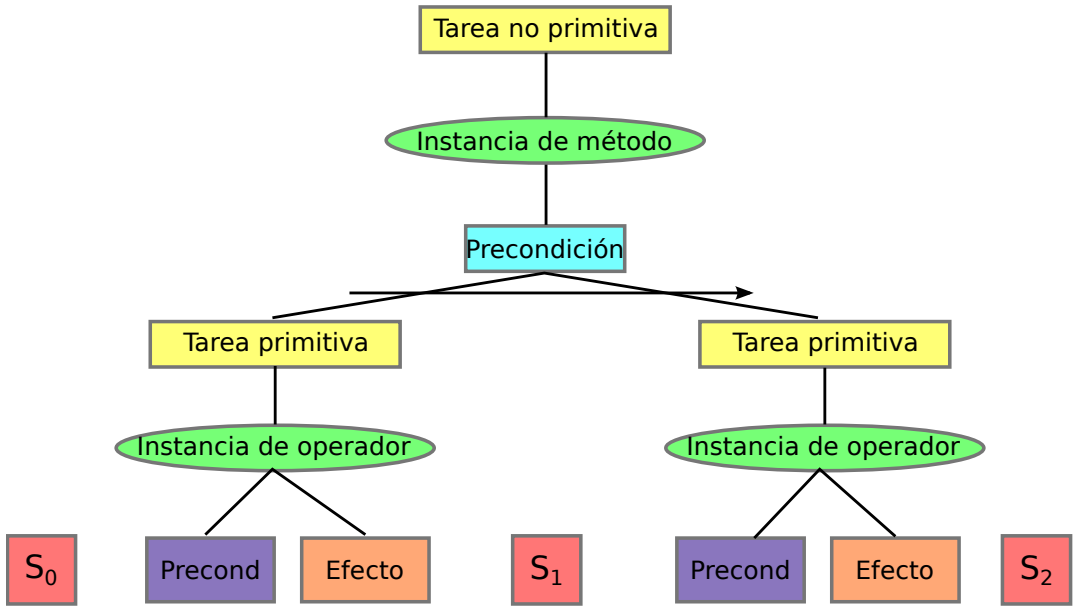


Planificación en espacio de planes - Ejemplo



- Los métodos anteriores trabajan en un único nivel de abstracción
- Nosotros podemos generar planes a diferentes niveles, desde planes de muy alto nivel a planes muy detallados
- Planificación Jerárquica:
  - Describe tareas y subtareas, y les asocia acciones.
  - Las tareas forman una red de tareas jerárquica (Hierarchical Task Network o HTN)
  - El motor de planificación es capaz de explorar los diferentes niveles de (sub)tareas

- Estados y operadores, como en planificación clásica
- No hay objetivos
- Planificación guiada por tareas
  - Tareas primitivas
  - Tareas no primitivas (descomponibles)
- Los métodos descomponen tareas en sub-tareas



- **Tarea**: una expresión de la forma $t(u_1, \ldots, u_n)$
- t es un símbolo de tarea, y cada u_i es un término
- Dos tipos de símbolos de tarea:
  - Primitivos: tareas que sabemos cómo ejecutar directamente, el símbolo de tarea es un nombre de operador
  - No-primitivos: tareas que se han de descomponer en subtareas, debemos usar un método

- **Método**: una tupla $m = (nombre(m), tarea(m), precond(m), subtareas(m))$
  - $nombre(m)$: una expresión de la forma $n(x_1, \ldots, x_n)$ donde x_i son parámetros (variables)
  - $tarea(m)$: una tarea no-primitiva
  - $precond(m)$: precondiciones (literales)
  - $subtareas(m)$: una secuencia parcialmente ordenada de las tareas $\langle t_1, \ldots, t_k \rangle$

- Dominio: métodos, operadores
- Problema: métodos, operadores, estado inicial, lista de tareas
- Solución: cualquier plan ejecutable que se pueda generar por aplicar de forma recursiva
  - métodos para tareas no-primitivas
  - operadores para tareas primitivas
- Buscamos en el espacio de descomposiciones
- Búsqueda computacionalmente más eficiente (escalable)

método:	login
TAREA:	login(User,Tienda)
PRECOND:	registrado(User,Tienda)
SUBTAREAS:	enviar-usuario(User,Tienda)

método:	login
TAREA:	login(User,Tienda)
PRECOND:	¬registrado(User,Tienda)
SUBTAREAS:	registrar-usuario(User,Tienda), login(User,Tienda)

método:	comprar-objeto
TAREA:	comprar(User,Obj,Tienda,Pmax)
PRECOND:	∅
SUBTAREAS:	login(User,Tienda), obtener-precio(User,Obj,Tienda), pagar(User,Tienda,Obj,Pmax)

método:	pagar
TAREA:	pagar(User,Tienda,Objeto,Pmax)
PRECOND:	precio(Obj,Pr≤Pmax),logged-in(User,Tienda)
SUBTAREAS:	enviar-pago(User,Tienda), procesar-pago(User,Tienda), enviar-recibo(User,Tienda)

operador:	enviar-usuario(User,Tienda)
PRECOND:	¬logged-in(User,Tienda)
EFECTOS:	logged-in(User,Tienda)

operador:	registrar-usuario(User,Tienda)
PRECOND:	¬resgistrado(User,Tienda)
EFECTOS:	registrado(User,Tienda)

operador:	obtener-precio(User,Tienda,Obj)
PRECOND:	logged-in(User,Tienda),¬ precio(Obj,P)
EFECTOS:	precio(Obj,P)

operador:	enviar-pago(User,Tienda,Obj)
PRECOND:	tarjeta-válida(User)
EFECTOS:	pagado(User,Obj)

operador:	procesar-pago(User,Tienda,Obj)
PRECOND:	pagado(User,Obj), tarjeta-válida(User)
EFECTOS:	pago-aprobado(User,Obj)

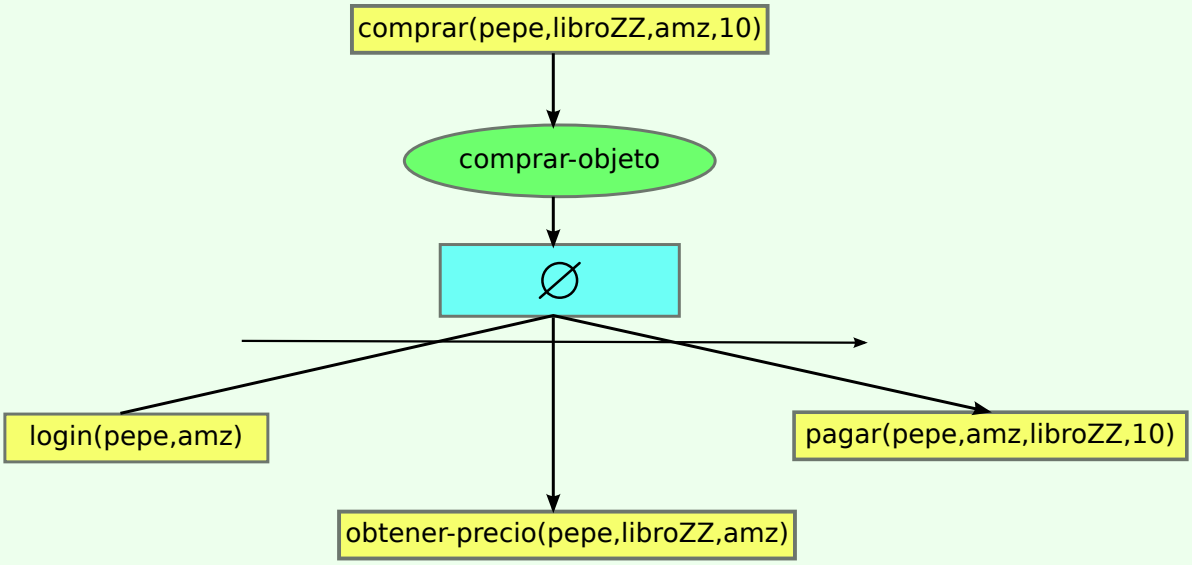
operador:	enviar-recibo(User,Tienda)
PRECOND:	pago-aprobado(User,Tienda)
EFECTOS:	recibo(User,Obj)

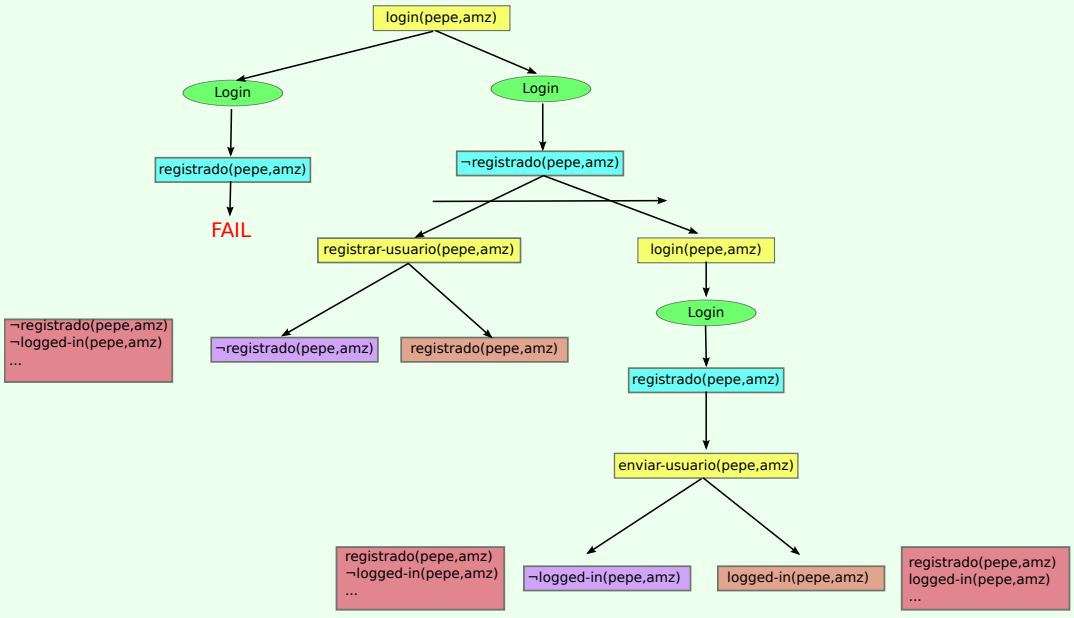
- El estado inicial sería:

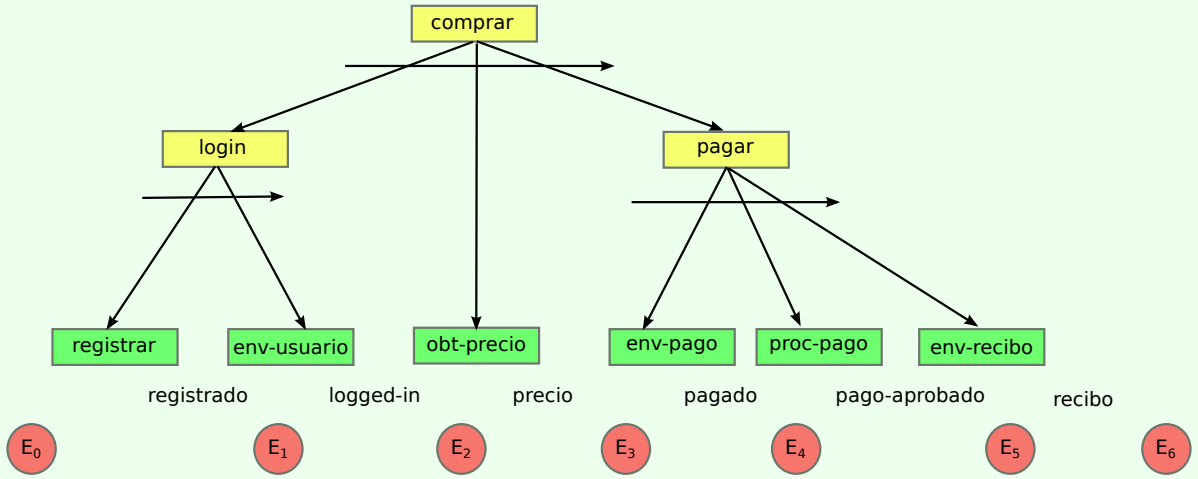
¬registrado(pepe),
¬logged-in(pepe,amz),
tarjeta-válida(pepe)

- La tarea sería:

comprar(pepe,libroZZ,amz,10)

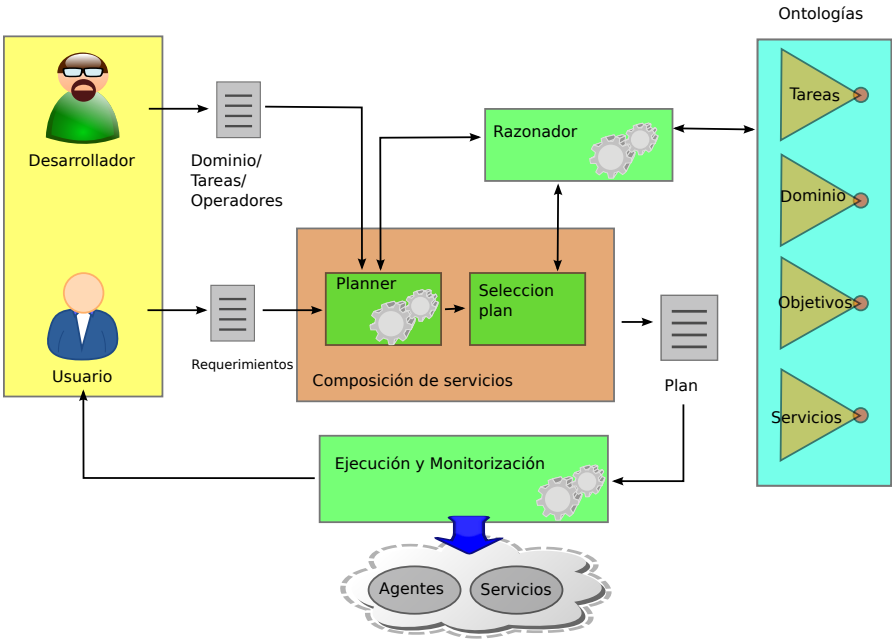






- La composición de un conjunto de servicios se puede hacer a partir de su descripción
- Cada servicio indica sus entradas, salidas, precondiciones y efectos
- Estas descripciones coinciden con las de los operadores de planificación
- **Posibilidades:**
  - Servicios atómicos: Planificación lineal/no lineal
  - Servicios complejos: Planificación jerárquica

- Los lenguajes de descripción de servicios permite describir un proceso como una descomposición de tareas
  - Servicios atómicos (operadores)
  - Servicios compuestos (métodos)
- Las entradas, salidas, precondiciones y efectos se expresan como términos de una ontología (razonamiento)
- La composición la realiza un motor de planificación
- Una vez elaborado el plan se puede ejecutar como una orquestación



- Podemos hacer la descripción de servicios más modular
- Podemos describir servicios a alto nivel (descripción de la composición)
- Podemos monitorizar automáticamente la ejecución del plan (precondiciones)
- Podemos corregir excepciones en la ejecución:
  - Replanificando acciones
  - Insertando planes de contingencia

14

Aprendizaje y Personalización

Personalización y Aprendizaje

Aprendizaje en Servicios y Agentes

Javier Béjar

CS-GEI-FIB

ECSDI - 2023/2024 2Q

Objetivos del tema

- Aprendizaje como mecanismo de adaptación
- Recomendación y perfilado de usuarios
  - Tipos de perfiles
  - Tipos de sistemas de recomendación
- Aprendizaje automático para perfilado/recomendación
  - Árboles de decisión
  - Aprendizaje Bayesiano
  - Filtrado colaborativo

Adaptacion y perfilado

Adaptación al entorno

- Dos características interesantes de los agentes son la adaptación y la proactividad
  - Adaptación es la capacidad de cambiar su comportamiento según sus experiencias y las percepciones del entorno
  - Proactividad es la capacidad de adelantarse a las necesidades a partir de las consecuencias de su modelo del mundo

Adaptación al entorno

- Estas dos capacidades pasan porque el agente genere un modelo de su entorno
- Este modelo se basará en la integración de sus experiencias en una representación
- En inteligencia artificial estos modelos se construyen mediante técnicas de aprendizaje automático

Aprendizaje en agentes

- Existen múltiples escenarios en los que se puede aplicar aprendizaje automático en agentes:
  - **Perfilado de usuarios, recomendación**: Aprender las características y necesidades de un usuario asociadas a una tarea/dominio
  - **Automatización de tareas**: Aprender los pasos para realizar una tarea a través de la interacción con un usuario o el entorno
- Se pueden aplicar diferentes técnicas de aprendizaje

Perfilado y Recomendación

- Un ámbito interesante del uso de aprendizaje en agentes/servicios es el perfilado de usuarios y la recomendación
- Se aprenden las preferencias del usuario para poder personalizar el funcionamiento del agente/servicio
- Las aplicaciones más comunes son en agentes que sirven para el acceso a algún tipo de contenido (productos, información)
- Puede realizarse a partir de la información que se obtiene de manera individual por usuario o de manera colectiva

Tipos de perfiles

- **Perfil individual estático**: Se recolecta una vez la información de un usuario y se utiliza como modelo para sus preferencias
- **Perfil individual dinámico**: Se observa o se obtiene feedback del usuario y se adapta el perfil inicial a las observaciones
- **Perfil colaborativo**: Se organizan a los usuarios en perfiles de acuerdo con sus características y se utiliza el modelo obtenido del colectivo para determinar sus preferencias

Perfil individual dinámico - Ejemplo

- **Filtrado del correo no deseado**
- El contenido son los mensajes de correo que recibe un usuario
- El comportamiento observable es el marcado como correo no deseado de mensajes
- El perfil individual son las características que diferencian los correos no deseados para el usuario
- El perfil se aplica para etiquetar nuevos correos

Perfil colaborativo - Ejemplo

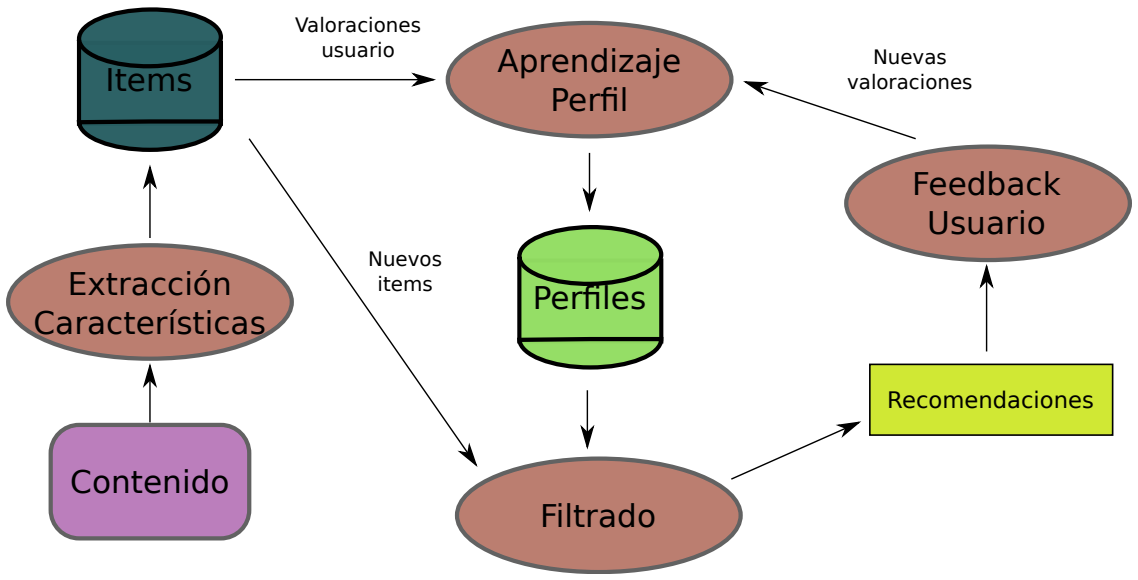
- **Recomendación de productos en una tienda**
- El contenido son los productos de la tienda
- El comportamiento observable es el historial de compra y navegación de los usuarios
- El perfil colectivo se puede obtener explícita (eg: partición de usuarios) o implícitamente (eg: usuarios con compras similares)
- El perfil se aplica para recomendar al usuario compras realizadas por usuarios con un comportamiento similar

Representación de Perfiles

- Las características en un perfil dependen del dominio de aplicación
- Pueden incluir:
  - Información propia del usuario (eg: info. demográfica)
  - Información que representa el contenido que se va a recomendar (eg: descripción de productos, texto...)
- La representación depende de la estructura del contenido
  - No estructurada (vector de atributos)
  - Estructurada (grafos de relaciones, secuencias...)

- Recomendación basada en contenido
  - Se elabora un perfil explícito (modelo) de los usuarios a partir de sus características y las del contenido
  - Los nuevos elementos se recomiendan o no dependiendo de la respuesta del modelo
- Filtrado colaborativo
  - No se elabora un perfil explícito del usuario
  - Se determina la adecuación de nuevo contenido a partir de la similaridad del usuario con otros usuarios o contenidos y sus evaluaciones de estos

- Basada en el análisis de las descripciones de contenido valorado por el usuario
- El perfil será una representación de las preferencias del usuario a partir de esas descripciones
- Este perfil permite obtener la predicción de la valoración del usuario para nuevo contenido
- Esta valoración permite tomar la decisión sobre su recomendación



Ventajas

- Las recomendaciones de un usuario son independientes del resto
- Se puede obtener una explicación de la recomendación a partir del perfil
- Se pueden recomendar contenidos nuevos que no han sido valorados por nadie

Inconvenientes

- La calidad del perfil depende de las características usadas
- Sobreespecialización en las recomendaciones, no podemos recomendar contenidos muy diferentes de los que ya conocemos
- Hacen falta un numero inicial de valoraciones por parte del usuario para poder recomendar eficazmente (*cold start*)

- El contenido puede tener una representación explícita (estructurada)
  - Una ontología define la descripción del contenido
  - eg: Productos en una tienda (libros, discos...)
- El contenido puede tener una representación no estructurada
  - Se han de utilizar técnicas especializadas de extracción de características
  - ej.: Texto de noticias → técnicas de recuperación de la información (Information Retrieval)

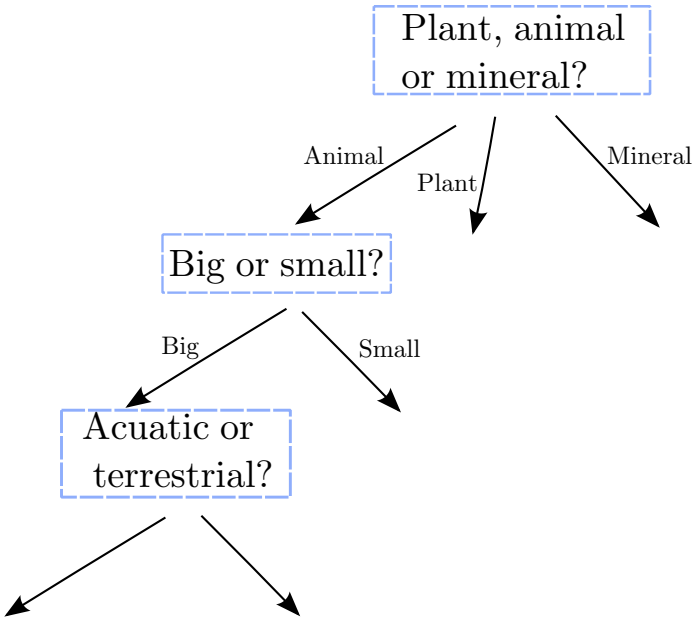
- El feedback puede ser explícito:
  - A partir de una valoración cualitativa (e.g.: gusta, no gusta)
  - A partir de una valoración cuantitativa (e.g.: una puntuación)
  - A partir de una valoración no estructurada (e.g.: el texto de una opinión)
- El feedback puede ser implícito:
  - Acciones del usuario que podemos transformar en una valoración (eg: tiempo de lectura de una noticia)

- Los métodos más utilizados para la recomendación basada en contenidos provienen del aprendizaje inductivo supervisado
- **Inducción**: Pasamos de lo específico a lo general
- **Supervisión**: Conocemos el concepto al que pertenece cada ejemplo

- A partir de ejemplos etiquetados obtenemos un modelo
- El modelo generaliza los ejemplos, representando los conceptos que definen las etiquetas
- Obtenemos lo que es común entre los ejemplos de un concepto que les diferencia de los otros conceptos

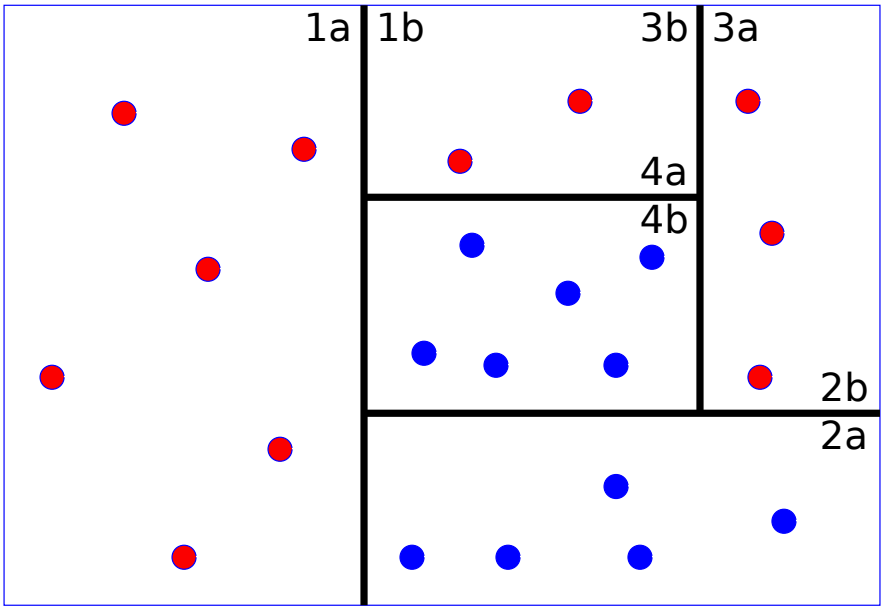
- Métodos usados en recomendación basada en contenido
  - Modelos caja blanca (podemos inspeccionar el modelo)
    - Árboles de decisión/reglas de inducción
    - Modelos probabilísticos
  - Modelos caja negra
    - Redes de neuronas artificiales
    - Máquinas de soporte vectorial
- Podemos definir el problema como:
  - Clasificación: predecir un conjunto finito de conceptos
  - Regresión: predecir una función continua

- Aprender un concepto como la búsqueda del conjunto de preguntas que lo distingue de otros
- Estas preguntas se pueden organizar de manera jerárquica (árbol)
- El árbol de preguntas nos sirve de lenguaje de representación, cada nodo es un test sobre un atributo
- Representacion es equivalente a una FND (2^{2^n} conceptos posibles)
- Búsqueda en el espacio de árboles de preguntas



- Buscar en el espacio de todas las FND es muy costoso
- Para reducir el coste computacional imponemos un sesgo
- **Restricción:** Queremos el árbol que represente la mínima descripción del concepto objetivo dados los ejemplos
- **Justificacion:** Este árbol será el mejor clasificando nuevos ejemplos (la probabilidad de tener preguntas innecesarias se reduce)
- **Navaja de Occam**: "*En igualdad de condiciones, la explicación más sencilla suele ser la correcta*"

- El algoritmo **ID3** realiza una búsqueda por ascenso en el espacio de árboles
  - Para cada nuevo nodo de decisión se elige un atributo y los ejemplos son distribuidos según sus valores
  - Este procedimiento es repetido recursivamente hasta que todos los ejemplos son del mismo concepto
- La selección de cada atributo es decidida mediante una función heurística



- Una heurística es un método aproximado para la solución de un problema
- Las heurísticas para árboles de decisión miden lo adecuado que es un atributo para formar un árbol mínimo
- Esta decisión se realiza de manera local (en cada nodo del árbol) aproximando el problema global
- Heurísticas utilizadas:
  - Entropía, entropía normalizada
  - GINI index
  - ...

- La teoría de la información estudia los mecanismos de codificación de mensajes y el coste de su transmisión
- Dados un conjunto de mensajes $M = \{m_1, m_2, ..., m_n\}$, con una probabilidad $P(m_i)$, podemos definir la cantidad de información (I) contenida en un mensaje de M como:

$$I(M) = \sum_{i=1}^{n} -P(m_i)log(P(m_i))$$

- Este valor se interpreta como la información necesaria para distinguir entre los mensajes de M (Cuantos bits de información son necesarios para codificarlos)

- Podemos hacer la analogía con codificación de mensajes:
  - las clases son los mensajes
  - la proporción de ejemplos de cada clase su probabilidad
- Árbol de decisión = codificación que distingue las clases

  (Aprender un árbol de decisión $\Longleftrightarrow$ Aprender un código)
- Buscamos el mínimo código que distingue entre las clases
- Cada atributo se evalúa para decidir si se le incluye

- La elección se realiza en cada punto de decisión
- Elegir un atributo si hace que la cantidad de información que quede por cubrir sea la menor (bits restantes por codificar)
- La elección resulta en una decisión donde los ejemplos para cada posibilidad están sesgados hacia una clase
- Una heurística calcula la cantidad de información que no cubre un atributo (Entropia, E)

- Bits necesarios para codificar los ejemplos $\mathcal{X}$ y siendo $\mathcal{C}$ su clasificación, sin ninguna información adicional

$$I(\mathcal{X},\mathcal{C}) = \sum_{\forall c_i \in \mathcal{C}} -\frac{\sharp c_i}{\sharp \mathcal{X}}\, log(\frac{\sharp c_i}{\sharp \mathcal{X}})$$

- Bits necesarios para codificar los ejemplos dado un atributo A y siendo $[A(x) = v_i]$ los ejemplos con valor v_i

$$E(\mathcal{X}, A, \mathcal{C}) = \sum_{\forall v_i \in A} \frac{\sharp[A(x) = v_i]}{\sharp \mathcal{X}}\, I([A(x) = v_i], \mathcal{C})$$

(Suma ponderada de la cantidad de I para cada partición)

Algorithm: ID3 ($\mathcal{X}$: Ejemplos, $\mathcal{C}$: Clasificación, $\mathcal{A}$: Atributos)

```
if todos los ejemplos son de la misma clase
then
    return una hoja con el nombre de la clase
else
    Calcular la cantidad de información de los ejemplos (I)
    foreach atribute en A do
        Calcular la entropia (E) y la ganancia de información (G)
    Escoger el atributo que maximiza G (a)
    Borrar a de la lista de atributos (A)
    Generar el nodo raíz para el atributo a
    foreach partición generada por los valores del atributo a do
        Árbol_i=ID3(X[a] = v_i, C ,A − a)
        generar una nueva rama con a=v_i y Árbol_i
    return El nodo raíz para a
```


Tomemos el siguiente conjunto de ejemplos de películas

Ej.	Década	País	Género	Gusta
1	70	USA	Drama	+
2	70	no USA	Comedia	+
3	80	no USA	Drama	−
4	90	no USA	Drama	−
5	90	no USA	Comedia	+
6	80	no USA	Acción	−
7	90	USA	Acción	−
8	70	no USA	Drama	+

ID3 - Ejemplo (2)

Década	N ejemplos	Gusta	no Gusta
70	3/8	3/3	0/3
80	2/8	0/2	2/2
90	3/8	1/3	2/3

G(X,década)= 1 - 0.34= 0.65

País	N ejemplos	Gusta	no Gusta
USA	2/8	1/2	1/2
no USA	6/8	3/6	3/6

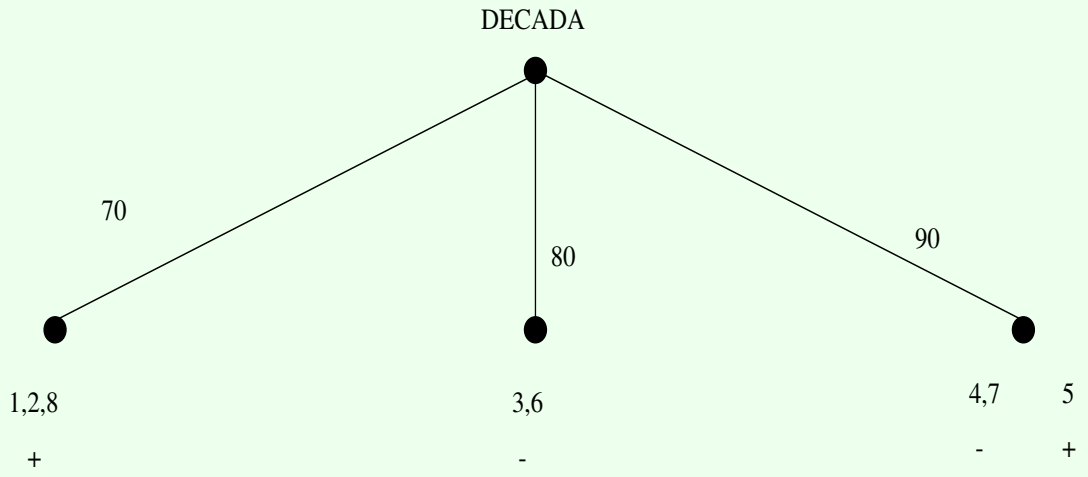
G(X,país)= 1 - 1 = 0

Género	N ejemplos	Gusta	no Gusta
comedia	2/8	2/2	0/2
drama	4/8	2/4	2/4
acción	2/8	0/2	2/2

G(X,genero)= 1 - 0.5 = 0.5

ID3 - Ejemplo (3)

Este atributo nos genera una partición que forma el primer nivel del árbol.



ID3 - Ejemplo (4)

Ahora solo en el nodo correspondiente al valor **90s** tenemos mezclados objetos de las dos clases, por lo que repetimos el proceso con esos objetos.

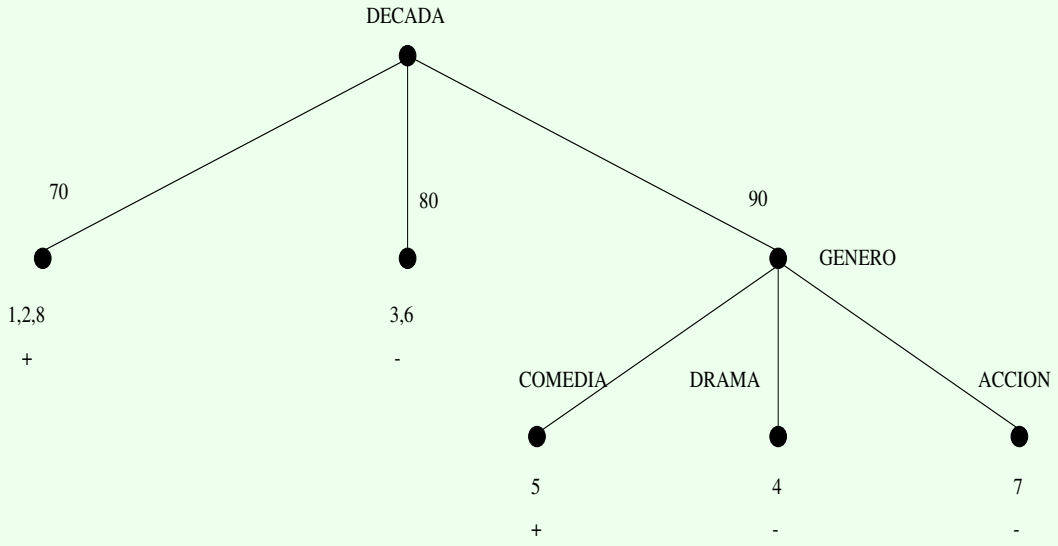
Ej.	País	Género	Gusta
4	no USA	drama	$-$
5	no USA	comedia	$+$
7	USA	acción	$-$

Ahora el atributo que maximiza la función es **género**.

$$\begin{aligned} G(X, pais) &= 0{,}918 - 0{,}666 = 0{,}252 \\ G(X, genero) &= 0{,}918 - 0 = 0{,}918* \end{aligned}$$

ID3 - Ejemplo (5)

El árbol resultante es ya totalmente discriminante y podemos usarlo como descripción del perfil



Aprendizaje bayesiano

- Los modelos basados en árboles de decisión o reglas asumen una división perfecta entre conceptos
- Para algunos problemas es más interesante tener decisiones difusas (soft)
- Esto se puede modelar usando distribuciones de probabilidad para representar los conceptos
- Asume que con una muestra de datos problema podemos obtener un modelo para evaluar nuestras hipótesis (la clasificación de nuestros datos) estimando probabilidades

- El mecanismo de razonamiento del aprendizaje bayesiano es el **teorema de Bayes**
- Este teorema liga hipótesis con observaciones

$$P(h|\mathcal{E}) = \frac{P(h) \cdot P(\mathcal{E}|h)}{P(\mathcal{E})}$$

- Obtiene de un modelo probabilístico de un problema una estimación de la probabilidad de una decisión

- Supongamos que queremos recomendar una novela a un amigo
- Podemos basar esta decisión en:
  - Nuestra opinión sobre la novela (evidencia)
  - Probabilidad a priori de los gustos de nuestro amigo
  - Como se parecen nuestros gustos a los de nuestro amigo modelado como la probabilidad conjunta de los gustos de ambos (asumimos que correlación es causalidad)

- Sabemos que la probabilidad de que a nuestro amigo le guste una novela es del 60 % ($p(h)$, probabilidad a priori)
- Sabemos que nuestro amigo tiene un gusto similar al nuestro ($P(\mathcal{E}|h)$, probabilidad condicionada)
- Suponemos que parámetros estimados de $P(\mathcal{E}|h)$ son:
  - Probabilidad de tener nosotros una opinión positiva si nuestro amigo tiene una opinión positiva es del 90 %
  - La probabilidad de tener nosotros una opinión negativa si nuestro amigo tiene una opinión negativa es del 95 %

- A nosotros nos ha gustado la novela ¿deberíamos recomendársela a nuestro amigo? (¿cual sería su predicción?) ($P(h|\mathcal{E})$)

Si enumeramos todas las probabilidades relevantes:

- P(Amigo) = $\langle 0{,}60; 0{,}40\rangle$ (pos/neg)
- P(Nuestra | Amigo=pos) = $\langle 0{,}9; 0{,}1\rangle$ (pos/neg)
- P(Nuestra | Amigo=neg) = $\langle 0{,}05; 0{,}95\rangle$ (pos/neg)

El teorema de bayes nos dice:

$$P(Amigo|Nuestra) = \frac{P(Amigo) \cdot P(Nuestra|Amigo)}{P(Nuestra)}$$

Dado que nuestra opinión es positiva (los datos) y dado que el resultado ha de sumar 1 para ser una probabilidad:

$$\begin{aligned} P(Amigo|Nuestra = pos) &= \langle P(A = pos) \cdot P(N = pos|A = pos), \\ &\quad P(A = neg) \cdot P(N = pos|A = neg)\rangle \\ &= \langle 0{,}6 \times 0{,}9; 0{,}4 \times 0{,}05\rangle \\ &= \langle 0{,}94; 0{,}06\rangle \quad (normalizada) \end{aligned}$$

Es muy probable que a nuestro amigo le vaya a gustar la novela

- El objetivo de aprendizaje es estimar la función de densidad de probabilidad (FDP) de los datos
- Estimar una FDP necesita ciertas suposiciones sobre:
  - El modelo de distribución que describe los atributos (continuos, discretos)
  - El modelo de distribución que describe las hipótesis
  - La dependencia entre las variables (todas independientes, algunas independientes...)

- La aproximación más simple es asumir que todos los atributos son independientes (no es cierto en general)
- La FDP de los atributos se puede expresar como:

$$P(\mathcal{E}|h) = \prod_{\forall i \in attr} P(\mathcal{E}_i|h)$$

- La estimación del modelo para cada atributo se puede hacer por separado $P(\mathcal{E}_i|h)$ a partir de los datos
- La probabilidad de un conjunto de hipótesis se expresa:

$$P(h|\mathcal{E}) = \underset{h \in \mathcal{H}}{\operatorname{argmax}} \left[P(h) \times \prod_{\forall i \in attr} P(\mathcal{E}_i|h) \right]$$

Algorithm: Naive Bayes

Entrada: $\mathcal{E}$ ejemplos, $\mathcal{A}$ atributos, $\mathcal{H}$ hipótesis/clases
Salida : $P(\mathcal{H})$,$P(\mathcal{E}_\mathcal{A}|\mathcal{H})$
foreach $h \in \mathcal{H}$ **do**
  $P(h) \leftarrow$ Estimar la probabilidad a priori de la clase ($\mathcal{E}$,h)
  foreach $a \in \mathcal{A}$ **do**
    $P(\mathcal{E}_a|h) \leftarrow$ Estimar la FDP del atributo de la clase ($\mathcal{E}$, h, a)

- Predecir nuevos ejemplos implica calcular la probabilidad de las hipótesis (aplicar el teorema de Bayes)

- Atributos discretos: $P(\mathcal{E}_i|h)$ se estima a partir de la frecuencia de los valores del atributo en los de datos para cada clase (distribución multinomial)
- Atributos continuos: $P(\mathcal{E}_i|h)$ se estima asumiendo una distribución modelo continua (e.g. gausiana) y estimando sus parámetros con los datos

Ejemplo

Ej.	Década	País	Género	Gusta
1	70	USA	Drama	+
2	70	no USA	Comedia	+
3	80	no USA	Drama	−
4	90	no USA	Drama	−
5	90	no USA	Comedia	+
6	80	no USA	Acción	−
7	90	USA	Acción	−
8	70	no USA	Drama	+

Década			País		Género			$P(\mathcal{H})$
70	80	90	USA	noUSA	Comedia	Drama	Acción	Gusta
1	0	.33	.5	.5	1	.5	0	+ (.5)
0	1	.66	.5	.5	0	.5	1	- (.5)

Ej: (90, USA, Drama)

$$\underset{h\in\{+,-\}}{\text{argmax}}\langle 0{,}5 \times 0{,}33 \times 0{,}5 \times 0{,}5; 0{,}5 \times 0{,}66 \times 0{,}5 \times 0{,}5\rangle =$$

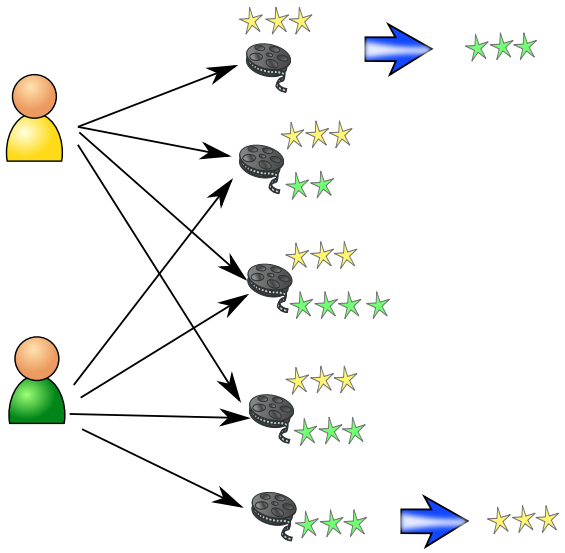
$$\underset{h\in\{+,-\}}{\text{argmax}}\langle 0{,}33; 0{,}66\rangle = 0{,}66 \Rightarrow -$$

- El modelo bayesiano es utilizado frecuentemente en la clasificación de texto
- El filtrado de correo no deseado se puede ver como un proceso de recomendación:
  - **Contenido**: Correos electrónicos
  - **Recomendación**: Leerlo o no
- En el caso del texto/documentos el contenido se representa por la frecuencia de sus palabras (*Bag of words*)

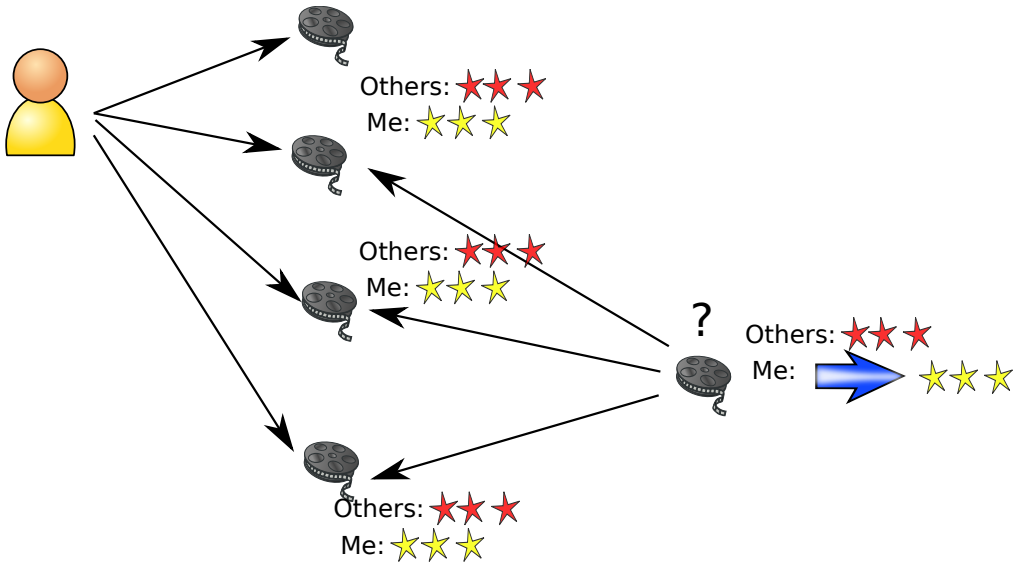
- Se basa no solo en la información del usuario, sino también en la de los usuarios que más se le parecen
- Este método permite superar el problema de no tener una buena descripción a partir del contenido
  - Asumimos que a personas similares les gustan contenidos similares
- Esto permite recomendar contenidos que no se parezcan a los que el usuario ya ha calificado (*serendipia*)
  - Si ha sido calificado por personas similares a él, seguramente se le puede recomendar

- Basado en las valoraciones de los ítems del contenido
- La recomendación utiliza una combinación de las valoraciones de los k usuarios/ítems más parecidos
- La recomendación se puede interpretar a partir de las calificaciones de los vecinos
- La valoración de un nuevo ítem se puede obtener
  - **Usuario con usuario**: Buscamos usuarios que hayan calificado el item y que tengan un conjunto de ítems calificados similares, con valoraciones similares
  - **Ítem con ítem**: Buscamos los ítems del usuario que son más similares al que queremos recomendar

Filtrado colaborativo basado en vecindad - Usuario a usuario



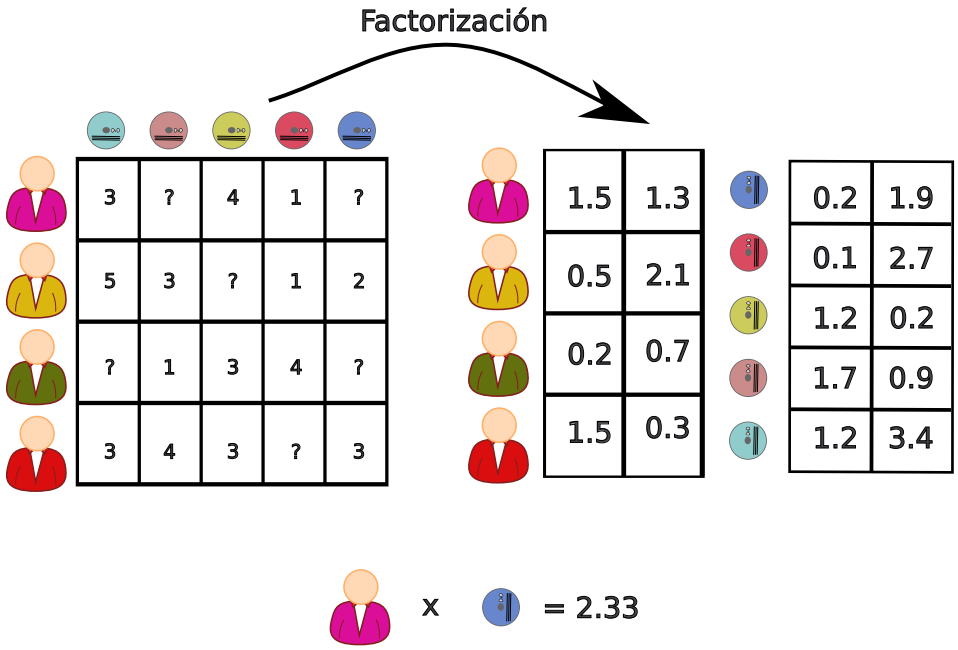
Filtrado colaborativo basado en vecindad - Item a item



Filtrado colaborativo basado en vecindad

- Si las valoraciones son un valor continuo podemos calcular la valoración como una suma ponderada de las valoraciones de los vecinos
- Si las valoraciones son valores discretos se puede aplicar una estrategia de voto ponderado
- La ponderación se obtiene de la similaridad/distancia entre los elementos que estamos combinando (usuarios/ítems)

- Construye un modelo de las interacciones entre usuarios e ítems del contenido
- El modelo representa un conjunto de variables ocultas (variables latentes) que caracterizan las asociaciones
- Estas variables latentes son una combinación de las variables originales
- Este modelo se puede obtener mediante algoritmos de factorización de matrices $M = U \times V^\top$
- A partir del modelo se obtiene una calificación para los items



	item 1	item 2	item 3	item 4	item 5	item 6	item 7	item Recom
Usuario Recom	G	NV	NG	G	G	G	NV	?
Usuario 1	G	NG	NV	NV	G	G	NV	G
Usuario 2	NG	NG	NG	G	G	NG	G	NG
Usuario 3	G	G	NG	NG	G	G	NV	G
Usuario 4	NG	G	G	NV	NG	G	NV	G
Usuario 5	G	NV	NG	NV	NG	NG	G	G

G= Gustado/NG=No Gustado/NV=No Valorado

Filtrado Colaborativo: Ejemplo 1

Asumiremos que tenemos una función de similaridad que asigna un valor 1 si las valoraciones coinciden, un -1 si no coinciden y un 0 si alguno de los usuarios no ha valorado el ítem.

	item 1	item 2	item 3	item 4	item 5	item 6	item 7	SIM
Usuario 1	1	0	0	0	1	1	0	3
Usuario 2	-1	0	1	1	1	-1	0	1
Usuario 3	1	0	1	-1	1	1	0	3
Usuario 4	-1	0	-1	0	-1	1	0	-2
Usuario 5	1	0	1	0	-1	-1	0	0

3 Vecinos: $(3 \times G + 3 \times G, 1 \times NG) \Rightarrow G$

Filtrado Colaborativo: Ejemplo 2

	item 1	item 2	item 3	item 4	item 5	item 6	item 7	item Recom
Usuario Recom	3	?	1	4	5	3	?	?
Usuario 1	4	1	?	?	4	4	?	4
Usuario 2	1	2	1	3	5	1	4	2
Usuario 3	4	3	1	2	4	5	?	5
Usuario 4	1	4	5	?	1	4	?	5
Usuario 5	5	?	2	?	2	1	4	4

Puntuación = [1...5]

Filtrado Colaborativo: Ejemplo 2

Distancia = suma del valor absoluto de la diferencia entre las valoraciones
La distancia cuando alguna valoración es desconocida es un valor desconocido, por lo que reescalamos la distancia multiplicando por el número de ítems dividido por el número de ítems en común

	it 1	it 2	it 3	it 4	it 5	it 6	it 7	DIST
Usuario 1	1	?	?	?	1	1	?	3 * (7/3) = 7
Usuario 2	2	?	0	1	0	2	?	3 * (7/5) = 4.2
Usuario 3	1	?	0	2	1	2	?	6 * (7/5) = 8.4
Usuario 4	2	?	4	?	4	1	?	11 * (7/4) = 19.25
Usuario 5	2	?	1	?	3	2	?	8 * (7/4)= 14

Ahora podemos calcular la valoración eligiendo de nuevo los tres usuarios más similares y calculándola como la suma ponderada por el inverso de la distancia de las tres puntuaciones

$$rec = \frac{\sum_{i=1..,3} p_i/d_i}{\sum_{i=1..,3} 1/d_i} = \frac{\frac{4}{7} + \frac{2}{4,2} + \frac{5}{8,4}}{\frac{1}{7} + \frac{1}{4,2} + \frac{1}{8,4}} = 3,27$$

Si queremos tener un valor entero podemos simplemente truncar al entero más cercano, lo que nos daría una puntuación de 3