Composición Dinámica

Javier Béjar

ECSDI - 2023/2024 2Q

CS-GEI-FIB CC BY NC SA

Introducción

- 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)

- 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

- 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)

- 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

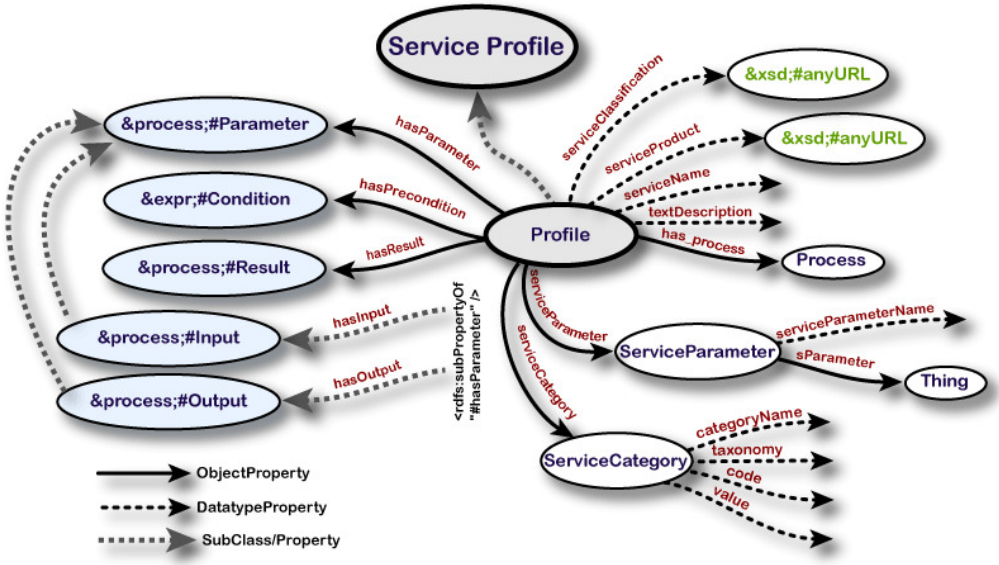
- El descubrimiento semántico de servicios pretende ir más allá de la coincidencia sintáctica
- La descripción de servicio usando conceptos de una ontología para entradas, salidas, condiciones y efectos permite el uso de deducción automática
- También se pueden utilizar las relaciones clase/subclase 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

- 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)

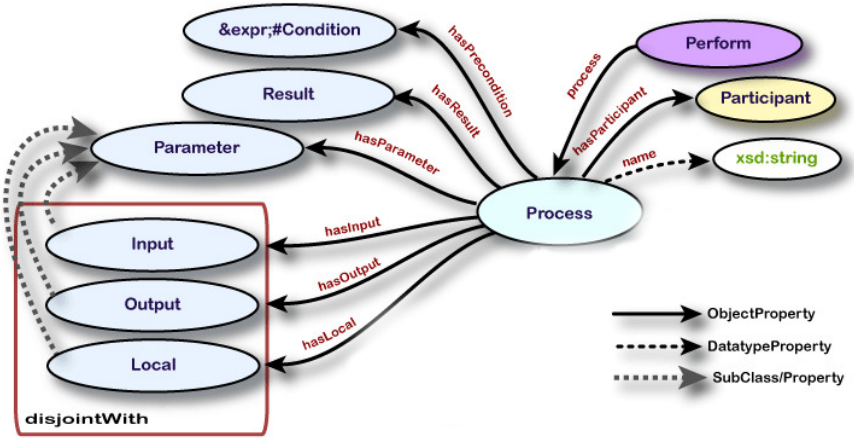
- Permite complementar la búsqueda por coincidencia literal entre las entradas y salidas de los servicios
- Las descripciones WSDL incluyen anotaciones que referencian ontologías
- La ontología indica el significado del parámetro
- Se puede razonar sobre los significados de los parámetros
  - Para calcular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

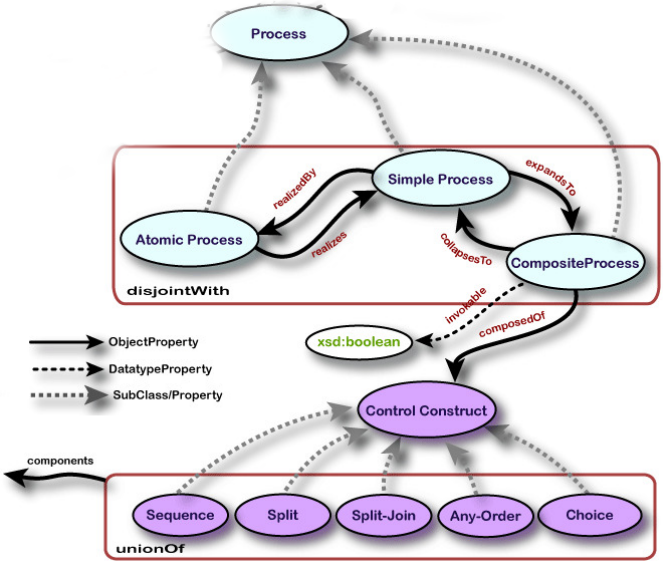
- Ontología de servicios desarrollada sobre OWL
- Un servicio se describe a partir de tres elementos:
  - **Perfil del servicio** (Service Profile): Qué requiere de sus usuarios y qué provee
  - **Modelo del servicio** (Service Model): Cómo funciona el servicio
  - **Uso del servicio** (Service Grounding):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

- 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


Service Profile
&process;#Parameter
hasParameter
serviceClassification
&xsd;#anyURL
serviceProduct
&xsd;#anyURL
serviceName
&expr;#Condition
hasPrecondition
textDescription
Profile
has_process
Process
&process;#Result
hasResult
serviceParameter
<rdfs:subPropertyOf "#hasParameter" />
serviceParameterName
&process;#Input
hasInput
ServiceParameter
sParameter
Thing
serviceCategory
&process;#Output
hasOutput
categoryName
ServiceCategory
taxonomy
code
value
ObjectProperty
DatatypeProperty
SubClass/Property

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




Process
Simple Process
realizedBy
expandsTo
Atomic Process
realizes
CollapsesTo
CompositeProcess
disjointWith
invokable
composedOf
xsd:boolean
ObjectProperty
DatatypeProperty
SubClass/Property
Control Construct
components
Sequence
Split
Split-Join
Any-Order
Choice
unionOf

- **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)

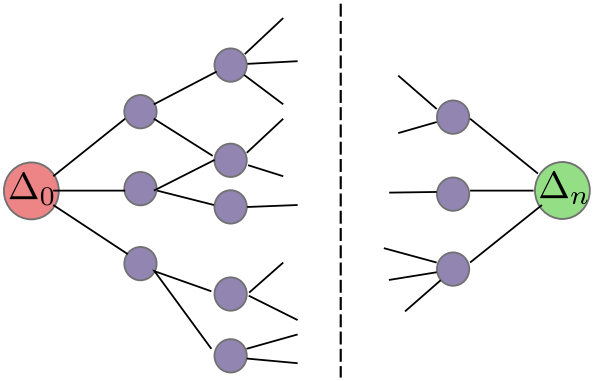
- La descripción e implementación del servicio ha de conectarse transformando los elementos descritos en el proceso a invocaciones
  - Correspondencia entre las entradas y salidas de los procesos atómicos (abstracción del proceso) y la implementación
  - Identificación de protocolos, formatos de mensajes,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

Composición Dinámica/Planificación

- 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

- 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

- 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

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

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) $\wedge$ En(av, aerop) $\wedge$ Carga(c) $\wedge$ Avion(av) $\wedge$ Aeropuerto(aerop)
EFECTO:	$\neg$En(c, aerop) $\wedge$ Dentro(c, av)

`descarga(c, av, aerop)`	
PRECOND:	Dentro(c, av) $\wedge$ En(av, aerop) $\wedge$ Carga(c) $\wedge$ Avion(av) $\wedge$ Aeropuerto(aerop)
EFECTO:	En(c, aerop) $\wedge$ $\neg$Dentro(c, av)

`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)

Solucion 1: dos aviones para hacer el traslado

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

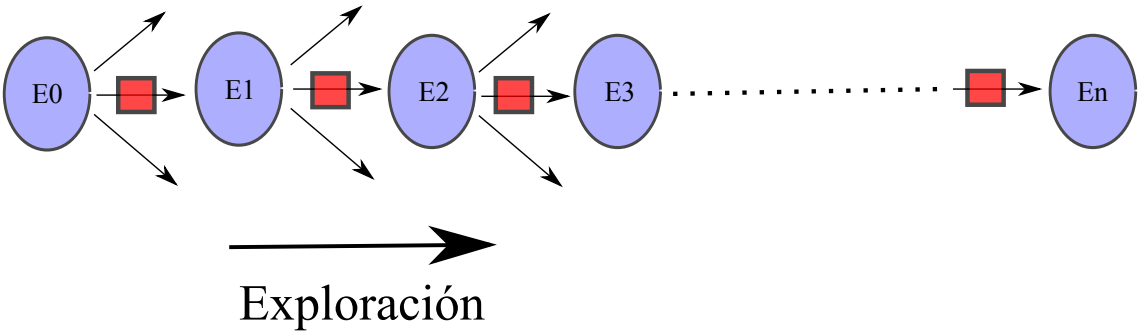
Solucion 2: usamos solo un avión

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

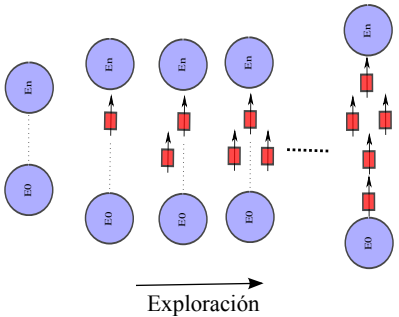
Estrategias de planificación

- 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

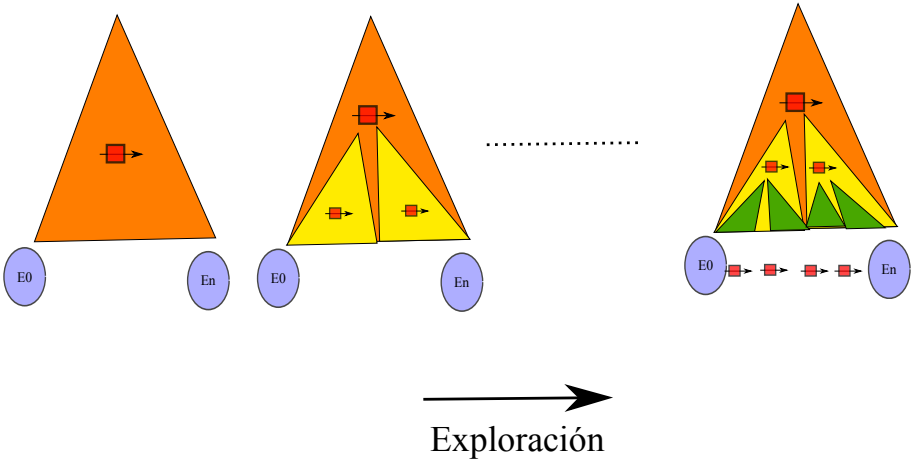
- El problema se define como una secuencia de acciones que va completando los objetivos del problema
- Se explora el espacio de caminos que conecta el estado inicial y final



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

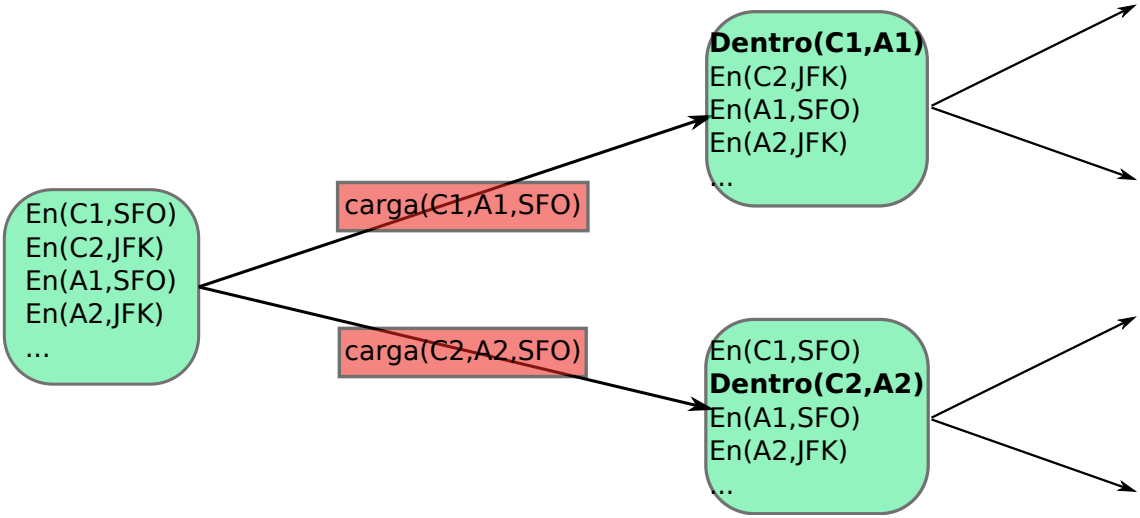


- El problema se plantea como una descomposición jerárquica de problemas
- Se explora el espacio de descomposiciones hasta obtener una secuencia de problemas primitivos



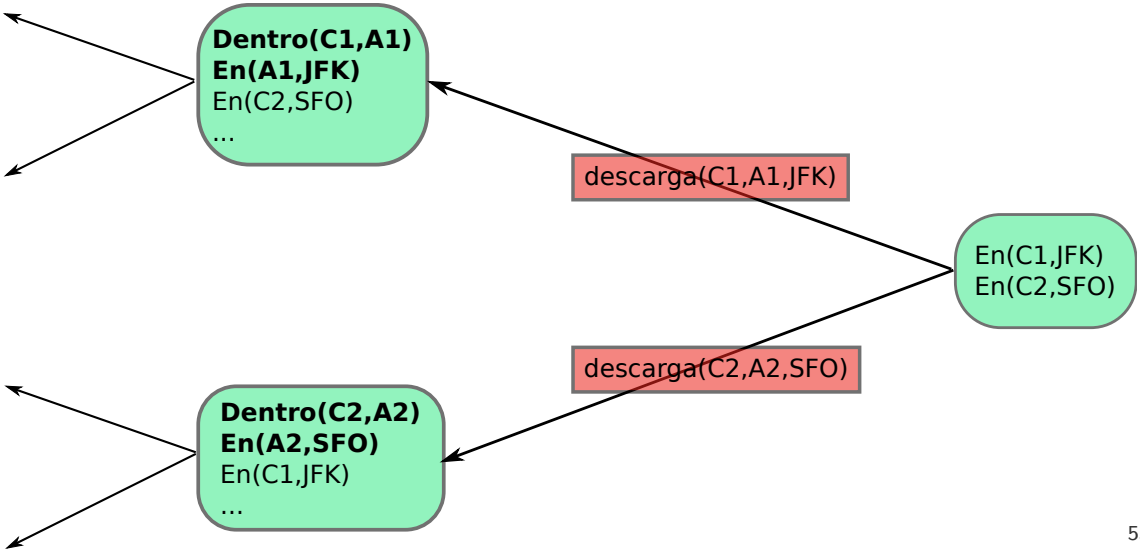
Planificacion Lineal

- 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


En(C1,SFO)
En(C2,JFK)
En(A1,SFO)
En(A2,JFK)
...
carga(C1,A1,SFO)
carga(C2,A2,SFO)
Dentro(C1,A1)
En(C2,JFK)
En(A1,SFO)
En(A2,JFK)
...
En(C1,SFO)
Dentro(C2,A2)
En(A1,SFO)
En(A2,JFK)
...

- 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)


Dentro(C1,A1)
En(A1,JFK)
En(C2,SFO)
...
descarga(C1,A1,JFK)
En(C1,JFK)
En(C2,SFO)
descarga(C2,A2,SFO)
Dentro(C2,A2)
En(A2,SFO)
En(C1,JFK)
...

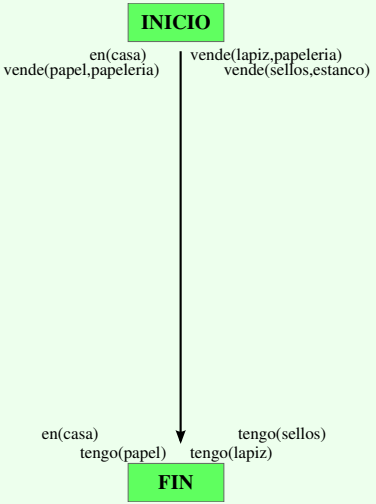
- Permite focalizar mejor la búsqueda (solo precondiciones que llevan al objetivo del problema)
- El espacio de búsqueda es aún bastante grande
- Son necesarias heurísticas generales y específicas para reducir el espacio de búsqueda

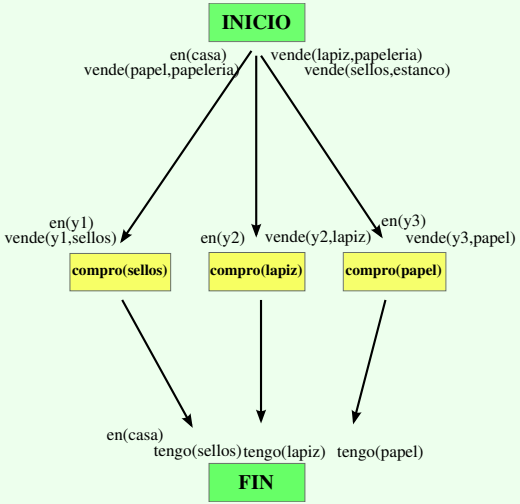
Planificación no Lineal

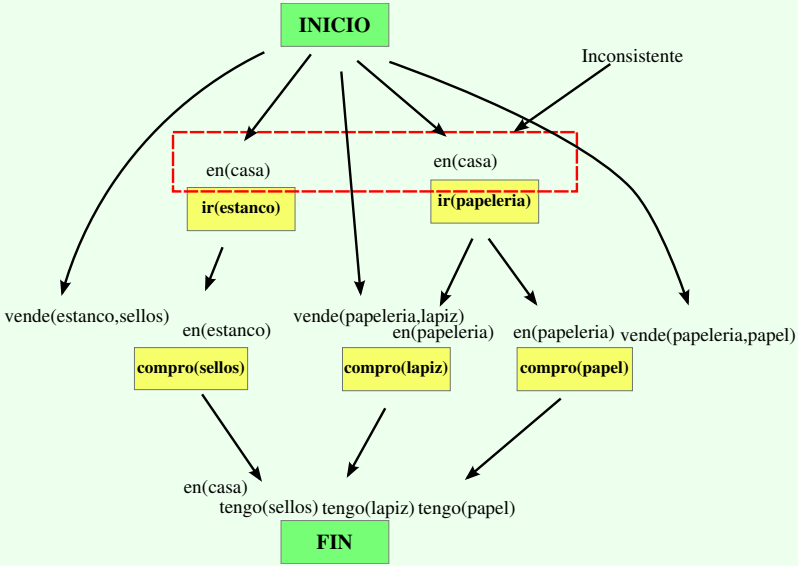
- Búsqueda **hacia atrás** desde el objetivo
- Cada nodo del espacio de búsqueda es un plan parcial que incluye
  - un conjunto de operadores parcialmente instanciados
  - un conjunto de restricciones sobre los operadores

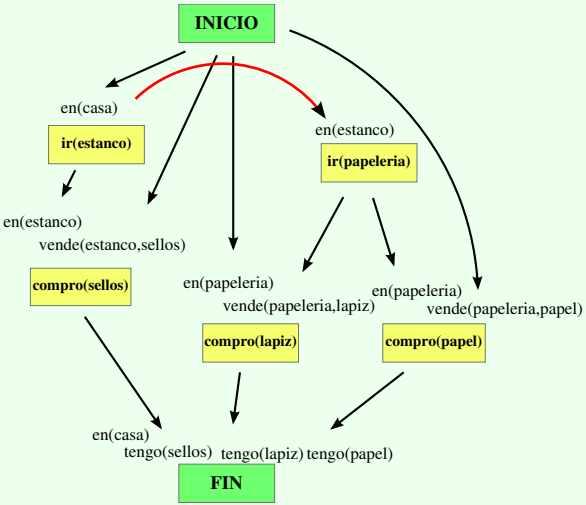
- Descomponer conjuntos de objetivos en objetivos individuales
- Planificar cada subobjetivo por separado
- Detectar y corregir conflictos entre objetivos imponiendo restricciones
- Aplicar estrategia de mínimo compromiso (mínima modificación que hace válido el plan)
- Se acaba cuando se obtiene un plan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

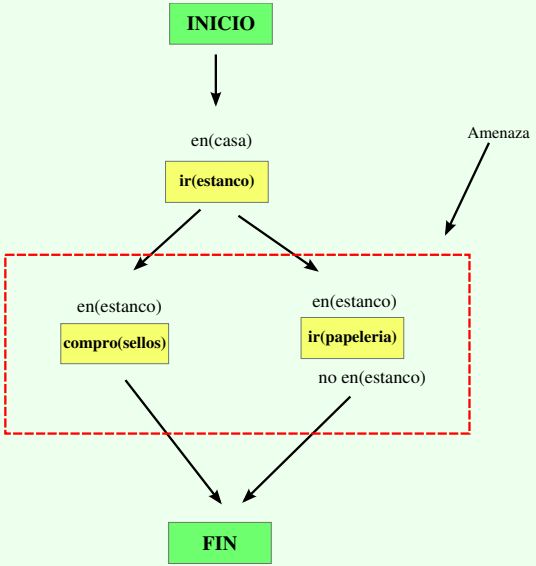
- **Problema:** escribir una carta
- Condiciones iniciales: `en(casa)`, `vende(papel, papelería)`, `vende(lápiz, papelería)`, `vende(sellos,estanco)`
- Condiciones finales: `en(casa)` `tengo(papel)`, `tengo(sellos)`, `tengo(lápiz)`
- Operadores:
  - `ir(x)` → prec: `[en(y)]`, efec: `[en(x),¬en(y)]`
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

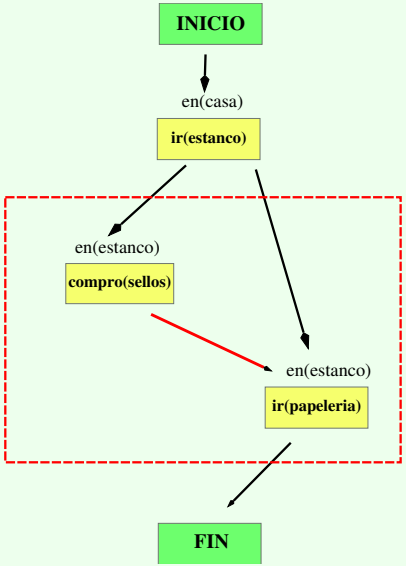

INICIO
en(casa)
vende(papel,papeleria)
vende(lapiz,papeleria)
vende(sellos,estanco)
en(casa)
tengo(papel)
tengo(sellos)
tengo(lapiz)
FIN

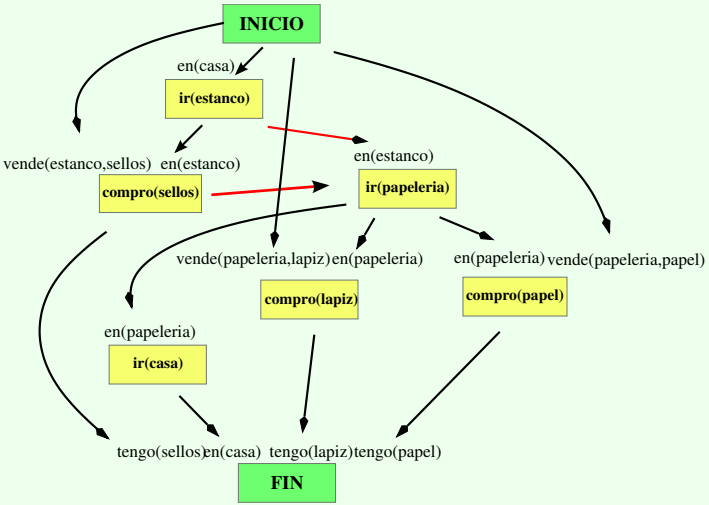

INICIO
en(casa)
vende(papel,papeleria)
vende(lapiz,papeleria)
vende(sellos,estanco)
en(y1)
vende(y1,sellos)
en(y2)
vende(y2,lapiz)
en(y3)
vende(y3,papel)
compro(sellos)
compro(lapiz)
compro(papel)
en(casa)
tengo(sellos)
tengo(lapiz)
tengo(papel)
FIN

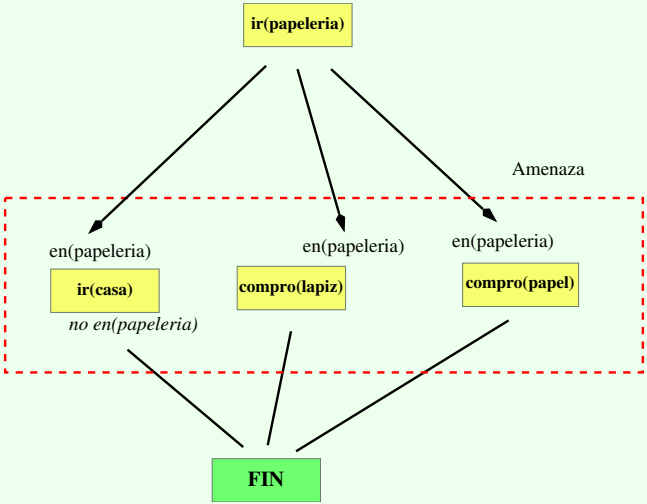

INICIO
Inconsistente
en(casa)
ir(estanco)
en(casa)
ir(papeleria)
vende(estanco,sellos)
en(estanco)
compro(sellos)
vende(papeleria,lapiz)
en(papeleria)
compro(lapiz)
en(papeleria)
compro(papel)
vende(papeleria,papel)
en(casa)
tengo(sellos)
tengo(lapiz)
tengo(papel)
FIN

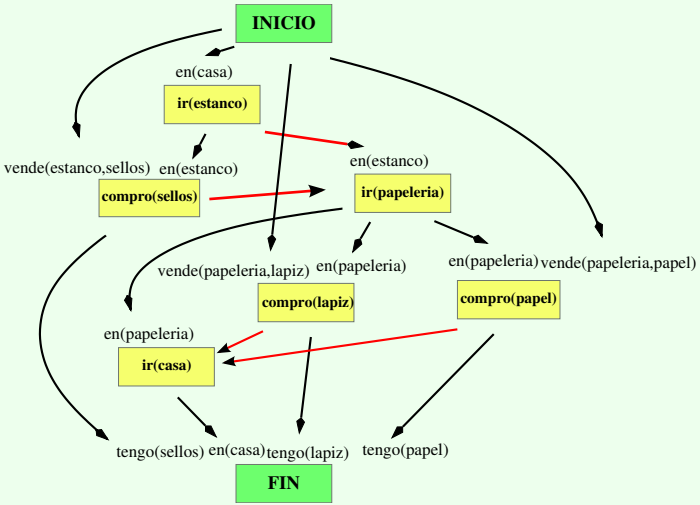

INICIO
en(casa)
ir(estanco)
en(estanco)
ir(papeleria)
en(estanco)
vende(estanco,sellos)
compro(sellos)
en(papeleria)
vende(papeleria,lapiz)
compro(lapiz)
en(papeleria)
vende(papeleria,papel)
compro(papel)
en(casa)
tengo(sellos)
tengo(lapiz)
tengo(papel)
FIN


INICIO
en(casa)
ir(estanco)
Amenaza
en(estanco)
compro(sellos)
en(estanco)
ir(papeleria)
no en(estanco)
FIN


INICIO
en(casa)
ir(estanco)
en(estanco)
compro(sellos)
en(estanco)
ir(papeleria)
FIN


INICIO
en(casa)
ir(estanco)
vende(estanco,sellos)
en(estanco)
compro(sellos)
en(estanco)
ir(papeleria)
vende(papeleria,lapiz)
en(papeleria)
compro(lapiz)
en(papeleria)
vende(papeleria,papel)
compro(papel)
en(papeleria)
ir(casa)
tengo(sellos)
en(casa)
tengo(lapiz)
tengo(papel)
FIN

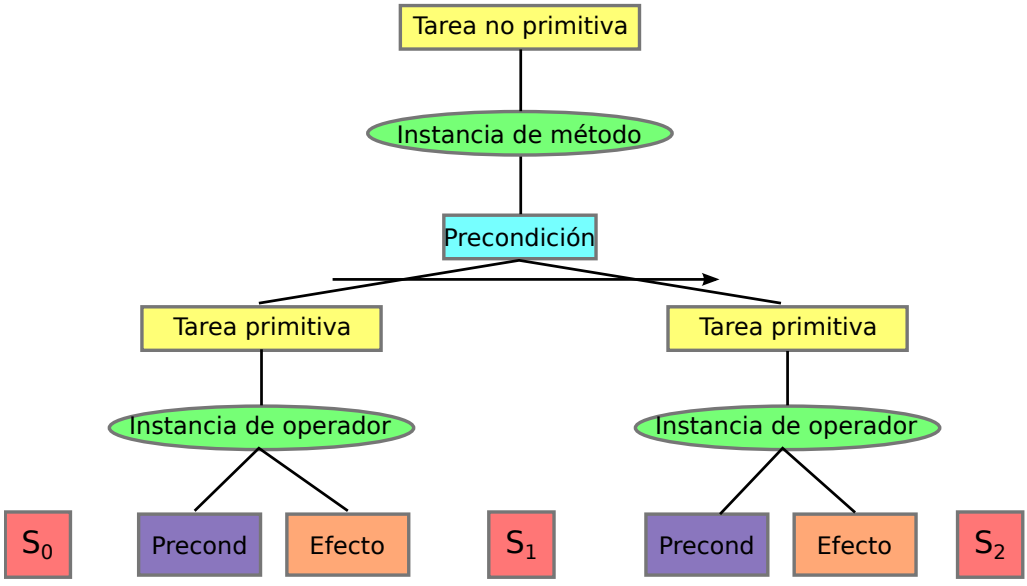

ir(papeleria)
Amenaza
en(papeleria)
ir(casa)
no en(papeleria)
en(papeleria)
compro(lapiz)
en(papeleria)
compro(papel)
FIN


INICIO
en(casa)
ir(estanco)
vende(estanco,sellos)
en(estanco)
compro(sellos)
en(estanco)
ir(papeleria)
vende(papeleria,lapiz)
en(papeleria)
compro(lapiz)
en(papeleria)
vende(papeleria,papel)
compro(papel)
en(papeleria)
ir(casa)
tengo(sellos)
en(casa)
tengo(lapiz)
tengo(papel)
FIN

Planificación Jerárquica

- 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 no primitiva
Instancia de método
Precondición
Tarea primitiva
Tarea primitiva
Instancia de operador
Instancia de operador
S_0
Precond
Efecto
S_1
Precond
Efecto
S_2

- **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:	$\neg$registrado(User,Tienda)
SUBTAREAS:	registrar-usuario(User,Tienda),
	login(User,Tienda)

método:	comprar-objeto
TAREA:	comprar(User,Obj,Tienda,Pmax)
PRECOND:	$\varnothing$
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$\leq$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)

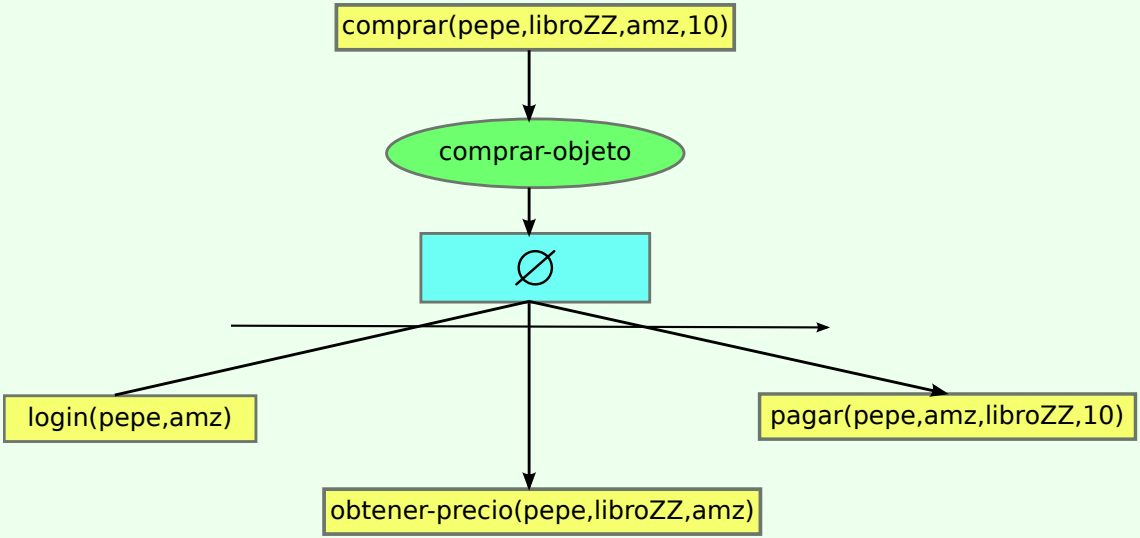
operador:	enviar-recibo(User,Tienda)
PRECOND:	pago-aprobado(User,Tienda)
EFECTOS:	recibo(User,Obj)

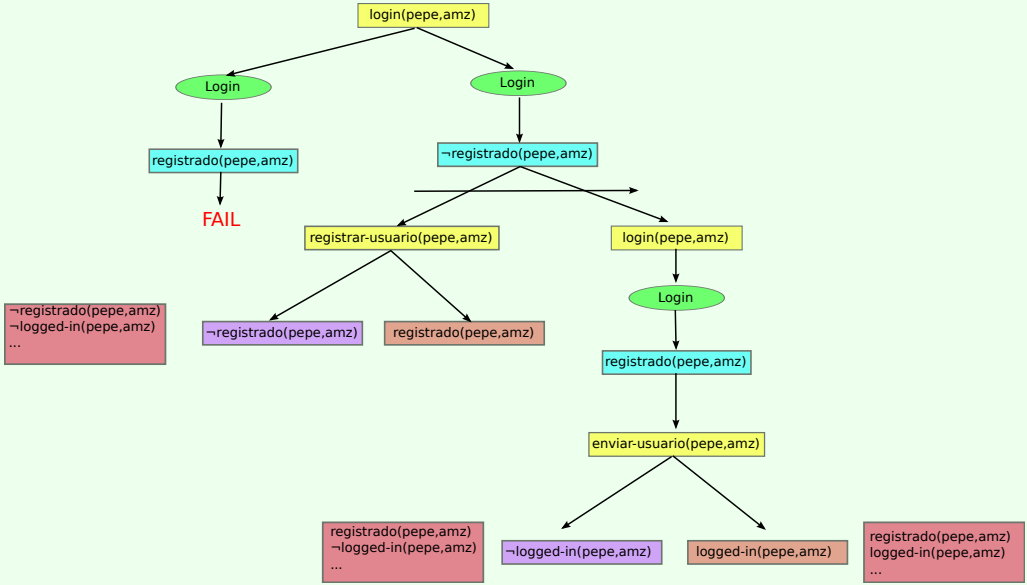
- El estado inicial sería:

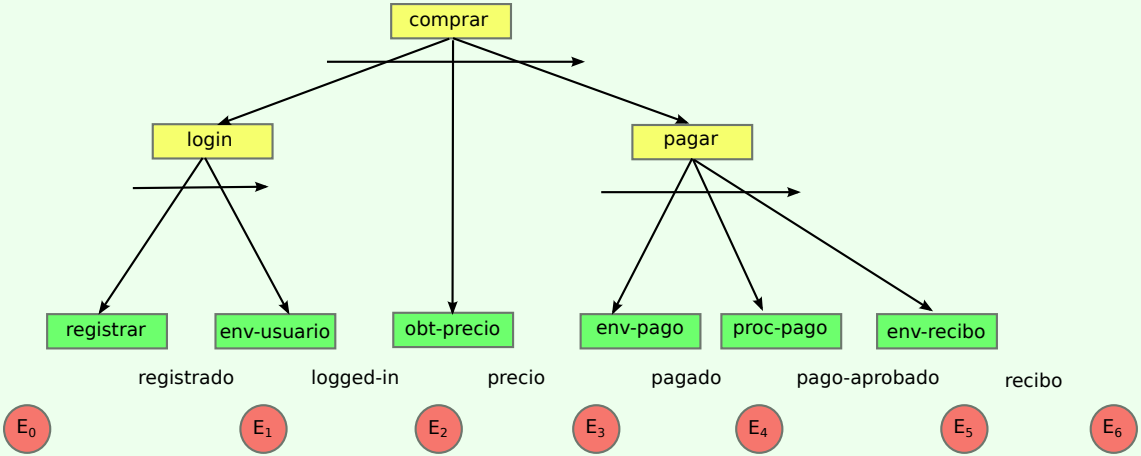
¬registrado(pepe),
¬logged-in(pepe,amz),
tarjeta-válida(pepe)

- La tarea sería:

comprar(pepe,libroZZ,amz,10)


comprar(pepe,libroZZ,amz,10)
comprar-objeto
∅
login(pepe,amz)
obtener-precio(pepe,libroZZ,amz)
pagar(pepe,amz,libroZZ,10)

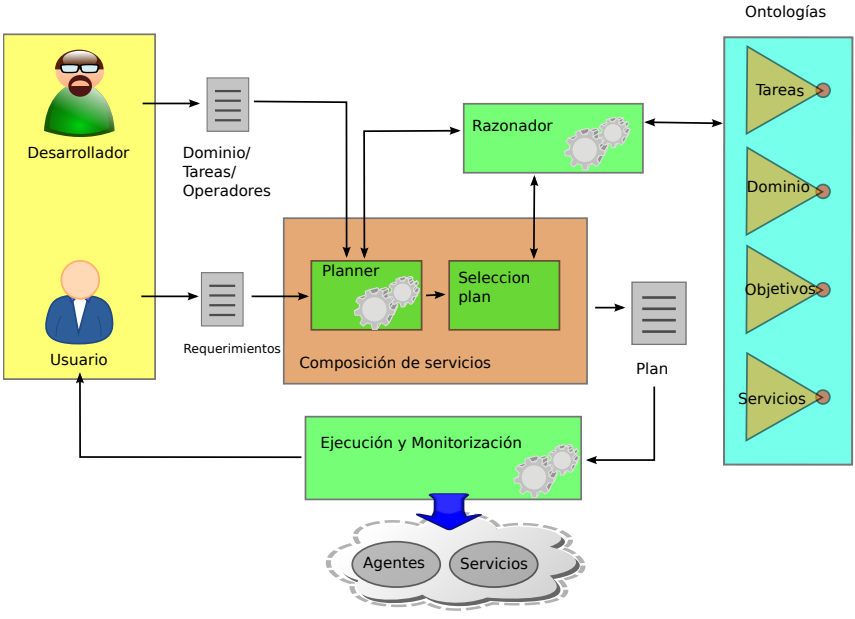

login(pepe,amz)
Login
Login
registrado(pepe,amz)
¬registrado(pepe,amz)
FAIL
registrar-usuario(pepe,amz)
login(pepe,amz)
¬registrado(pepe,amz)
¬logged-in(pepe,amz)
...
¬registrado(pepe,amz)
registrado(pepe,amz)
Login
registrado(pepe,amz)
enviar-usuario(pepe,amz)
registrado(pepe,amz)
¬logged-in(pepe,amz)
...
¬logged-in(pepe,amz)
logged-in(pepe,amz)
registrado(pepe,amz)
logged-in(pepe,amz)
...


comprar
login
pagar
registrar
env-usuario
obt-precio
env-pago
proc-pago
env-recibo
registrado
logged-in
precio
pagado
pago-aprobado
recibo
E0
E1
E2
E3
E4
E5
E6

Planificación en Agentes/Servicios

- 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


Ontologías
Desarrollador
Dominio/
Tareas/
Operadores
Razonador
Tareas
Dominio
Objetivos
Servicios
Planner
Seleccion
plan
Usuario
Requerimientos
Composición de servicios
Plan
Ejecución y Monitorización
Agentes
Servicios

- 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