Departamento de Ciencias de la Computación e
Inteligencia Artificial

Universidad de Granada

TESIS DOCTORAL

Planificación en dominios temporales usando técnicas HTN

Óscar Jesús García Pérez

Directores:
Luis Castillo Vidal
Juan Fernández Olivares

Editor: Editorial de la Universidad de Granada
Autor: Óscar Jesús García Pérez
D.L.: Gr. 1885 - 2007
ISBN: 978-84-338-4439-2

Agradecimientos:

A mis directores de tesis Luis Castillo Vidal y Juan Fernández Olivares, por animarme, guiarme y empujarme para que este trabajo fuese una realidad.

Al personal de IACTIVE Intelligent Solutions S.L., por respaldarme y apoyarme durante el tiempo que he necesitado para escribir este puñado de folios.

A miña filliña por aguantar mis nervios y mis agobios.

Gracias a todos.

I - Índice de contenido

II - *Índice de ilustraciones*

III - *Índice de algoritmos*

IV - *Presentación*

Planear: preocuparse por encontrar el mejor método para lograr un resultado accidental.

***Ambrose Bierce** (1842-1914) Escritor estadounidense.*

La capacidad de planificar se reconoce como una actividad intelectual de nivel superior. Los humanos somos capaces de realizarla prácticamente sin darnos cuenta de ello. Planificamos nuestra jornada laboral, nuestras vacaciones o hasta el futuro de nuestros hijos. En general planificamos siempre que queremos alcanzar un objetivo que para realizarse necesita que previamente se elabore un curso de actuación.

Hasta hace poco tiempo esta capacidad de planificar por adelantado, calculando previamente las acciones a realizar, se creía únicamente humana. Se sabía que había chimpancés y otros animales que transportaban herramientas (palos, piedras) de un lugar a otro a fin de resolver un problema (romper un coco, o introducir el palo en un hormiguero) a fin de conseguir comida, pero recientes estudios [101] han demostrado que curiosamente algunos simios son capaces de guardar la herramienta adecuada para posteriormente utilizara para conseguir comida.

Como investigadores en el área de la *Inteligencia Artificial* cuyo objetivo es producir un comportamiento inteligente en máquinas, con el fin de que estas mejoren nuestras condiciones de vida, conseguir que las máquinas sean capaces de planificar de una forma eficiente e inteligente ha sido un objetivo principal desde el nacimiento de esta área del conocimiento.

La *planificación automática* es el área de la Inteligencia Artificial que trata de construir planes de actuación usando programas de ordenador, que se llaman *planificadores automáticos*. La planificación automática es un problema complejo debido a varias circunstancias. La primera es que encontrar una secuencia de

acciones (plan) se convierte fácilmente, con un número medio de acciones y posibilidades entre las mismas, en un problema combinatorio intratable con las técnicas de búsqueda clásicas. La segunda viene de la dificultad para modelar el conocimiento necesario para la planificación. La planificación clásica se centraba en resolver problemas sencillos, tipo puzzle, con un conjunto pequeño de acciones y objetos diferentes con los que tratar. En problemas reales, para resolver un plan, hay que modelar cientos de acciones y miles de objetos. Los lenguajes de modelado clásicos no están preparados para describir toda la casuística de un problema real. En los problemas reales se añade además la dificultad de la *gestión del tiempo* ya que no es únicamente necesario que las acciones se ejecuten en un determinado orden, sino que se ejecuten en un instante de tiempo determinado. Por último un planificador automático generalmente no es un componente aislado y cerrado, sino que forma parte de un sistema mucho más complejo, en donde tiene que interactuar con otros actores, ya sean humanos o programas software.

El presente trabajo trata de incidir en la solución de estos problemas que son objeto de intenso estudio y trazan importantes líneas de investigación en la comunidad de planificación.

IV.i - Objetivos

Los objetivos de este trabajo consisten en presentar el planificador automático HTNP y la arquitectura de planificación construida a su alrededor. Este planificador se construye con el objetivo de satisfacer una serie de subobjetivos que posemos encuadrar dentro de las tres líneas:

- **Mejorar la eficiencia:** La eficiencia del algoritmo de planificación es fundamental para resolver problemas cada vez más complejos. El algoritmo HTNP está diseñado para tratar con problemas con cientos de acciones y miles de objetos y resolverlos de una manera eficiente. Para ello hemos utilizado un modelo de planificación basado en redes jerárquicas de tareas (*Hierarchical Task Networks HTN*) que es un modelo que permite la estructuración de las acciones en base a tareas abstractas de alto nivel que se descomponen en una red de tareas más sencillas. El modelo HTN permite la introducción de conocimiento de control y heurístico que permite acotar la búsqueda obteniendo rápidamente soluciones a problemas complejos.

- **Mejorar la expresividad de la planificación HTN:** Es importante disponer de un lenguaje de descripción de dominios lo suficientemente expresivo para codificar toda la casuística de un problema complejo y que se adecue al máximo a la forma de representar tareas de los expertos en un determinado ámbito de aplicación. Se presentará el lenguaje HTN-PDDL como

una extensión para la definición de dominios jerárquicos (con abstracciones a nivel de tarea) del lenguaje PDDL (*Planning Domain Description Language*), que además permite la especificación de conocimiento de control de la búsqueda.

- **Mejorar la expresividad y eficiencia temporal de los modelos HTN.** Una de las principales debilidades de los planificadores HTN existentes es la dificultad que tienen para tratar con problemas que hacen un uso muy extensivo de restricciones temporales complejas. Se presentarán las herramientas expresivas de las que dispone el lenguaje HTN-PDDL para la gestión del conocimiento temporal, así como de diferentes versiones del algoritmo de planificación HTNP muy eficientes en el manejo de dominios temporizados.

- **Definir una arquitectura de planificación orientada a servicios:** Puesto que el planificador no es una pieza aislada sino que forma parte de un sistema más complejo, se estudiará una arquitectura de planificación donde el planificador automático es un componente más, y en la que los distintos componentes son servicios web. Se estudiará cómo se puede resolver el problema de la comunicación con el usuario y con otros sistemas.

IV.ii - Aplicación de los resultados

Los resultados que se presentan en este trabajo de investigación siempre han buscado una aplicación práctica que se ve reflejada en varios proyectos de investigación:

- **SIADEX (***NET033957***):** Sistema Inteligente de Ayuda a la Decisión en la Extinción de Incendios Forestales, es un proyecto financiado por la Consejería de Medio Ambiente de la Junta de Andalucía, en donde se aplican técnicas de planificación automática para ayudar al director técnico de extinción a en la elaboración de planes de extinción.

- **ADAPTAPLAN (***TIN2005-08945-C06-02***):** Adaptación basada en aprendizaje, modelado y planificación para tareas complejas orientadas al usuario, es un sistema orientado a desarrollar planes educativos en una serie de materias de forma telemática adaptados al perfil de usuario.

- **SAMAP (***TIC2002-04146-C05-02***):** Sistema adaptativo multiagente para la planificación dependiente de contexto, es un proyecto que trata de construir planes de visita personalizados a los lugares de interés en una ciudad.

Además los resultados se ven avalados por diversas publicaciones (4 revistas

y 14 publicaciones en congresos nacionales e internacionales) y varios premios:

- SIADEX fue premiado como la mejor aplicación de transferencia de la tecnología durante la primera conferencia española de informática (CEDI 2005).

- El artículo que describe HTNP y la arquitectura en la que se basa, recibió el premio como mejor artículo aplicado durante la Conferencia Internacional de Planificación y Scheduling (ICAPS 2006)

IV.iii - Estructura del trabajo

La presente memoria se estructura en base a cuatro bloques o capítulos.

En el primer capítulo está dedicado a presentar el estado del arte de la planificación. Se definirá en qué consiste un problema de planificación y se hará una revisión de cómo distintos investigadores mediante el uso de diferentes técnicas de planificación lo han abordado. Se hará especial énfasis en aquellos modelos sobre las cuales apoyaremos el desarrollo de los siguientes capítulos.

Este análisis nos permitirá tener una visión clara y global de aquellos aspectos que la planificación actual cubre y de aquellos en los que aún es necesario investigar más a fondo. En base a esto se empezará el segundo capítulo planteando las necesidades que se tienen que cubrir para resolver problemas de planificación reales a los cuales se debe enfrentar un investigador en planificación. Se pondrá como caso práctico de ejemplo tres proyectos distintos SIADEX, SAMAP y ADAPTAPLAN. Una vez conocidos estos problemas se planteará una arquitectura general de planificación que cubre todos los aspectos del ciclo de vida de un plan y se describirán las características de cada una de sus partes. La pieza fundamental de esta arquitectura es el planificador HTNP del cual describiremos su estructura y el algoritmo en el que se basa. Se continuará describiendo las características más importantes del lenguaje HTN-PDDL en el cual basa su expresividad el algoritmo HTNP. Se estudiará un ejemplo práctico de aplicación de la arquitectura y del planificador presentados viendo como se utilizan en el sistema SIADEX. Se terminará con algunas comparativas y notas sobre la eficiencia del algoritmo.

En el capítulo tres se comenzará argumentando sobre la necesidad de que los algoritmos de planificación incluyan algún tipo de mecanismo para realizar razonamientos temporales ya que las aplicaciones reales necesitan disponer de planes temporizados. Se presentará el modelo de razonamiento temporal usado por el algoritmo HTNP2 que es una extensión del algoritmo HTNP capaz de obtener planes temporizados. Se analizarán las diferentes construcciones de las que dispone el lenguaje HTN-PDDL para disponer de una rica expresividad

temporal en la definición de dominios. Por último se finalizará con las modificaciones que es necesario introducir sobre el algoritmo básico HTNP para construir el planificador HTNP2 y se argumentará sobre la flexibilidad y capacidad expresiva de HTNP2 frente a otros planificadores también basados en técnicas HTN.

El capítulo cuatro comenzará argumentando sobre la necesidad de disponer de una mayor eficiencia en el algoritmo HTNP2, y se presentará una variante del mismo, el algoritmo HTNP2CL que usa el conocimiento de la estructura causal del plan para reducir el número de inferencias temporales. Estudiaremos la aplicabilidad de estos algoritmos temporales usando como ejemplos los sistemas ADAPTAPLAN y SAMAP. Por último se realizará una evaluación comparativa entre los distintos algoritmos, demostrando que éstos no son sólo más expresivos sino también más eficientes que el resto de algoritmos HTN temporales existentes en la literatura.

La memoria terminará haciendo un resumen de lo expuesto y recordando cuáles han sido las aportaciones principales al área de la planificación automática y exponiendo aquellos aspectos que aún quedan por resolver y que serán objeto de trabajo futuro.

Junto con la memoria se incluyen varios anexos con información técnica necesaria sobre el funcionamiento del planificador y el lenguaje de descripción de dominios utilizado.

1. - *Introducción a la planificación automática*

En los preparativos del combate siempre he notado que los planes son inútiles pero que la planificación es indispensable.

Dwight D. Eisenhower *General de los Estados Unidos de América y político republicano (1890 - 1969)*

Planificar por anticipado no es únicamente importante para los individuos sino también para las organizaciones y comunidades. En una organización la planificación es utilizada de tres formas. Primero, como un medio de proteger a la organización de posibles amenazas que surgen de un entorno peligroso y cambiante, segundo, como forma de mejorar la situación actual global de la organización, y por último como forma de organizar y coordinar a todos los miembros de la organización en persecución de unos objetivos comunes.

Planificar cuando el conjunto de variables en el entorno a considerar, el número de recursos implicados y de tareas a realizar es elevado se convierte en una tarea muy compleja, más si tenemos en cuenta criterios de optimalidad y de integridad en el plan obtenido, que puede requerir del trabajo conjunto de un grupo de personas durante mucho tiempo.

Esta triple enfoque de la planificación y la dificultad del problema convierten a la *planificación automática* en un área de conocimiento dentro del ámbito de la *Inteligencia Artificial* de gran interés para empresas e instituciones gubernamentales que soportan y promueven la investigación en este sentido.

La planificación automática estudia la construcción de *planificadores* que son un programas de ordenador que generan planes de actuación para resolver *problemas* en un determinado *dominio.*

Se han desarrollado planificadores automáticos en una gran cantidad de

contextos como la robótica [38] sistemas de manejo de emergencias (inundaciones [113], operaciones de evacuación y rescate [56] [27], incendios forestales [105] [87], gestión de procesos empresariales [71], sistemas de manufacturación [70], o incluso en misiones espaciales [63] [96] [42]. Este éxito de la planificación automática para resolver problemas reales está haciendo crecer el interés en la materia y aumentar la presión sobre los investigadores para construir planificadores mas amigables, que resuelvan un mayor número de problemas, y para reducir los costes de adaptación e implantación, de ahí la gran actividad investigadora en el área con la aparición continua de nuevas técnicas y teorías.

1.1. - Conceptos básicos

Desarrollar un planificador automático específico para cada una de los posibles dominios de aplicación de esta tecnología sería una labor ardua y costosa. Además cuando quisiéramos adaptar el planificador para resolver un tipo de problema distinto o bien cuando las condiciones del dominio para el cual el planificador fue diseñado cambiasen habría que rehacer el planificador completamente.

Por ello desde los inicios de la planificación se han tratado de desarrollar algoritmos de planificación **independientes del dominio** de aplicación. Esta separación requiere que, por un lado dispongamos de unos modelos generales que nos permitan almacenar la información del contexto con el que estamos trabajando y por otro lado diseñemos los algoritmos que operen sobre estas estructuras.

El algoritmo de planificación deberá ser capaz de comprender el modelo de representación de la información y ser capaz de razonar sobre ella, pero sin estar especializado en el dominio en concreto. Un **problema de planificación** se formula en base a tres elementos (ver ilustración 1):

- Una representación de las variables y los objetos del contexto, así como de un conjunto de **acciones** que podemos aplicar sobre ellos y cuyos **efectos** alteran el entorno. Estos elementos definen el **dominio** de planificación.

- Una representación del **estado** en el que se encontrarán los objetos del dominio en el entorno antes de que empecemos la ejecución de nuestro plan llamado **estado inicial**.

- Una representación de la meta u **objetivo** que deseamos cumplir tras la ejecución de nuestro plan.

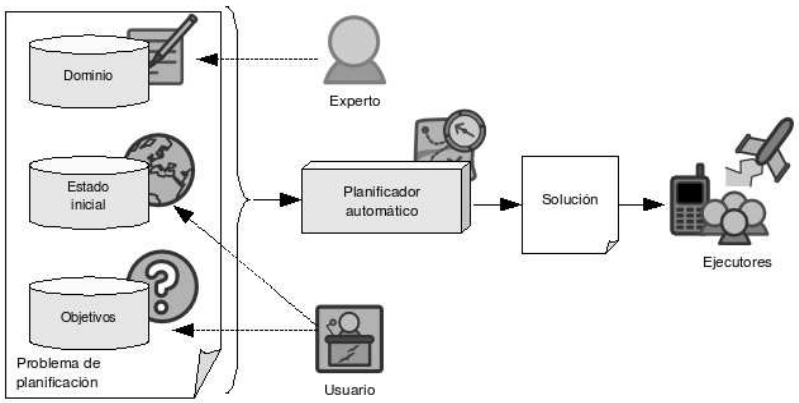


Ilustración 1: Partes que componen un problema de planificación

El problema de planificación consiste en alcanzar el objetivo, partiendo del estado inicial, utilizando las acciones y objetos definidos en el dominio.

Estas representaciones deben tener una sintaxis y una semántica bien definida además de ser lo suficientemente expresivas para poder resolver una gran cantidad de problemas en diversos dominios de aplicación.

La labor de un investigador en el área de planificación no se reduce exclusivamente a desarrollar teorías y escribir algoritmos de planificación más eficientes y expresivos, sino que también tiene que realizar labores de *Ingeniería del Conocimiento* con el fin de extraer el modelo de representación adecuado para el dominio. Además el investigador debe dotar al sistema de planificación de herramientas que permitan la comprensión del plan resultante y facilitar la interacción y comunicación con el mismo por parte de los usuarios finales. Como se puede apreciar en la ilustración 1, hay otros actores involucradas en el proceso de planificación, distintos de los investigadores y desarrolladores del planificador:

- **Expertos en el dominio de aplicación** que conocen el comportamiento del entorno y las medidas a tomar para alcanzar los objetivos. A partir de su experiencia se extrae la información necesaria para construir el *dominio* del planificador, en un proceso puro de ingeniería del conocimiento.
- **Operadores o usuarios finales**. Son los encargados de mantener

actualizada la información o estado del cual parte el planificador. Además son los que solicitan los servicios al planificador automático para resolver un *problema*, descrito como un conjunto de objetivos.

- **Ejecutores del plan**, son los agentes (software o humanos) que finalmente reciben una parte o el plan completo y realizan las actuaciones en él contempladas.

Una vez que se dispone del modelo necesario para la representación del conocimiento asociado al planificador, se debe diseñar el algoritmo de planificación que comprenda y haga uso de este conocimiento y que sea capaz de resolver el tipo de problemas para los que se necesita.

Debido al gran número de aplicaciones en las que se puede usar un planificador automático existen también una gran cantidad de modelos de planificación orientados a resolver diferentes tipos de problemas. Por ello podemos definir distintos criterios para realizar la clasificación.

1.2. - Clasificación de los modelos de planificación

Atendiendo a la forma en que se va construyendo el plan resultante [136] podemos distinguir entre:

- Planificación **por refinamiento**, según la cual el plan se va construyendo por la adición incremental de acciones y restricciones a un plan inicial vacío. También es planificación por refinamiento cuando se van eliminando acciones insertadas previamente.

- **Planificación mediante transformación**, que a diferencia de la anterior el planificador añade o elimina acciones, restricciones o relaciones de orden al plan en construcción.

Otro clasificación sería en base a cuales son los elementos básicos que perfilan un plan.

- **Planificación generativa**, que se caracteriza por que los elementos que se utilizan son acciones y restricciones entre las acciones. La mayoría de los planificadores se engloban dentro de este ámbito.

- **Planificación basada en casos**, es un modelo de planificación mediante t*ransformació*n que parte de una librería de casos o trozos de plan y los adapta para resolver un determinado problema [65] [67] [57].

Otra posible clasificación es en base a cómo actúa el algoritmo de búsqueda subyacente.

- Tenemos por un lado, algoritmos que buscan en el **espacio de estados.** Estos algoritmos pueden realizar una búsqueda hacia adelante o *progresiva* (partiendo del estado inicial) o una búsqueda hacia atrás o *regresiva* (partiendo de los objetivos).

- Y por otro lado tenemos a los algoritmos que buscan en el **espacio de planes**, que pueden ir construyendo un plan *totalmente ordenado* o *parcialmente ordenado.*

También se pueden clasificar en base a como el planificador maneja la incertidumbre, o si es independiente del dominio o no.

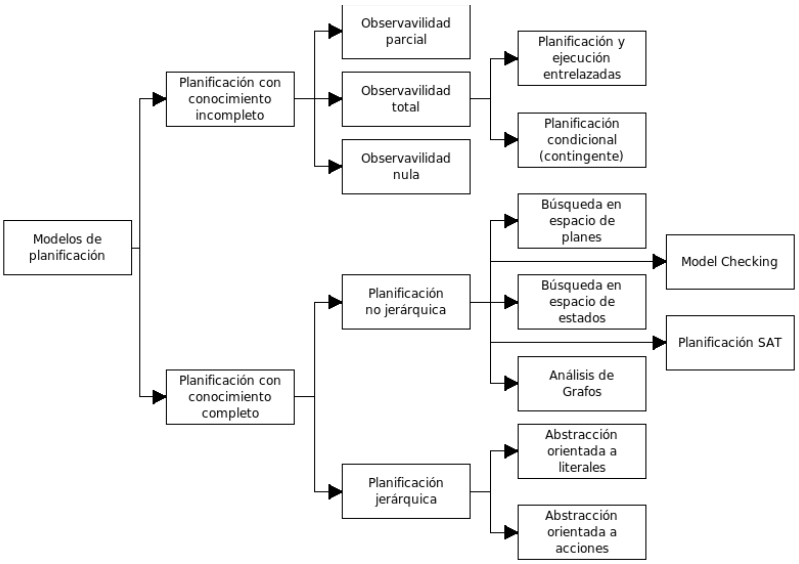


Ilustración 2: Clasificación de los distintos modelos de planificación

Obviamente esta clasificación no es categórica sino que existen modelos de planificación que mezclan varios enfoques [134] [78]. En las siguientes secciones para explicar los distintos modelos de planificación, se seguirá la clasificación mostrada en la Ilustración 2 que mezcla la capacidad que tienen los planificadores para manejar la incertidumbre en un nivel con el modelo de planificación que utilizan en el siguiente. Además trataremos de seguir en la medida de lo posible el orden cronológico en el cual fueron apareciendo los distintos modelos. En un primer nivel distinguiremos entre los planificadores que tienen conocimiento

completo, de aquellos planificadores que no pueden tenerlo. Dentro de los que no tienen observabilidad completa separaremos dependiendo de hasta donde llega su observabilidad. En los planificadores en los que suponemos conocimiento completo, distinguiremos entre los que siguen un modelo no jerárquico, es decir no hay abstracciones ni en las acciones ni en los literales de los que si lo son.

1.3. - Planificadores con conocimiento completo

La planificación con conocimiento completo es la base de la planificación clásica y la mayoría de los planificadores y del esfuerzo investigador se ha realizado en ese área. La planificación con conocimiento completo se establecen en base a tres suposiciones principales que facilitan y simplifican los problemas de planificación:

1. **Omnisciencia**: La representación del entorno es completa y correcta. El planificador dispone de toda la información, por tanto partiendo de un estado inicial perfectamente definido y de un objetivo satisfacible alcanzará una solución.

2. **Efectos deterministas**: El planificador conoce perfectamente los efectos de una acción que siempre son los mismos.

3. **Causa de cambio exclusiva:** Los únicos cambios producidos en el estado de nuestro entorno son provocados por acciones planificadas por el planificador. No hay cambios provocados por otros agentes o elementos externos. De esta forma se evita el problema de tener que observar el mundo, ya que todos los cambios están previamente previstos.

1.3.1. - *Planificadores clásicos*

Los primeros planificadores que surgieron en los orígenes de la planificación como GPS (1963) (*General Problem Solver*) [100] o QA3 (1969) [54] partían de estas hipótesis. GPS era un razonador que trataba de resolver cualquier problema de una manera general mediante una técnica de búsqueda basada en el análisis de medios-fines [102]. QA3 era un demostrador de teoremas que resolvía problemas simples de comportamiento de un agente en un entorno y que puede ser considerado como un planificador básico.

QA3 supuso un gran avance y estaba basado en el cálculo de situaciones de McCarthy y Hayes [61]. En el cálculo de situaciones usado por QA3 el estado es representado en base a un conjunto de predicados y las consecuencias de un

evento son descritas mediante *axiomas de efecto*. Un axioma de efecto es una regla en cuyo antecedente aparecen las condiciones que se deben de producir en el entorno para que la regla pueda ser disparada, y en el consecuente aparecen los cambios producidos por dicha regla en el entorno. Usando estos axiomas de efecto se pueden realizar inferencias lógicas que permiten obtener una secuencia de pasos para alcanzar el estado final u objetivo partiendo de un estado inicial.

Sin embargo el cálculo de situaciones presenta un problema importante conocido como el *problema del marco* (frame problem) [61]. Debido al mecanismo de inferencia lógica usado en el cálculo de situaciones, en los axiomas de efecto es necesario codificar no sólo lo que cambia en el entorno, sino también lo que no cambia. Para hacer esto Green [54] propuso incluir una serie de axiomas, llamados *axiomas de marco*, que describen la parte del mundo que permanece inalterada.

Aún así el problema de la representación del estado y del cambio no quedo resuelto. Aunque la representación era formalmente correcta, hacer la descripción de un dominio incluso sencillo hacía necesario escribir una gran cantidad de axiomas, y los mecanismos de inferencia, ya de por sí poco eficientes, se hacían insoportables con un número de axiomas tan alto.

Estos problemas llevaron a abandonar la idea de trabajar con resolutores de propósito general. Hubo una fase de crisis en el campo hasta que se buscó un nuevo enfoque para intentar resolver problemas más específicos.

A principios de los años 70 apareció un planificador llamado STRIPS [37] cuya representación, orientada a resolver exclusivamente problemas de planificación fue revolucionaria y aún hoy se sigue utilizando.

Con STRIPS apareció el concepto de acción como elemento de representación de cambio. La *Asunción de STRIPS* presupone que el valor de verdad de un predicado no es afectado por la ejecución de una acción a no ser que aparezca explícitamente como efecto de la misma.

La representación de STRIPS de estados y acciones se basaba en la lógica de primer orden. En lógica de primer orden, un predicado se describe como una tupla de la forma *(cabeza* a_1, a_2,...,a_n*)* donde *cabeza* es el nombre del predicado y a_1,a_2,...,a_n es un indeterminado número de argumentos que pueden ser variables sin instanciar, variables instanciadas o constantes.

En planificación es muy conocido un problema clásico que trata de simular el comportamiento de un brazo robótico o una grúa que tiene que apilar o desapilar cajas partiendo de una situación inicial hasta alcanzar una configuración final determinada. Usando este ejemplo (ver Ilustración 3), podemos describir que un bloque marcado como "B" está encima de otro bloque "A" con el siguiente

predicado *(encima B A)* y que el bloque "C" está encima de la mesa de a siguiente forma *(encima C mesa)*.

Construcciones mas complejas se pueden definir usando expresiones lógicas de primer orden. Las expresiones lógicas son fórmulas compuestas de átomos, conectores, operadores, términos y cuantificadores usando algunas reglas de buena composición.

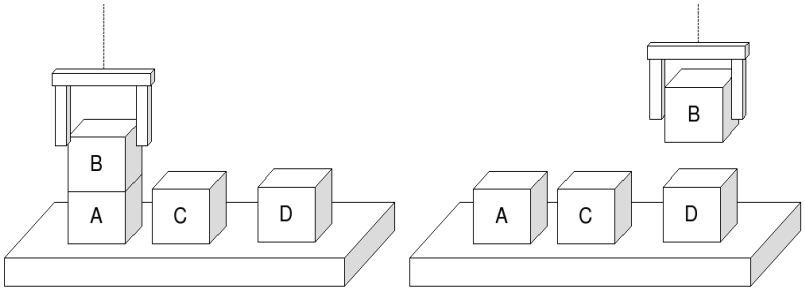


Ilustración 3: Mundo de los bloques

Los estados, usando la representación de STRIPS se describen usando un conjunto de predicados afirmados que son ciertos. Todo predicado que no aparezca en la descripción del estado se supone falso. Esto es conocido como *la hipótesis de mundo cerrado*, que aunque no es muy realista, es muy cómoda a la hora de evitar inconsistencias en la representación del estado. Así por ejemplo la descripción del estado observado en la Ilustración 3 (primero) sería: *(encima B A), (encima A mesa), (cogido brazo B)*, (encima C mesa) (encima D mesa) mientras que no habría que describir los predicados tales como: *(no (encima B A))*.

Las acciones de STRIPS se describen en base a dos componentes:

- **Precondiciones**. Es la lista de predicados que deben ser ciertos en un estado para que pueda aplicarse la acción.
- **Efectos**. Es la lista de predicados que cambian en el estado tras la ejecución de la acción. Se pueden subdividir en dos grupos.
    - *Lista de adición* (o predicados afirmados), son los nuevos predicados que se añaden (son ciertos) al estado actual.
    - *Lista de supresión* (o predicados negados), son los predicados que se hacen falsos y por tanto se eliminan del estado actual.

STRIPS además introdujo otra importante idea, el concepto de búsqueda regresiva.

1.3.2. - *Planificación como búsqueda en el espacio de estados*

El problema de planificar se podría definir como un problema de búsqueda en el espacio de estados. Partiendo de un estado inicial, un estado objetivo y el conjunto de acciones disponibles, si representamos cada uno de los estados mediante un nodo en el espacio de búsqueda, y cada una de las acciones como un arco de transición entre los estados, el problema se reduciría a buscar un camino que conecte el estado inicial con el estado objetivo, que sería el plan resultante. Podemos utilizar cualquier algoritmo de búsqueda informado (A*) o no informado (primero en profundidad), muy conocidos en la Inteligencia Artificial, para mejorar la eficiencia del planificador.

El plan se puede obtener como una secuencia de pasos totalmente ordenada. A los planificadores que obtienen los planes como una secuencia totalmente ordenada se los denomina *planificadores de orden total.*

En base a cómo los planificadores realizan la búsqueda podemos clasificarlos en **planificadores regresivos** o **hacia atrás** o bien en **planificadores progresivos o hacia adelante.** STRIPS usaba un algoritmo de búsqueda por regresión en el espacio de estados. Es decir, el algoritmo de búsqueda de STRIPS por tanto busca *hacia atrás* partiendo del objetivo un camino a través de los nodos o estados intermedios para alcanzar el estado inicial.

También se puede usar una estrategia progresiva o de búsqueda hacia adelante partiendo del estado inicial y buscando un camino hasta el estado objetivo (ver Ilustración 17).

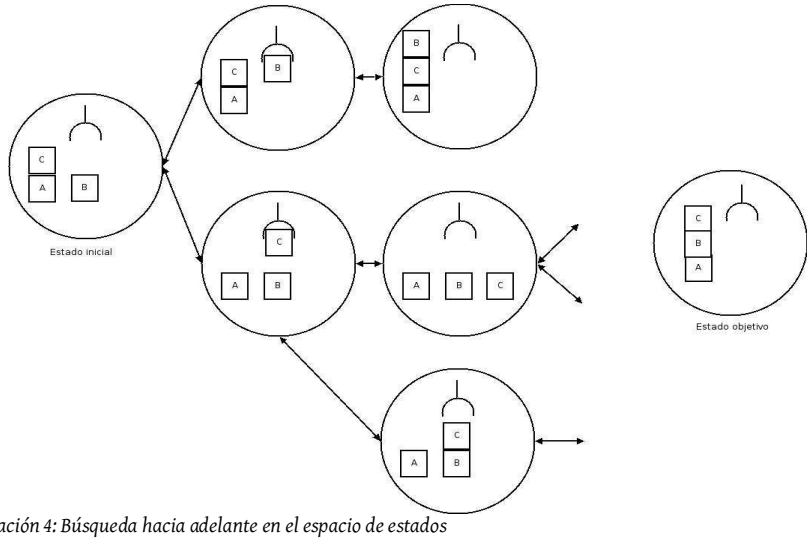


Ilustración 4: Búsqueda hacia adelante en el espacio de estados

Sea:

- ε_i el estado inicial.
- ε_o el estado objetivo.
- $\Pi = (\sigma_1, \sigma_2,...,\sigma_i,...,\sigma_n)$ la secuencia de acciones que conforman el plan en construcción.
- δ el conjunto de acciones en el dominio.
- Υ la función satisfechos(ε_o,ε_i) que calcula: $\forall_j / \zeta_j \in \varepsilon_0 \rightarrow \zeta_j \in \varepsilon_i$

La estructura del algoritmo regresivo usado por STRIPS tiene la siguiente estructura:

regresion(ε_i ,ε_o,Π, δ)

```
1.  if satisfechos(εo,εi) then:
2.    return OK
3.  forall σj in δ do:
4.    if effects(σj) in εo then: //No determinísticamente busca
      una acción a en δ tal que sus efectos satisfacen un
      subconjunto de objetivos de εo
```

```
5.        ε₀' = ε₀ - delete_list(σⱼ) + add_list(σⱼ)
6.        if regresion(εᵢ,ε₀',Π + σⱼ, δ) == OK then:
7.           return OK
8.  return FAIL //Si no quedan más acciones que probar entonces
    devolver fallo (backtrack)
```

Algoritmo 1: Algoritmo de regresión

La principal ventaja de la búsqueda regresiva sobre la búsqueda progresiva o hacia adelante, es que en la búsqueda regresiva el factor de ramificación, o número de acciones aplicables es menor, lo que hace que, en teoría, sea más eficiente. La búsqueda regresiva se basa en la suposición de que el estado objetivo necesita menos literales para describirse que el estado inicial que es un estado "completo".

Sin embargo la búsqueda regresiva presenta otra serie de problemas a la hora de enfrentarse con el manejo de efectos de tipo numérico y con el manejo de recursos [81], debido principalmente a que es más difícil conocer el estado actual y por lo tanto hacer cálculos sobre él o sobre los recursos consumidos en ese momento.

La búsqueda progresiva al partir del estado inicial, completo, resuelve estos problemas mucho más fácilmente. El algoritmo de búsqueda progresiva podría describirse como sigue:

***progresion**(εᵢ ,ε₀,Π, δ)*

```
1.  if satisfechos(ε₀,εᵢ) then:
2.     return OK
3.  forall σⱼ in δ do:
4.     if preconditions(σⱼ) in εᵢ then: //No determinísticamente
       busca una acción  cuyas precondiciones se satisfagan en εᵢ
5.        εᵢ₊₁ = εᵢ - delete_list (σⱼ) + add_list (σⱼ)
6.        if progresion(εᵢ₊₁,ε₀,Π + σⱼ,δ)== OK then:
7.           return OK
8.  return FAIL //Si no quedan más acciones que probar entonces
    devolver fallo
```

Algoritmo 2: Algoritmo de progresión

1.3.3. - *Planificadores neoclásicos*

Cuando el estado objetivo viene representado como un objetivo conjuntivo, el proceso de búsqueda regresivo usado en STRIPS requiere ir tomando cada uno de estos objetivos de forma individual e ir satisfaciéndolos uno a uno.

Cuando dichos objetivos son independientes, es decir, los recursos y agentes que intervienen en los mismos son distintos para cada objetivo, el algoritmo funciona perfectamente. Pero cuando hay interrelaciones entre los mismos, el algoritmo al tratar de cumplir uno de los subobjetivos puede impedir el cumplimiento de otro, cayendo en un bucle infinito (cuando trata de satisfacer un objetivo, pierde el otro, cuando vuelve a tratar de satisfacerlo, pierde el primero y así sucesivamente). A este problema se le conoce como la *anomalía de Sussman.*

Tratando de resolver este problema Sussman y otros investigadores desarrollaron una nueva generación de planificadores. HACKER (1970) [131] fue desarrollado por Sussman para resolver los problemas de incompletitud de STRIPS. En este planificador se introdujo el concepto de *críticos* como una serie de heurísticas que ayudaban a decir el orden correcto en el cual se deben resolver los objetivos para evitar la anomalía de Sussman. HACKER también contaba con una base de datos de heurísticas que se habían probado ser efectivas en ciertas situaciones y que se utilizaban para parchear el plan en curso. Debido a esta forma de planificar hoy en día HACKER es conocido como el antecesor de la *planificación basada en casos*. La planificación basada en casos, trata de obtener planes partiendo de una base de datos de casos previos y una serie de reglas de refinamiento para adaptar el plan a las necesidades del objetivo a alcanzar.

Pocos años después de HACKER, Sacerdoti en su planificador ABSTRIPS (1974) [118] daría origen a un nuevo paradigma de planificación. En ABSTRIPS se introdujo el concepto de jerarquías de acciones abstractas. Sacerdoti trataba con estas jerarquías de mejorar la eficiencia del planificador reduciendo el tamaño del espacio de búsqueda dirigiendo el esfuerzo de planificación primero a aquellas partes más críticas. ABSTRIPS es considerado el primer *planificador jerárquico.*

1.3.4. - *Modelos de planificación que buscan en el espacio de planes*

Se estudiaron otras alternativas diferentes a la forma de abordar el problema de la planificación tratando de evitar el requisito de independencia de los objetivos bajo ciertas condiciones de STRIPS. Con los planificadores NOAH [117] y NONLIN [132] y posteriormente refinada en TWEAK [18] se establecieron las

bases de la planificación como búsqueda en el espacio de planes. Este concepto fue, como casi siempre ocurre en planificación, posteriormente formalizado con los planificadores SNLP [91] y UCPOP [62] .

En el espacio de estados los nodos son los estados por los que va pasando el plan y las transiciones entre los estados o arcos representan acciones que pueden ser aplicadas sobre un determinado nodo, el orden en el que las decisiones son tomadas durante el proceso de búsqueda por el planificador tiene un gran impacto en la eficiencia del planificador, ya que reduce rápidamente el espacio de búsqueda explotando el backtracking, y todavía hoy continúa siendo un área de investigación muy activa [11] [141] [58].

En cambio con la búsqueda en el *espacio de planes* se trata de retrasar la toma de estas decisiones hasta que no quede otra alternativa, apoyándose en la idea de que las decisiones prematuras restringen muy rápidamente el espacio de búsqueda descartando alternativas válidas. En el espacio de planes los nodos representan planes parcialmente especificados, y los arcos son operaciones de refinamiento que pueden ser aplicadas a ese nodo, para pasar de un plan a otro mas detallado. Se sigue una estrategia de *mínimo compromiso* que consiste en retrasar la instanciación de variables y la ordenación entre los operadores hasta que surja alguna restricción que lo imponga.

En el caso de los planificadores que buscan en el espacio de planes, estas restricciones hacen que se añada algún arco de refinamiento entre dos nodos del grafo. Las estructuras que soportan a los planificadores que buscan en el espacio de planes suelen ser mas complejas que aquellos que buscan en el espacio de estados, pero tienen la ventaja de trabajar con una familia de planes de forma simultánea (ver Ilustración 5).

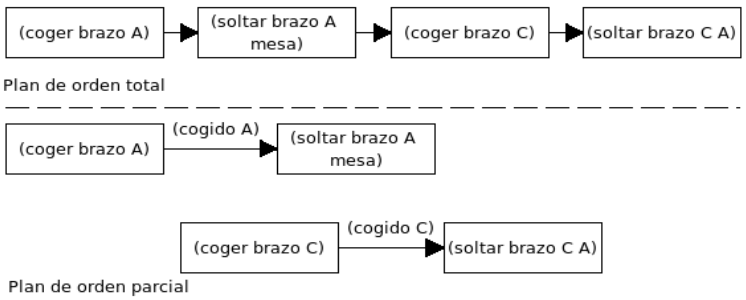


Ilustración 5: Diferencias en la estructura de planificación de orden parcial y orden total. En el plan total existe un único plan en exploración, mientras que en plan parcial pueden existir varios de forma simultánea (coger primero el bloque C o el bloque A)

Podemos representar un plan parcial como una tupla $<\varphi, \theta, \lambda>$ [136] donde φ es un conjunto de acciones, θ es el orden entre las acciones de δ y λ es un conjunto de *vínculos causales* cuyo concepto se explicará a continuación. Los posibles refinamientos que pueden ser aplicados a un plan parcial son los siguientes:

- **Añadir una acción**. El orden de inserción de las nuevas acciones en el plan no tiene por qué mantenerse en θ, es decir dado el plan en construcción mostrado en la Ilustración 5, aunque el orden en el cual las acciones fueron insertadas en el plan sea el que se muestra en la parte superior de la ilustración ((*coger brazo A), (soltar brazo A mesa), (coger brazo C), (soltar brazo C A)*), como se ve en la parte inferior no se añade un orden hasta que sea necesario (otro plan válido podría ser *(coger brazo C), (soltar brazo C A), (coger brazo A), (soltar brazo A mesa)*). Por eso el plan en construcción es un *plan parcial* para diferenciarlo de un *plan de orden total* donde la ordenación entre las acciones está perfectamente definida. De cualquier forma cualquier plan de orden total que se extraiga del plan parcial debe respetar las ordenaciones de θ y los vínculos causales en λ.

- **Añadir una restricción de orden**. A veces es necesario hacer que una acción se ejecute antes con otras. Para realizar esta ordenación se introducen restricciones de orden del tipo $(\sigma_i < \sigma_j)$ en θ que significa que σ_i debe ejecutarse antes de σ_j.

- **Añadir un vínculo causal**. Los vínculos causales [132] son construcciones creadas para evitar inconsistencias en el plan. Los vínculos causales protegen las decisiones hechas con anterioridad de las interferencias producidas al añadir una nueva acción. Los vínculos causales se representan como $\sigma_p \rightarrow^{\zeta} \sigma_c$ donde σ_p es la acción productora de un efecto que es necesario para dar validez de la precondición ζ de la acción consumidora σ_c Decimos que una acción σ_t *amenaza* $\sigma_p \rightarrow^{\zeta} \sigma_c$ cuando alguno de los siguientes criterios se satisface:

  - $\theta \; U \; \sigma_p < \sigma_t < \sigma_c$ es consistente, y
  - σ_t produce $\neg \zeta$ como efecto.

  El planificador no puede entrelazar una acción entre otras dos de forma que produzca una amenaza sobre un vínculo causal, por lo que se debe añadir alguna restricción de ordenación en θ. El orden parcial de los

planes permite evitar la anomalía de Sussman [130] descrita anteriormente. Las restricciones de ordenación impuestas por la estructura de vínculos causales, hacen que las acciones se ejecuten en un orden consistente con sus precondiciones y sus efectos. En el ejemplo de la Ilustración 5 existen amenazas ya que no hay ordenaciones explícitas (coger brazo A) amenaza a la acción (coger brazo C) y viceversa ya que ambas requieren la precondición (libre brazo). Se debe imponer una ordenación entre las acciones que provocará el añadir un nuevo vínculo causal entre las mismas. Una posible solución podría ser el indicado en la Ilustración 6.

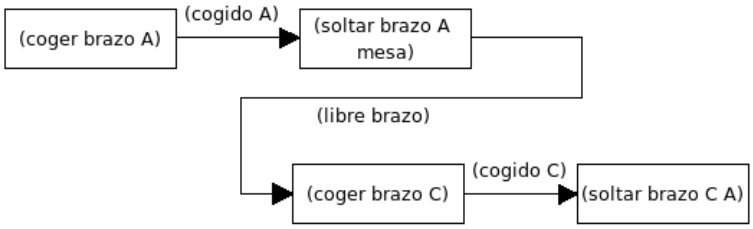


Ilustración 6: Añadir un vínculo causal para resolver las amenazas

- **Añadir una instanciación de variable**. Las acciones son esquemas genéricos sin instanciar con variables como parámetros. Pero esas variables deben ser instanciadas a objetos concretos. El plan final debe estar completamente instanciado para ser correcto. El principio de mínimo compromiso se aplica también a la instanciación de variables que es retrasada hasta que es estrictamente necesario.

El algoritmo básico de orden parcial (POP *Partial Order Planner)* [136], es un algoritmo regresivo que dados:

- δ el conjunto de acciones en el dominio.
- α la agenda de literales pendientes por resolver, que se inicializa con la lista de literales objetivo.
- Y el plan parcial inicialmente vacío <φ,θ,λ> (φ es un conjunto de acciones, θ es el orden entre las acciones de δ y λ es la estructura de vínculos casuales).

Puede ser descrito de la siguiente forma:

POP (<φ,θ,λ>,δ,α)

1. Terminación:
   Si la agenda está vacía empty(α), devolver <φ,θ,λ>

2. Selección de objetivos:
   Escoger de la agenda α una pareja <ζ,σ_{need}>, donde ζ es una condición que σ_{need} necesita hacer cierta para poder aplicarla.

3. Selección de operadores:
   Escoger un operador σ_{add} que proporciona ζ.
   El operador puede ser seleccionado de las acciones ya presentes en el plan o instanciando una nueva acción del dominio δ. Si no existe tal acción devolver fallo.
   Sea:
   $\lambda' = \lambda \cup \{\sigma_{add} \rightarrow^{\zeta} \sigma_{need}\}$
   $\theta' = \theta \cup \{\sigma_{add} < \sigma_{need}\}$
   Si σ_{add} es un operador nuevo recién instanciado entonces:
   $\varphi' = \varphi \cup \{\sigma_{add}\}$
   $\theta' = \theta \cup \{\sigma_O < \sigma_{add} < \sigma_\infty\}$
   En otro caso:
   $\varphi' = \varphi$

4. Actualizar los objetivos:
   Sea:
   $\alpha' = \alpha - \{\langle \zeta, \sigma_{need} \rangle\}$
   Si σ_{add} es una tarea recién instanciada entonces:
   $\forall_{\zeta_i / \zeta_i \, es\ una\ conjunción \in precontitions(\sigma_{add})} \rightarrow \alpha' = \alpha' \cup \langle \zeta_i, \sigma_{add} \rangle$

5. Protección de los vínculos causales:
   Para todo operador σ_t que tenga un vínculo causal amenazado $\sigma_p \rightarrow^{R} \sigma_t \in \lambda'$ escoger una restricción de ordenación válida, que puede ser:
   — Democión: $\theta' = \theta' \cup \{\sigma_t < \sigma_p\}$
   — Promoción: $\theta' = \theta' \cup \{\sigma_c < \sigma_t\}$
   Si ninguna es válida devolver fallo.

6. Llamar recursivamente:
   return POP (<φ',θ',λ'>,δ,α')

Algoritmo 3: Partial Order Planner

Basados en el modelo de planificación de orden parcial hay una gran cantidad de planificadores (*Partial Order Causal Link Planners POCL*), el más

representativo y del que de alguna u otra manera se derivan el resto de planificadores es UCPOP [62]. VHPOP [141] es un algoritmo POP más reciente que destacó en la IPC[1] del 2002. Incluye la posibilidad de incluir distintas estrategias para el cálculo del costo de cada decisión con respecto a la cercanía a la solución final y al coste de la reparación en caso de fallo [108] [126]. También incluye una red de restricciones temporales STN (*Simple Temporal Network)* que le permite hacer ciertos cálculos sobre la duración de las acciones.

1.3.5. - *Planificadores basados en el análisis de grafos*

Basándose en los mismos principios y suposiciones que los planificadores clásicos, en los últimos años han aparecido nuevas técnicas de búsqueda, las más importantes son la planificación basada en grafos y la planificación SAT.

El primer planificador que usó técnicas de construcción y búsqueda en grafos para resolver problemas de planificación fue GRAPHPLAN [3] [116]. GRAPHPLAN construye un grafo acíclico distribuido en dos tipos de capas con distintos nodos. Las capas pares contienen nodos compuestos de proposiciones. La capa 0 contiene todas las proposiciones del estado inicial. Las capas impares contienen nodos compuestos de acciones. El algoritmo construye las distintas capas de la siguiente forma:

- Sea γ_i la capa i-ésima y suponiendo que la capa inicial $\gamma_0 = \varepsilon_o$.
- ε_g es el estado objetivo.

construir_capa $((\gamma_0, \gamma_1, .., \gamma_{i-1}), i, \varepsilon_g)$

1. **Terminación:**
   Si i−1 es par y γ_{i-1} contiene todos los literales del estado objetivo ε_g entonces terminar.

2. **Construcción de capas:**
   en otro caso:
   - Si i−i es impar entonces construir γ_i a partir de los literales que resultan de aplicar los efectos de todas las acciones de γ_{i-1}.

1 IPC (International Planning Competition). La Competición Internacional de Planificadores es un evento bianual, que se realiza junto a la Conferencia Internacional de Planificación y Secuenciación ICAPS *(International Conference of Planning and Scheduling)*. Los objetivos de esta competición son el proporcionar una base para el debate y la comparación de los sistemas de planificación, en forma de problemas justo en la frontera de las capacidades de los planificadores actuales para proponer nuevas direcciones de investigación, además de proporcionar un conjunto de problemas con una representación común que permiten la comparación y evaluación de los distintos sistemas de planificación.

	—Si i–i es par entonces construir γ_i con todas las acciones cuyas precondiciones son satisfechas usando los literales de γ_{i-1}.
3.	**Corregir exclusiones:** 1. Dos acciones son excluyentes si: —Una de ellas borra una precondición o un efecto de la lista de adición de la otra. —Hay una precondición en una de las acciones y otra precondición de la otra que están marcadas como excluyentes. 2. Dos proposiciones ζ_p, ζ_q son excluyentes si cada acción que consigue ζ_p es excluyente con cualquier acción que consigue ζ_q, es decir toda forma de crear ζ_p es mutuamente excluyente con toda forma de conseguir ζ_q.
4.	**Recursión:** construir_capa((γ_0, γ_1,.., γ_{i-1})+γ_i,i+1 ,ε_g)

Algoritmo 4: GRAPHPLAN

Una vez que se ha construido el grafo, se realiza una búsqueda regresiva sobre el mismo tratando de encontrar un camino (o plan válido), teniendo en cuenta las restricciones de exclusión mutua impuestas en cada nivel.

Las técnicas de búsqueda en grafos han sido objeto de intenso estudio durante los últimos años debido al éxito en velocidad de respuesta y capacidad de resolver problemas que mostraron durante la Competición Internacional de Planificadores IPC del año 1998. Hay una gran cantidad de sucesores de GRAPHPLAN entre los que cabría destacar STAN [82] que realizaba un preprocesamiento del dominio para eliminar acciones y proposiciones y que utilizaba unas estructuras de datos más avanzadas que las de GRAPHPLAN, SGP [137] que maneja incertidumbre, TGP [123] que permite que las acciones tengan duración, IPP [68] que maneja recursos en las acciones, o TPSYS [45] que añade otra fase más al algoritmo básico de GRAPHPLAN, en la cual se calcula un grafo temporal con las capas ordenadas cronológicamente por el instante temporal, que le permite calcular planes temporizados.

1.3.6. - *Planificadores heurísticos*

Otra forma de utilizar los planificadores que buscan en grafos es para construir heurísticas que guíen a los planificadores heurísticos como en el planificador FF [58] o AltAlt [129] (a estos también se los conoce como

planificadores híbridos). En general los planificadores heurísitcos transforman un problema de planificación en un problema de búsqueda heurística [11]. Las heurísiticas se construyen relajando el problema de planificación, por ejemplo una heurística muy utilizada consiste en ignorar las listas de supresión en un modelo de acciones tipo STRIPS. La investigación en este tipo de planificadores se centra en encontrar mejores heurísticas admisibles para mejorar la eficiencia. Algunos de los planificadores heurísticos más destacados son HSP [11], HSP2 [12], Sapa [89], FF [58] o AltAlt [129].

1.3.7. - *Planificación como un problema de satisfacibilidad*

Otra forma de transformar un problema de planificación es convertirlo en un problema de satisfacibilidad que pueda ser resuelto por algoritmos estocásticos eficientes y bien conocidos. Por ejemplo los planificadores SATPLAN [2] [10] o LPG [48] usan esta estrategia.

1.3.8. - *Planificación como un problema de chequeo de modelos.*

Una última trasformación del problema de planificación consiste en convertirlo en una fórmula lógica para la cual tenemos que buscar un modelo adecuado. Este tipo de aproximación es llamada *chequeo de modelos (model checking)* [19].

1.3.9. - *Planificación jerárquica*

Los modelos de planificación estudiados hasta el momento se centran en resolver problemas en dominios "planos" con un único nivel de abstracción. Existe un gran interés investigador en los últimos años en este tipo de problemas, debido en gran parte a la IPC, en donde los problemas a resolver son todos sobre dominios planos. En la planificación heurística, de búsqueda en grafos, SAT y *model checking,* se obtienen cada vez planificadores más eficientes y más expresivos. Sin embargo, parece que la expresividad que se puede alcanzar a la hora de describir dominios aceptables por este tipo de planificadores sigue siendo limitada. Existe una creciente preocupación en la comunidad investigadora en acercar más la planificación a las necesidades reales de los posibles usuarios finales y para ello se necesita una mejor gestión del conocimiento. Muestra de esta preocupación es la aparición de una nueva competición en el 2005 ICKPES *International Competition on*

Knowledge Engineering for Planning and Scheduling (que se repetirá en el 2007) donde se premia las plataformas basadas en un planificador que hagan una mejor gestión y uso del conocimiento.

Representar el conocimiento del dominio para poder planificar sobre él, es una tarea compleja y delicada, debido a que afecta al número de problemas que el planificador pueda resolver, así como a la velocidad a la que es capaz de resolver los mismos. La expresividad del modelo de acciones de STRIPS es en este sentido además bastante limitada ya que no permite la abstracción de conocimiento. Para tratar de solucionar este problema se han desarrollado nuevas técnicas de representación de conocimiento que permiten su abstracción. La abstracción se puede aplicar considerando distintas perspectivas:

- **A nivel de literal**, esta abstracción permite definir abstracciones de literales de un nivel superior, mediante la composición de los literales de su nivel inmediatamente inferior. Esta abstracción fué usada por primera vez en el planificador ABSTRIPS [118]. Esta abstracción es sencilla en su concepción y además rápida en su procesamiento. De hecho se puede preprocesar un dominio con abstracciones de literales y reducirlo a otro dominio en donde todos los literales sean del nivel más básico [66]. Sin embargo, aunque esta abstracción puede simplificar la escritura de dominios, no supone un salto cualitativo notable en cuanto al modelo de STRIPS.

- **A nivel de acción**, que permite construir abstracciones o jerarquías de acciones. Existe una amplia gama de algoritmos para trabajar sobre redes jerárquicas de tareas (*Hierarchical Task Networks* o HTN) [33] [97] [28]. En este modelo además los objetivos se representan de una manera distinta al modelo tradicional de STRIPS. Los objetivos son redes de tareas a resolver (en contra de estados que alcanzar), donde la red objetivo puede tener acciones u operadores de bajo nivel, o acciones abstractas compuestas de alto nivel.

Otra forma de abstracción es la mostrada en HYBIS [17]. HYBIS es un planificador jerárquico híbrido en donde la abstracción se lleva a cabo a nivel de agentes. Un dominio se describe como una jerarquía composicional de agentes, donde los agentes hoja son los que finalmente actúan. Cada agente puede ejecutar distintas acciones representadas mediante autómatas finitos. La estructura de las acciones se generaliza también a través de la jerarquía de agentes a los que las acciones van asociadas.

1.3.10. - *Planificación de redes de tareas jerárquicas*

La planificación de redes de tareas jerárquicas o HTN, para simplificar de sus siglas en inglés (*Hierarchical Task Networks*) tiene su origen en la necesidad de acercar el modelo de representación de acciones a la representación con distintos niveles de abstracción que el ingeniero del conocimiento utiliza a la hora de modelar un dominio. De hecho está demostrado que la expresividad del modelo HTN es mayor que la del modelo de STRIPS [34].

Aunque los orígenes de la planificación jerárquica son antiguos [118] [132], incluso ya existiendo algunas aplicaciones notables [29], como por otro lado ocurre habitualmente en el área de planificación, las bases teóricas no fueron establecidas hasta algún tiempo después [33]. Aún así todavía hoy en día no hay un modelo común y unificado, aunque todos comparten una serie de características muy similares.

Cada planificador HTN tiene sus propias peculiaridades y su propio lenguaje para la descripción de dominios. Incluso en la IPC no hay todavía una sección exclusiva dedicada a planificadores HTN. Ha habido no obstante esfuerzos e intentos notables para desarrollar un lenguaje de representación de dominios unificado [81] [40], sin que hayan tenido éxito debido principalmente a esta heterogeneidad y complejidad en los sistemas.

Otra razón debido a la cual los planificadores HTN han tenido tanto éxito es debido a su eficiencia al enfrentarse a problemas reales. La expresividad del HTN y la forma en que este modelo realiza la búsqueda, permiten que el diseñador del dominio tenga un mayor control sobre el proceso de búsqueda, pudiendo este optimizar el dominio para evitar realizar búsqueda innecesaria.

El modelo de HTN que se explica está basado en el modelo de UMCP (*Universal Method Composition Planner*) [33]. En este modelo encontramos los siguientes elementos:

- **Acciones primitivas u operadores primitivos, base u hoja:** Los operadores primitivos son muy similares a las acciones usadas en el modelo de STRIPS. Son las acciones que finalmente son ejecutadas y alteran los literales del estado.

- **Tareas o acciones compuestas o abstractas:** Una tarea abstracta requiere para su descomposición a un nivel de abstracción inferior, que exista un *esquema de reducción de acciones,* que básicamente es un conjunto de otras tareas con unas determinadas relaciones de orden, por las cuales la tarea abstracta se sustituye. Estas tareas forman una red de tareas, que no es más que un grafo dirigido acíclico que puede estar compuesto de otras abstractas, operadores primitivos, o bien una mezcla de ambos.

- **Métodos:** Una tarea abstracta puede tener varios esquemas de reducción distintos. Para decidir que esquema de reducción es aplicable en cada momento o es el más adecuado se introduce el concepto de método, que establece una serie de condiciones previas a la aplicación de la reducción.

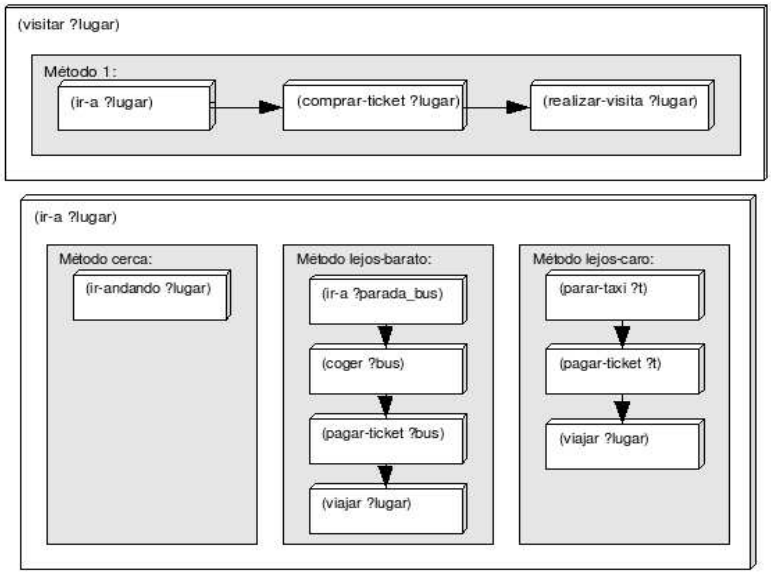


Ilustración 7: Ejemplo de descomposición HTN

Esta forma de organizar el conocimiento procedural en base a tareas de más alto y más bajo nivel, es mucho más cercana a la forma de planificar de los seres humanos, que solemos marcar una serie de tareas generales que es necesario realizar y las vamos refinando conforme vamos definiendo el plan.

Como se ha señalado, el modelo de planificación HTN es similar al modelo de STRIPS [37] en algunos aspectos. Primero ambos tienen una representación común del espacio de estados. Por otro lado la mayoría de los planificadores HTN también presuponen un conocimiento completo del entorno. Y finalmente la representación de los operadores primitivos es similar, con unas precondiciones, y unos efectos, que en STRIPS están divididos en una lista de adición y una lista de supresión. La principal diferencia radica en qué se considera un objetivo y cómo se consigue. Para STRIPS un objetivo es alcanzar un estado meta, en HTN el objetivo es aplicar los esquemas de reducción sobre todas las tareas abstractas de la red de

tareas de partida, hasta tener una red de tareas válida, únicamente compuesta de operadores primitivos, que será finalmente el plan resultante.

Supongamos el ejemplo de la figura 7 en el que se muestra una tarea abstracta para visitar un museo. Esta tarea tiene un único método que establece cual es la red por la que sustituimos la tarea abstracta, que está compuesta de tres tareas que se realizan en secuencia. Primero debemos desplazarnos al lugar de visita, después hay que adquirir un ticket de entrada y por último realizamos la visita. En el ejemplo también se muestra la estructura de la tarea abstracta ir-a, que dispone de tres posibles métodos de descomposición. Cada uno de estos métodos puede llevar una serie de precondiciones asociadas para seleccionar en cada momento cual es la descomposición más adecuada. Por ejemplo si el lugar de destino está a una distancia menor de un umbral predeterminado se puede ir andando, en otro caso se puede coger un taxi o un autobús. Observar que para coger el autobús previamente hay que desplazarse hasta la parada, para lo cual podemos aplicar el mismo método ir-a. Cualquiera del resto de tareas que no se muestran en el ejemplo como pagar-ticket o viajar, pueden ser otras tareas abstractas o operadores primitivos. El planificador va construyendo mediante la sustitución de tareas abstractas por su correspondiente red de tareas, un *árbol de expansión de tareas* que constituye su espacio de búsqueda, distinto del espacio de búsqueda de STRIPS que es un grafo de estados con transiciones de uno a otro estado, y también del espacio de búsqueda de planes de UCPOP [136] ya que las tareas son introducidas en el plan candidato siguiendo una filosofía distinta. El árbol de expansión de tareas puede verse como un árbol Y/O donde los nodos O son las distintas métodos alternativos que tenemos para expandir una tarea compuesta, y los nodos Y serían las distintas tareas codificadas en la red de tareas que es necesario expandir.

Las tareas abstractas permiten al diseñador de dominios escribir el dominio con diferentes y múltiples niveles de abstracción, pudiendo utilizar un enfoque top-down en donde partimos de tareas muy abstractas y las vamos especificando cada vez mas en tareas de más bajo nivel o bien un enfoque bottom-up en donde partimos de los operadores primitivos y vamos construyendo las tareas abstractas que lo asocian.

Así pues se puede describir el algoritmo de descomposición de tareas HTN muy básico (hacia adelante y con estados) de la siguiente forma [33]:

- Sea ε_0 el estado inicial.
- $\mathcal{D}$: el dominio de planificación.
- Γ_0: la red de tareas inicial que deseamos descomponer.

- π: el plan en construcción que es una lista de tareas primitivas, inicialmente vacío.

- Y α_i la agenda de tareas pendientes que inicialmente se construye de la siguiente forma: $\alpha_0 \equiv \cup_{i\in(1..n)}\ t_i \in \Gamma_o$, y que se utiliza en la llamada al primer ciclo HTN.

```
HTN(<ε_i, Γ_i, π_i, α_i>, D)
1.  if empty(α_i) then: return π_i
2.  while escoger no determinísticamente una tarea: ρ∈α_i
3.     if primitive(ρ) then:
4.         if preconditions(ρ, ε_i) == true:
5.             ε_i+1 = ε_i + add_list(ρ) - delete_list(ρ)
6.             π_i+1 = π_i + ρ
7.             return HTN(<ε_i+1, Γ_i, π_i+1, α_i>, D)
8.          else: backtrack
9.      elif compound(ρ) then:
10.         Escoger un método μ válido o backtrack
11.         Γ_i+1 = Γ_i - ρ + expand(ρ, μ)
12.         α_i+1 = update(α_i , Γ_i+1)
13.         return HTN(<ε_i, Γ_i+1, π_i, α_i+1>, D)
14. return FAIL
```

Algoritmo 5: HTN básico

Cuando la expansión de una tarea abstracta falla, por que alguna de las acciones que componen la red de tareas en la que se descompone no se puede aplicar se produce un backtracking o vuelta atrás, en la cual el algoritmo de planificación debe explorar otras alternativas (otros métodos o otra forma de unificar las variables). El mecanismo de bactracking es el que permite al planificador adaptar el plan al estado del entorno, pero al mismo tiempo es lo que consume más tiempo y recursos durante la planificación. Todos los planificadores jerárquicos y no jerárquicos tratan de encontrar formas de evitar el backtracking, pero quizá en los planificadores jerárquicos es más sencillo [98] .

Ya en ABSTRIPS [118] que se basa en la abstracción de literales se usaban heurísticas para determinar el nivel de criticidad de cada predicado abstracto, tratando de resolver primero aquellos predicados más críticos y evitar backtracing.

Este mecanismo se mejoró en el planificador ALPINE [66] o con ABTWEAK [138] que usaba el planificador TWEAK [18] en lugar de STRIPS [37].

Los métodos de descomposición de las tareas compuestas μ también pueden utilizarse como mecanismo para mejorar la eficiencia del planificador [132] [139]. Desde los primeros planificadores HTN se han usado *críticos* como los usados por primera vez en el planificador NOAH [117] para imponer restricciones sobre los métodos. Los críticos detectan posibles interacciones entre las expansiones de dos métodos que lleven a una inconsistencia y proponen soluciones reduciendo por lo tanto la cantidad de backtracking necesaria hasta la recuperación. En planificadores HTN más modernos como por ejemplo SHOP2 [28] no se usan críticos de una manera explícita y las interacciones son previstas entre las precondiciones impuestas en los métodos y las restricciones de la propia red de tareas a expandir.

1.4. - Planificadores con conocimiento incompleto

En la planificación clásica se adopta como suposición básica para trabajar que el conocimiento del que dispone el planificador es completo y además correcto. Desafortunadamente, en problemas reales casi nunca suelen darse estas condiciones. Es imposible que el planificador tenga conocimiento completo esto solo ocurren en sistemas totalmente cerrados sin posibilidad de interferencia externa y donde se garantiza que todo funciona como lo esperado (un juego con un solo jugador y sin azar por ejemplo). Siempre pueden ocurrir situaciones imprevistas por el planificador (a los humanos nos ocurre lo mismo de ahí la dificultad de los problemas de planificación). Además los efectos de las acciones adoptadas por el planificador tampoco tienen porque ser los esperados. Existen una gran cantidad de planificadores que tratan de resolver estos problemas manejando de una forma u otra la incertidumbre, todos ellos deben observar o suponer los efectos de sus acciones durante el tiempo de ejecución para evaluar el resultado de las mismas. Dependiendo de las posibilidades de detección que tengan los sistemas planificadores los podemos clasificar en varios grupos.

1.4.1. - *Sistemas con observabilidad total*

Al igual que los planificadores clásicos estos sistemas conocen en cada momento la situación de su entorno y además pueden ver los efectos de sus acciones. Sin embargo también parten de la suposición de que el plan previsto puede fallar en cualquier momento. Existen varios modelos que explotan esta información.

1.4.1.1. - *Planificadores reactivos*

En este modelo la planificación y la ejecución del plan están entrelazadas [46] [119]. Este tipo de planificación es muy común en la naturaleza (muchos seres vivos actúan así) y es muy usado en el ámbito de la robótica (permite respuestas reflejas del robot muy rápidas). El individuo tiene una serie de objetivos o deseos que le empujan a tener un determinado comportamiento (hambre, sed, frío) que es capaz de sensorizar, al igual que su entorno. Además dispone de una serie de reglas, o de conocimiento instintivo, de acciones a tomar ante determinadas circunstancias. El sistema realiza de forma automática las acciones codificadas en su base de conocimiento que responden ante los estímulos observados en su entorno.

1.4.1.2. - *Planificadores que manejan la incertidumbre*

Cuando tenemos información sobre el comportamiento del entorno, podemos elaborar estrategias más inteligentes o a más largo plazo que simplemente reaccionar. Podemos intentar prever las posibles causas de fallo e intentar establecer un plan que las evite en lo posible. Existen diferentes aproximaciones a considerar. Se puede usar una distribución de probabilidad sobre los estados y buscar un plan que maximice la probabilidad de éxito [95] [24]. O usar teoría de la posibilidad [16]. Bonet y Geffner proponen usar un planificador heurístico tipo HSP para buscar en el espacio de creencias donde los estados tienen asociada una distribución de probabilidad [55]. O también se pueden convertir los problemas de planificación en MDPs que se pueden resolver mediante teoría de la decisión [115].

1.4.1.3. - *Reparación de planes*

La *replanificación* o *reparación de planes* [23] [25] [26] es una técnica que aprovecha la observabilidad del entorno para actuar ante los cambios producidos en el entorno por otros individuos o variables externas no controlables. Se parte de la suposición de que la mayor parte del plan actualmente calculado es válido por lo que es una pérdida de tiempo calcularlo de nuevo. De esta forma se va adaptando o reparando el plan ante nuevas contingencias no previstas sin que además los cambios en el plan sean radicales. Puede sin embargo que en determinadas situaciones esta suposición no sea válida y el costo de replanificar sea incluso mayor que el de planificar desde cero [99]. Otra ventaja de la replanificación es que generalmente se pueden adaptar las técnicas utilizadas para realizar la planificación a la hora de realizar la reparación.

1.4.1.4. - *Planificación continua*

La planificación continua o *contiual planning* [7] [78] [96] [64] [42] trata de continuar ejecutando aquellas partes del plan que no están dañadas al mismo tiempo que repara o reconsidera el resto del plan. En la planificación continua generalmente no se planifica demasiado por antelación ya que se asume que el plan puede fallar en cualquier momento, por lo que no se malgastan recursos en planificar a muy largo plazo. Además el planificador puede utilizar la información que se recoge en tiempo de ejecución para mejorar los planes. Se toma un enfoque determinista se planifica suponiendo que todo va bien pero cuando algo falla se lanza el proceso de reparación. Existen varias alternativas en este modelo. Primero que la planificación y la ejecución se ejecuten de forma alternada, de forma que si es necesario replanificar la ejecución se para. Segundo que la planificación y la ejecución se ejecuten de forma paralela, si hay un fallo se continúan ejecutando aquellas partes del plan que no se vean afectadas hasta que el plan esté completamente reparado.

1.4.1.5. - *Planificación contingente*

La planificación contingente establece un modelo más deliberativo que realiza toda una planificación previa y tratan de establecer de antemano ramas de actuación alternativas a tomar en caso de contingencia pudiendo introducir acciones de sensorización dentro del plan para obtener más información [135] [109] [86] [78]. Cuando se ejecuta el plan, se detectan los posibles fallos mediante las operaciones de sensorización si se dispone de ellas y se escoge de acuerdo al estado actual, la rama apropiada a seguir, disponiendo de esta forma de un *plan condicional*. Si no disponemos de operaciones de sensorización entonces se escoge de antemano la rama con más probabilidad de éxito. El principal inconveniente de este enfoque es la complejidad computacional y el reducido número de problemas que pueden afrontar debido a que no es siempre posible predecir todas las posibles contingencias.

1.4.2. - *Sistemas con observabilidad parcial*

Estos sistemas [7] [52] [104] no disponen de forma inmediata de la información de su entorno, o simplemente no pueden conocer completamente el resultado de sus operaciones, por tanto deben de disponer de alguna forma alternativa para poder llevar a cabo un comportamiento correcto. Para realizar la planificación en este tipo de entorno se pueden utilizar técnicas de reparación, planificación y ejecución entrelazadas, planificación contingente o planificación

continúa igual que los estudiados. La novedad radica en que los fallos no proceden únicamente del entorno, sino que también son provocados por nuestros sensores que son incapaces de recoger toda la información del entorno, y por nuestros efectores que son incapaces de asegurarnos la correcta ejecución de las acciones.

1.4.3. - *Sistemas sin observabilidad*

Estos sistemas se basan en modelos probabilísticos o posibilísticos [95] [24] [52] que tratan de maximizar la efectividad del plan final analizando los posibles distintos cursos de actuación. Este tipo de planificación se denomina ***conformant planning*** [19] [124]. No hay información en tiempo de ejecución, con lo cual:

- No conocemos el resultado de la ejecución de una acción.
- No conocemos al completo el estado inicial.
- Tampoco disponemos de operaciones de sensorización para testear el estado de nuestro entorno.

Los modelos probabilísticos y posibilísticos tratan de analizar el mayor número posibles de alternativas, tratando de encontrar el plan óptimo que no falla (su probabilidad es uno), el plan más óptimo, o bien puesto que el espacio de búsqueda puede ser inabordable, un plan cuya certeza o probabilidad sea suficiente.

1.5. - Lenguajes de representación de dominios

Un aspecto muy importante para cualquier modelo de planificación es definir cómo el algoritmo es informado de la situación en la que se encuentra su entorno y del problema que tiene que resolver. Cómo ya se ha indicado la mayoría de los algoritmos de planificación son independientes del dominio, es decir, son genéricos y en principio podrían utilizarse en cualquier dominio planificación, por lo tanto se hace necesario disponer de un lenguaje de representación de dominios que el planificador pueda procesar.

El primer lenguaje de descripción de dominios y problemas ampliamente aceptado fue el de STRIPS [37]. En STRIPS los problemas de planificación se especificaban en base a:

- Un conjunto de operadores o acciones que el planificador podía usar para resolver el problema. Las acciones estaban compuestas de unas precondiciones en forma de cláusula conjuntiva y unos efectos divididos en una lista de adición y una lista de supresión. Los literales usados para

definir las acciones podían tener variables sin instanciar que el planificador unificaba en tiempo de planificación. De esta forma las acciones eran genéricas, ya que podían operar sobre conjuntos de literales distintos.

- El estado inicial. Que se definía en base a un conjunto de literales que eran ciertos al inicio de la planificación (una cláusula conjuntiva).

- El objetivo o problema. También compuesto de una serie de literales que el planificador debía hacer ciertos al terminar el plan.

El plan devuelto es una lista de acciones ordenadas y cuyas variables se encuentran completamente instanciadas.

El lenguaje de STRIPS permite representar una cantidad de dominios y problemas bastante compleja, pero su expresividad en algunos casos puede no ser suficiente. Por ello a lo largo del tiempo han ido apareciendo lenguajes sucesores de STRIPS que aumentan su expresividad.

ADL [106] (*Action Description Language*) es una ampliación de STRIPS que incluía las siguientes mejoras en cuanto a expresividad:

- Posibilidad de incluir literales negados como objetivo del plan así como en las precondiciones de las acciones.

- Objetos del dominio y variables en los literales tipados. De esta forma los objetos de dominio que pueden unificar con las variables sin instanciar de un literal de una precondición están más limitados.

- Cláusulas disyuntivas, cuantificadas universalmente y existencialmente en las precondiciones de una acción.

- Inclusión de un operador de igualdad expresado como un literal que permite la comparación de variables dentro de las precondiciones de una acción.

- Y por último efectos condicionados que sólo se aplican cuando el estado del mundo cumple con una determinada condición. Los efectos condicionales permiten resumir varias acciones con una estructura de precondiciones y efectos muy similar en una sola.

Obviamente a medida que crece el grado de expresividad del lenguaje de representación también crece la complejidad de los algoritmos de planificación que lo procesan. ADL es un lenguaje muy utilizado, pero si hay un lenguaje que merece ser considerado un estándar, este es sin duda PDDL (*Planning Domain Description Language)* [49]. PDDL nace en 1998 como lenguaje para describir los problemas de la IPC de ese mismo año. Tiene en su haber dos grandes ventajas que

lo han hecho tan usado. Primera, es el lenguaje usado en la IPC, cualquier planificador que quiera participar en esta competición, que por otro lado es bastante prestigiosa, debe ser capaz de procesarlo. Segundo y no menos importante, al ser muchos los autores que colaboraron en su especificación, supo unificar las características principales de los lenguajes de los planificadores más importantes en la época. Su base fue el lenguaje ADL [106], pero también recogió formalismos de plataformas de planificación como PRODIGY [134] o SIPE-2 [25] [26], planificadores de orden parcial como UCPOP [136], Unpop [92], o jerárquicos como UMCP [33], aunque el modelo jerárquico propuesto en PDDL no ha sido muy aceptado como discutiremos más adelante.

PDDL es un lenguaje muy expresivo, de hecho hay muchos planificadores que no pueden soportarlo plenamente, por ello incorpora un mecanismo de requerimientos sobre el planificador, como una serie de flags, que el planificador debe soportar para operar sobre el dominio.

En la versión 1.0 de PDDL [49] su expresividad era muy similar a la de ADL con la incorporación de la posibilidad de definir dominios jerárquicos. Afortunadamente PDDL es un lenguaje vivo que va evolucionando con cada edición de la IPC. En la edición del 2002 se presentó una nueva versión del lenguaje la 2.1 [81] que incorporaba importantes novedades. PDDL 2.1 se organizaba en 4 niveles con expresividad creciente. En el nivel 1 tenía la misma expresividad que PDDL 1.0. En el nivel 2 se incluye una mayor expresividad para el tratamiento de expresiones numéricas o *fluents.* En el nivel 3 se introduce temporización en las acciones, con precondiciones y efectos at-start, at-end o overall. El nivel 4 permite la definición de acciones con efectos continuos en el tiempo. La última versión oficial de PDDL es la 3.0 [47] presentada en la IPC del 2006, esta versión recoge la posibilidad de introducir preferencias de usuario y restricciones en el dominio de planificación.

Existen otras extensiones de PDDL no presentadas en la IPC. Por ejemplo hay otras propuestas para dar soporte jerárquico a HTN [127], o la propuesta de un quinto nivel [41] para modelar procesos continuos que van cambiando con el paso tiempo los valores numéricos.

1.6. - Redes de restricciones temporales

El modelo subyacente a la mayoría de los planificadores que son capaces de operar con el tiempo son las redes de restricciones temporales TCN (*Time Constraint Networks*) [32], que es un tipo especial de red de restricciones (*Constraint Network*) [110]. Una red de restricciones CN puede ser vista como una tripleta (X,D,C), donde $X=\{x_1,...,x_n\}$ es un conjunto finito de variables, $D=\{d_1,...,d_n\}$ es el conjunto de valores que pueden tomar esas variables o dominio y $C=\{c_1,...,c_n\}$ es un

conjunto de restricciones. Las restricciones definen los valores del dominio que las variables pueden tomar de forma simultánea. En el caso de una red de restricciones temporales las variables representarán puntos temporales que marcan el inicio o el fin de un evento o intervalos de tiempo durante el cual el evento ocurre. El dominio corresponde a los posibles instantes de tiempo que pueden tomar esas variables y las restricciones pueden ser de dos tipos:

- Restricciones cualitativas: Son las restricciones que se imponen sobre la posición relativa de un par de puntos.
- Restricciones cuantitativas: Restringen la distancia temporal entre un par de puntos o bien enmarcan un punto en un instante concreto.

Un Problema de satisfacción de restricciones CSP (*Constraint Satisfaction Problem*) consiste en encontrar una una instanciación de las variables con los valores del dominio, de forma que todas las restricciones se cumplan. Un TCSP (*Temporal Constraint Satisfaction Problem*) es el problema más específico de encontrar la solución en una red de restricciones temporales.

Cualquier sistema que haga razonamientos con el tiempo, necesita los siguientes elementos:

- **Una base de conocimiento temporal**. Compuesta de proposiciones (en el caso de los planificadores literales y acciones) que son válidas en unos determinados instantes de tiempo y sobre las cuales se construirá la TCN. El intervalo temporal viene fijado por dos puntos el inicio y el fin. Notaremos como t_s^p el punto temporal en el cual comienza la proposición p y t_e^p será el punto temporal en el que esta finaliza. Con esta información temporal se pueden elaborar expresiones temporales más complejas. Por ejemplo se puede decir que la duración de una proposición *p* es 5 unidades temporales de la siguiente forma: $t_e^p - t_s^p=5$. O que una proposición *p* comienza a las 3 unidades temporales como: $t_s^p=3$. De la misma forma se pueden establecer relaciones (restricciones cualitativas) entre dos proposiciones distintas. Por ejemplo dadas dos proposiciones *p* y *q* se puede afirmar que *q* comienza justo después de *p* de la siguiente forma: $t_e^p=t_s^q$. De esta forma los puntos temporales pueden tomar valores absolutos o relativos a otros puntos temporales. Cada expresión temporal que describamos añade una o más restricciones temporales que se van almacenando en una red de restricciones temporales.
- **Un algoritmo de chequeo de consistencia**. Cada vez que se añade una restricción a la red temporal es posible que sea incompatible con una restricción previamente añadida. Es necesario conocer si la TCN sigue siendo consistente. Un ejemplo muy simple e inmediato de violación

de restricciones sería tener el siguiente conjunto de restricciones $\{t_e^p - t_s^p=5, t_s^p=0\}$ y tratar de añadir la restricción $t_e^p=6$.

- **Un mecanismo para hacer consultas.** Cuando se dispone de una TCN queremos además obtener información acerca de la relación temporal existente entre cualquier par de proposiciones. Se necesita un algoritmo de inferencia para realizar este proceso.

- **Un algoritmo de propagación de las restricciones**. Cada vez que se añade una nueva restricción a la STN, esta no solo afecta al par de variables involucradas en la restricción, sino que se puede provocar afectando a más relaciones entre variables dentro de la red. Normalmente siempre que se añade una restricción lo que se consigue es acotar el conjunto de variables del dominio (instantes de tiempo) que puede tomar una variable (punto temporal). Se pueden encontrar nuevas restricciones utilizando inferencia sobre la red de tareas. En general cuanto más explícita sea la red de restricciones, más restringido estará el espacio de posibles soluciones, y como consecuencia la búsqueda de una solución para el TCSP será más eficiente. Se puede inferir hasta encontrar una inconsistencia en la red o hasta que no se puedan inferir nuevas restricciones. En este último caso todas las soluciones se pueden derivar utilizando un algoritmo de búsqueda primero en anchura. La propagación de restricciones a toda la red utilizando inferencia es un proceso muy costoso que requiere la adición de un número exponencial de restricciones, por lo que la mayoría de los algoritmos de propagación de restricciones, solamente hacen propagación local, garantizando que se encuentre una inconsistencia o una solución, utilizando un algoritmo de búsqueda relativamente rápido. A estos algoritmos se les llama algoritmos de propagación de restricciones.

Podemos distinguir entre dos tipos de problemas TCSP. Cuando las restricciones entre dos variables (puntos temporales) pueden ser acotadas en un único intervalo temporal el TCSP es llamado STP (*Simple Temporal Problem*). Cuando por el contrario las restricciones se acotan en varios intervalos el TCSP se llama TCSP general que es bastante más complicado de resolver. La mayoría de las restricciones temporales que manejan los planificadores con extensiones temporales configuran un STP.

2. - *Un modelo completo para la planificación con redes jerárquicas de tareas*

Los buenos planes dan forma a las buenas decisiones. Es por eso que una buena planificación hace que los sueños esquivos se hagan realidad.

***Lester R. Bittel**, (1918) Profesor y escritor, en su libro: Las nueve claves maestras de la gestión (The Nine Master Keys of Management)*

El objetivo final de la planificación automática es que las investigaciones que se realicen en este área tengan un reflejo en cuanto a sistemas que reviertan en una mejora en la sociedad.

En este sentido desde el Departamento de Ciencias de la Computación e Inteligencia Artificial[2] de la Universidad de Granada[3], en el grupo dedicado a la planificación inteligente y al *scheduling* (secuenciación)[4], siempre se ha tenido muy clara esta vocación y toda la investigación se ha enfocado en la construcción de sistemas que puedan usarse en aplicaciones reales [70] [17].

En este capítulo se presentará un modelo completo de como afrontamos los problemas de planificación cuando es necesario integrarse con sistemas externos e interactuar con operadores humanos. Empezaremos definiendo qué es un ciclo de planificación, a continuación presentaremos la arquitectura de programas y tecnologías que es necesario implantar para hacer frente a ese ciclo de planificación. Después se presentará nuestro planificador jerárquico HTNP [87] (*Hierarchical Task Network Planner*), las tecnologías en las que se basa y las características principales en cuanto a expresividad del lenguaje de descripción de

2 DECSAI: http://decsai.ugr.es

3 http://www.ugr.es

4 Grupo SEPIA: http://decsai.ugr.es/~lcv/SEPIA/
Intelligent Systems Group: http://decsai.ugr.es/isg

dominios que utiliza HTN-PDDL. La exposición irá apoyada con ejemplos de sistemas reales basados en este modelo, como SIADEX[5] [87] [88] [76], ADAPTAPLAN [84] [83] y SAMAP [85] [114].

2.1. - Planificación aplicada en sistemas reales

Afortunadamente la planificación inteligente está abandonando el laboratorio académico y cada vez empieza a verse más su utilidad y a estar más aceptada dentro del ámbito empresarial e institucional.

La lista de planificadores que se aplican con éxito a la resolución de problemas reales cada vez es más amplia: Robótica [15], manufacturación [112], sistemas de manejo de emergencias (inundaciones [113], operaciones de evacuación y rescate [56] [27], incendios forestales [105] [87], gestión de procesos empresariales [71], sistemas de manufacturación [70], misiones espaciales [63] [96] [42], juegos [125] y un largo etcétera [50]. A medida que esta lista crece todos los investigadores están llegando a la misma conclusión: La dificultad no está únicamente en el modelo y en el algoritmo de planificación que se use, sino también en definir claramente el ciclo de vida del planificador, la interacción con el usuario final y los mecanismos de comunicación e interacción con este, y la integración con otros sistemas. Explicaremos como se ha abordado este problema en el desarrollo del sistema de ayuda a la decisión SIADEX y en otros proyectos donde se está aplicando el planificador HTNP.

El dominio de aplicación principal de HTNP y para el cual se empezó a desarrollar el algoritmo de planificación es el de la extinción de incendios forestales [75]. El objetivo principal de este proyecto es asistir a los técnicos de operaciones en la extinción de incendios forestales durante la fase de planeamiento, creando planes tentativos de movilización de recursos y ataque al incendio, partiendo de las estrategias definidas por los técnicos. Este proyecto es financiado por la Consejería de Medio Ambiente de la Junta de Andalucía bajo el contrato de investigación NET033957.

2.2. - El problema de los incendios forestales

Los incendios forestales son una de las mayores amenazas para la preservación del medio natural por lo que desde las distintas administraciones públicas se hace un esfuerzo continuo en recursos y tecnologías a fin de mejorar y ampliar los medios de extinción y detección existentes así como en mejorar su coordinación y capacidad de adaptación [77]. La planificación automática es sin

5 http://siadex.ugr.es

duda una tecnología que puede ayudar en este objetivo de mejorar la eficiencia y coordinación de los recursos. De hecho existen otras iniciativas distintas del sistema SIADEX encaminadas a este objetivo.

PHOENIX [21] es una iniciativa llevada a cabo en los estados unidos entre 1989 y 1993 que incorpora un sistema jerárquico multiagente con un planificador y un simulador gráfico de incendios forestales. La intención de PHOENIX es que el técnico en extinción interactúe con el simulador haciendo pruebas de la evolución del incendio cuando se introducen cierta cantidad de medios. Las capacidades de planificación de este sistema son bastante limitadas ya que el objetivo de la herramienta es la de utilizarse en la fase de evaluación de la situación más que en la de planificación en sí.

CHARADE [105] es otro sistema desarrollado en Italia entre 1992 y 1995 construido sobre un planificador basado en casos interactivo. Este sistema está orientado al primer ataque contra el incendio, su capacidad de adaptación ante imprevistos es limitada y carece de la capacidad de reparar el plan.

CARICA [9] es en cierta medida el sucesor de CHARADE pero desarrollado en Francia entre los años 1995 a 1997. El objetivo principal de esta herramienta es la de entrenar a técnicos en la extinción de incendios noveles, y por lo tanto no es una herramienta orientada a la ayuda a la extinción en incendios reales.

Tomando en cuenta la experiencia previa en estos sistemas se desarrolla el planificador HTNP, sabiendo por un lado, que el objetivo del sistema SIADEX es el de utilizarse en incendios reales y por otro que al basado en técnicas de planificación generativa su capacidad de adaptación es mayor que la de los sistemas basados en casos.

Diseñar un sistema de asistencia a la extinción de incendios forestales no es únicamente interesante desde el punto de vista de tratar de mejorar nuestro entorno natural sino también es un reto para cualquier investigador. En los incendios forestales y en general en cualquier sistema para la gestión de emergencias hay una impredecibilidad y dinamicidad en el entorno que lo convierten en un problema muy interesante.

2.3. - Generando planes de visita de forma automática

Otro problema muy interesante al cual se puede aplicar el mismo modelo es la generación de planes de visita para una ciudad. SAMAP (*adapative multi-agent planning systems dependent on context*) es un sistema que usa un planificador inteligente para desarrollar estos planes de visita, teniendo en cuenta las

preferencias del usuario. La idea es que el usuario final pueda acceder a través de una PDA o un teléfono móvil al propio sistema y que este le guíe a través de la ciudad e incluso pague las entradas de los monumentos o museos que quiera visitar o reserve habitación en el hotel [85] [114].

2.4. - Planes de aprendizaje adaptados al usuario

La última aplicación del modelo que se propone es la de la generación de planes de aprendizaje adaptados al usuario dentro del sistema ADAPTAPLAN [84] [83] (adaptación basada en aprendizaje, modelado y planificación para tareas complejas orientadas al usuario). Este sistema, dado un concepto educativo que el alumno necesita aprender, una serie de unidades educativas preparadas por el profesor, que tienen una duración y unos requisitos para poder realizarla (haber terminado otra unidad por ejemplo), y el perfil del alumno, elabora un plan de aprendizaje a lo largo del tiempo. El plan puede requerir de la realización de tareas periódicas o en ciertos instantes de tiempo. También hay que tener en cuenta por ejemplo que la carga del alumno (número de unidades que está realizando) tiene que adaptarse al esfuerzo máximo que este puede dedicar, o la disponibilidad de medios hardware o software de los que dispone el alumno.

2.5. - Ciclo de vida de la planificación

Uno de los principales problemas de la planificación es cómo enfrentarse al cambio, a lo imprevisto. Los enfoques utilizados en los modelos de planificación clásica parten de la suposición de conocimiento completo del entorno y por tanto asumen que el plan se va a ejecutar correctamente. Por lo tanto una vez que el planificador ha obtenido un plan válido, se da por cerrada la fase de planificación y los distintos agentes comienzan su ejecución.

Sin embargo hay una enorme cantidad de campos en los que se pueden aplicar técnicas de planificación, en los que no se puede asumir esta suposición. Cuando el planificador obtiene el primer plan tentativo es simplemente el inicio de una continua espiral de adaptaciones del plan inicial ante sucesos no previstos. Detectar estos sucesos imprevistos requieren de una monitorización de los agentes que ejecutan las tareas y del entorno donde se ejecuta el plan. La monitorización puede ser llevada a cabo por los propios agentes, o por otros entes externos, en cualquier caso debe haber un proceso que vaya realizando el seguimiento del plan y detectando que cambios en el entorno pudieran poner en peligro la ejecución del plan. Una vez que se ha detectado un cambio en el entorno que imposibilita la ejecución del plan, es necesario replanificar para reparar el plan inicial y obtener una nueva versión viable del plan.

En general, el ciclo de vida de cualquier planificación compleja, intervengan en ella o no un planificador automático, y ya sea para la gestión de crisis provocadas por un incendio forestal o para dirigir un robot en la superficie marciana, se puede dividir en una serie de etapas que se resumen en la Ilustración 8:

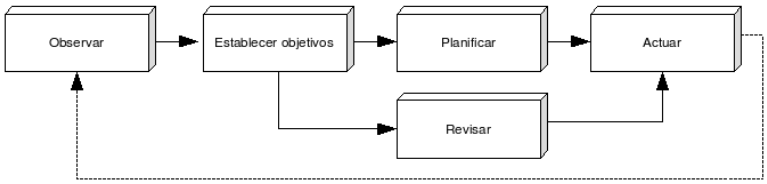


Ilustración 8: Ciclo de vida de la planificación

1. Observar y evaluar la situación: Durante esta fase se observa el entorno y el problema a resolver, recogiendo toda la información que sea relevante.

2. Establecer los objetivos: Una vez evaluado el problema, se marcan los objetivos que es necesario cumplir para tratar de resolverlo.

3. Planificar: Se calcula un plan que haciendo uso de los recursos disponibles, cumpla con los objetivos a satisfacer.

4. Actuar: Se distribuye el plan entre los distintos agentes encargados de realizar las distintas tareas y comienza la ejecución del plan.

5. Observar y monitorizar: Una vez lanzada la ejecución del plan pueden ocurrir alguno de los escenarios:

a) **Finalización:** El plan se ejecuta completamente y se consiguen los objetivos, con lo que termina el proceso.

b) **Reparación:** Ocurre alguna eventualidad que impide que se realice parte o la totalidad del plan. Es necesario reparar el plan teniendo en cuenta ahora las siguientes variables:

i. Que ya hay agentes realizando algún tipo de tarea.

ii. Que hay objetivos que pueden haberse satisfecho.

iii. Que el coste de realizar cambios radicales sobre el plan inicial puede ser elevado dependiendo del contexto, y que puede ser interesante mantener en parte la estructura del plan antiguo.

Las técnicas de planificación automática son inmediatamente aplicables en las fases de planificación y reparación de planes, pero para que el sistema resultante sea finalmente utilizable en la resolución de problemas reales, es necesario que se contemplen todas las fases.

2.6. - Un modelo general para la planificación

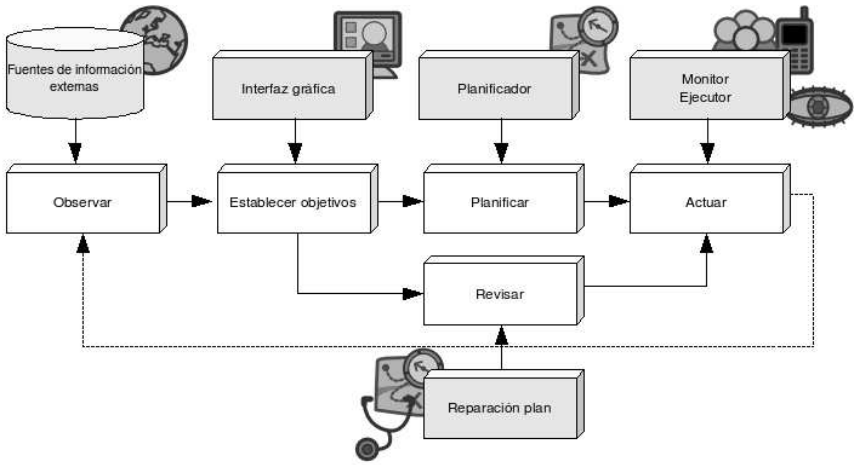


Ilustración 9: Modelo general para la planificación

En la Ilustración 9 se muestra como se pueden integrar ciertos subsistemas automáticos o semiautomáticos dentro del ciclo de planificación.

Un sistema basado en técnicas de planificación automática y en otras técnicas de la Inteligencia Artificial, puede integrarse en algunos de los pasos del ciclo de vida del proceso estudiado anteriormente de la siguiente forma:

- **Observar:** Para que el experto sea capaz de evaluar una situación y por lo tanto plantear los objetivos, necesita disponer de toda la información posible. La información suele encontrarse distribuida en distintos sistemas que es necesario recopilar, unificar y validar. Se pueden usar aquí todo tipo de técnicas de extracción de información y de clasificación así como ontologías [53] para etiquetar y unificar modelos de

datos que en principio son distintos y obtener un tipo de información que finalmente pueda ser procesada por el sistema de planificación.

- **Evaluación y planteamiento de los objetivos:** El planteamiento de los objetivos normalmente responde a decisiones de tipo estratégico. Es necesario disponer de herramientas gráficas para la consulta de información así como de asistentes que apoyen al experto a la hora de realizar la decisión. Se pueden usar sistemas de aprendizaje que tomen como base la experiencia previa de otras decisiones y que se adapten al tipo de decisiones que toma el usuario para apoyar al experto [59] .

- **Planificación:** Para realizar el proceso de planificación podemos utilizar técnicas de planificación automática. La planificación automática permite una toma de decisiones a mucho más bajo nivel, de forma más rápida, respetando todas las restricciones y sin cometer errores que lo que lo puede hacer un humano. El experto humano se "limitaría" a tomar las decisiones estratégicas de alto nivel o a aquellas decisiones que considere que son importantes debido a su experiencia en la materia. Aquí también entran en juego técnicas de secuenciación o "scheduling" [122] para la gestión óptima de tiempo y recursos, imprescindible en casi cualquier ámbito de aplicación de las técnicas de planificación. E incluso técnicas de aprendizaje para mejorar la calidad de los planes [134].

- **Distribución, monitorización y ejecución:** La distribución del plan puede ser realizada de forma automática, utilizando las nuevas tecnologías de telefonía móvil, internet sin cables y aprovechando la cada vez mayor potencia de cálculo de los dispositivos móviles. Cuando el plan es ejecutado a través de estos propios dispositivos o usando otros sensores externos, podemos ir monitorizando con un proceso automático la correcta ejecución del plan. Si disponemos de este monitor automático, éste puede advertir al experto de posibles fallos o retrasos, así como proporcionarle una visión general del estado del plan.

- **Reparación:** Es difícil que un plan medianamente complicado destinado a ejecutarse en un mundo real y cambiante donde intervengan múltiples agentes se ejecute a la primera. El monitor puede detectar esos fallos y lanzar un proceso de reparación que tratará de adaptar y reparar un plan previamente existente. El proceso de reparación, a no ser que sea trivial, muy probablemente requiera de la intervención del experto que puede replantearse la estrategia general, comenzando un nuevo ciclo.

2.7. - El planificador HTNP

Una vez que se ha presentado el modelo general de un sistema basado en técnicas de planificación, se estudiará el planificador HTNP (*Hierarchical Task Network Planner*), que es el que se estamos utilizando desde la Universidad de Granada para resolver problemas de planificación. Se destacarán sus características principales que lo hacen muy útil a la hora de enfrentarse con los problemas que estamos trabajando.

HTNP es un planificador basado en el modelo de planificación HTN [33]. Quizá el planificador que más se le parece es SHOP2 [28]. Sus principales características son:

- Como ya se ha dicho es un planificador **jerárquico.**
- Es un planificador **basado en estados**. Esto es, a pesar de que la búsqueda se va haciendo mediante la sustitución de una tarea abstracta por la red de tareas en la cual se expande, cuando se aplica un operador primitivo, se producen cambios en el estado actual, que va evolucionando.
- Es un planificador **hacia adelante**, ya que los estados van evolucionando a partir del estado inicial, aunque no se busca un estado objetivo, al utilizar un modelo HTN.
- Es un planificador **temporal.** Aunque en su versión más simple, que es la que se presenta en esta sección, no lo es. Estudiaremos las extensiones temporales en los siguientes capítulos.

HTNP es un planificador **independiente del dominio** que se está usando (y se usará) con muy buenos resultados en distintos dominios de aplicación. No debe enmarcarse dentro del proyecto SIADEX porque se diseñó como un planificador de propósito general. Sin embargo algunos de los requisitos de SIADEX, que son comunes a cualquier sistema de gestión de crisis, definieron el tipo de planificador por el que nos debíamos decantar.

2.7.1. - *Requisitos del planificador*

Cuando se trata de aplicar las técnicas de planificación a problemas complejos, la mayoría de los requisitos que se le imponen al planificador son siempre los mismos :

- **Adecuación al conocimiento de los expertos**. Escribir dominios de planificación es una tarea compleja, debido sobre todo a que la planificación clásica está muy orientada a representar el conocimiento

causal y no trata bien ni las preferencias del usuario ni la estructura a un nivel más abstracto del plan. La comunidad de planificación ahora que se empieza a disponer de planificadores capaces de resolver problemas cada vez más complejos empieza a dar una mayor importancia a la representación y modelado del conocimiento y a disponer de lenguajes de modelado de dominios cada vez más ricos y expresivos. Fruto del creciente interés en una mejor gestión del conocimiento es la aparición de la ICKEPS[6] o la inclusión de preferencias de usuario en la última versión de PDDL, la 3.0. Parece demostrada que la expresividad de los planificadores HTN es mayor que la de los planificadores planos [34]. De hecho el modelo HTN se adapta muy bien a la estructura de los protocolos de actuación ante una situación de emergencia. En general parece más sencillo y adecuado a un enfoque de ingeniería del conocimiento organizar las acciones en base a tareas abstractas y a operadores primitivos, siguiendo un enfoque HTN.

- **Necesidad de eficiencia.** Un motivo de preocupación para cualquier investigador que quiera incluir un planificador como una pieza más en una aplicación real, es que el tiempo de respuesta de éste sea razonable. Los dominios reales suelen trabajar con cientos o miles de objetos, y suelen necesitar de planes con cientos de acciones de muy diverso tipo. Los planificadores que se muestran en general más rápidos en la IPC un año tras otro, son los heurísticos, los SAT y los basados en GRAPHPLAN. Sin embargo todos estos planificadores sólo operan sobre dominios planos, tipo puzzle y la expresividad de los problemas que pueden afrontar está limitada (aunque hay que destacar que aumenta considerablemente competición tras competición). Los requerimientos de expresividad generalmente son más importantes que los de velocidad y no sólo eso sino que influyen en la velocidad del planificador. Algunas construcciones expresivas del lenguaje son costosas de calcular. Los planificadores jerárquicos también pueden ser muy rápidos, incluso en problemas tipo puzzle, como demostró SHOP2 en la IPC del 2002. HTNP es un planificador que demuestra ser aún más rápido que SHOP2. El planificador no debe ser únicamente eficiente en la búsqueda de planes, sino en memoria y en capacidad de procesamiento de volúmenes de información grandes. Esto es especialmente relevante en el sistemas donde el número de objetos de dominio, predicados y acciones es muy alto.

- **Necesidad de control sobre la generación del plan.** En

6 ICAPS Competition for Knowledge Engineering for Planning and Scheduling

muchas ocasiones hay procesos, entendiendo un proceso como una lista de acciones realizadas en un determinado orden, que se tienen que realizar de una determinada forma (porque lo impone la ley, o porque la experiencia ha demostrado que es la forma más adecuada de hacerlo). En aplicaciones reales además este tipo de procesos son muy frecuentes. No se puede dejar a la capacidad generativa del planificador la "reinvención" de estos procesos, se tienen que realizar de una forma en concreto. La planificación basada en casos es muy útil en estos casos, desgraciadamente se pierde mucha capacidad generativa que puede ser necesaria en otras partes del plan. El modelo de planificación HTN es ideal para codificar este tipo de procesos y para controlar el proceso de generación del plan. Esta capacidad de control que tiene el modelo HTN sobre el plan que finalmente se genera se puede aprovechar también en una segunda vertiente para incluir heurísticas que aceleren el proceso de planificación mejorando la eficiencia [35] [98].

- **Necesidades temporales.** El planificador que se usara debería tener capacidades de razonamiento temporal, ya que, en prácticamente, cualquier dominio de aplicación real son necesarias. Se deberían poder introducir de una manera sencilla restricciones temporales de cualquier tipo.

- **Soporte para PDDL.** Una opción muy interesante es que el planificador fuera capaz de procesar dominios escritos en PDDL, por dos motivos. El primero es el de así poder usar dominios escritos para la IPC o para otros planificadores con soporte PDDL. El segundo el adaptarse a un lenguaje que es un estándar dentro de la comunidad de planificación.

- **Necesidad de integración.** Es muy importante que el planificador pueda integrarse con otras plataformas. En este sentido es muy importante que el planificador tenga una interfaz de comunicación abierta que permita a otros componentes software interactuar con él. De esta forma se pueden construir interfaces de usuario o de depuración que interactúen directamente con el planificador o otros componentes software que hagan uso del servicio de planificación. Esta necesidad además ha ido creciendo con el tiempo a medida que se va integrando el planificador dentro de nuevas aplicaciones.

- **Necesidades de depuración.** Una última característica muy importante desde el punto de vista del escritor de dominios es que el proceso de planificación pudiera trazarse a fin de encontrar errores en la codificación del dominio. En dominios HTN esto es además especialmente relevante debido a la cantidad de información de control codificada en el

dominio.

SHOP2 [28] es quizá el planificador que mejor se adaptaba al tipo de problemas a los cuales se quería aplicar técnicas de planificación, pero las razones que motivaron la implementación de un planificador totalmente nuevo fueron:

- El manejo de SHOP2 del tiempo no es muy bueno. SHOP2 maneja el tiempo utilizando un mecanismo llamado MTP (*Multi-timeline preprocessing*) [28]. MTP deja en gran medida la labor de la gestión de la línea de tiempo en manos del escritor de dominios ya que esta se lleva a cabo mediante la lectura, modificación y escritura de literales numéricos (*fluents*). Sin cuidado en la escritura del dominio se permite la violación de restricciones temporales. Esto hace que la tarea de escribir dominios temporizados en SHOP2 sea complicada.

- Teníamos ciertas dudas de que SHOP2 fuese eficiente manejando grandes volúmenes de información.

- SHOP2 no utiliza un lenguaje de especificación de dominios estándar como PDDL, su lenguaje se apoya en la sintaxis de LISP, y está muy orientado a la forma de trabajar de ese planificador.

Incorporar todas estas características a SHOP2 era un proceso más complejo que partir de un modelo de planificador distinto, debido al esfuerzo que implicaba en ingeniería inversa. Por eso se estudió SHOP2 y otros planificadores jerárquicos y no jerárquicos a fin de incluir las características de los mismos más importantes dentro del nuevo planificador HTNP que cubriría todos los requisitos.

2.7.2. - *Nomenclatura*

La primera versión del algoritmo HTNP, que se expondrá en esta sección, no es temporal. Esto nos servirá para entender las bases del algoritmo antes de abordar las distintas extensiones temporales.

HTNP recibe como entrada un fichero de problema y un fichero de dominio descritos en el lenguaje HTN-PDDL cuya sintaxis concreta se discutirá más adelante[7]. HTN-PDDL es un lenguaje basado en PDDL que al igual que este, su dominio $\mathcal{D}$ se puede describir en base a cinco elementos:

- $\mathcal{V}$: Un conjunto de símbolos variables.

7 La versión actual en desarrollo del algoritmo soportará entradas en otros lenguajes como por ejemplo XML.

- $\mathcal{C}$: Un conjunto de símbolos constantes.
- $\mathcal{P}$: Un conjunto de proposiciones o predicados.
- δ: Un conjunto de acciones, que al tratarse de un modelo HTN pueden ser de dos tipos:
  - σ: Operadores primitivos.
  - τ: O tareas abstractas que llevan asociados un conjunto de métodos de descomposición.
- $\mathcal{A}$: Un conjunto de axiomas.

Y su problema $\mathcal{P}$ se puede definir en base a:

- ε_0: El estado inicial.
- Γ_g: La red de tareas a resolver.

Un estado en HTNP ε al igual que los estados de STRIPS está compuesto de una serie de proposiciones instanciadas. Una proposición instanciada es aquella que no tiene variables libres. Las proposiciones definen los hechos que son ciertos en ese estado. En HTNP se parte del estado inicial, que notaremos como ε_0 que se define en el fichero de problema, siguiendo una sintaxis muy similar a la de PDDL.

Los operadores primitivos definidos en el fichero del dominio son muy parecidos a los acciones del modelo de STRIPS y se basan en la sintaxis de PDDL. El operador primitivo tiene una precondición que debe ser cierta en el estado actual para poder aplicar los efectos, que son los que van produciendo el cambio en el estado.

Las tareas abstractas τ_i se describen en el fichero de dominio en el lenguaje HTN-PDDL. Una tarea abstracta se representa como:

- μ: Un conjunto de métodos con unas precondiciones.
- Γ: Un conjunto de redes de tareas por las que τ_i se puede sustituir, asociadas a los métodos de μ.
- θ_i: Un orden en cada una de las redes de tareas en Γ.

Una red de tareas puede tener tanto operadores primitivos como tareas abstractas o mezcla de ambos. Cuando una red de tareas Γ_i, está compuesta únicamente de operadores primitivos decimos que es una red de tareas primitiva.

El orden de ejecución θi que se impone sobre una red de tareas Γ_i, codifica un grafo dirigido acíclico, esto es no se permiten ciclos dentro de Γ_i.

Un plan π en construcción es una red de tareas con un orden, donde al menos una de las acciones que lo componen es una tarea abstracta.

Un método μ_j es una regla asociada a una tarea abstracta τ_i que define en que condiciones del estado actual se puede aplicar la sustitución de τ_i por una red de tareas Γ_j especificada en el método. Una tarea abstracta puede tener varios métodos de descomposición, pero debe tener al menos uno. Elegir el método adecuado a aplicar es tarea del planificador, que en principio puede escoger de forma no determinista[8] entre los métodos aplicables.

El plan se va construyendo mediante la aplicación de métodos de sustitución de tareas abstractas por subredes de tareas, y la aplicación de operadores primitivos que van produciendo cambios sobre el estado. Esta forma de construir el plan va creando un *árbol de expansión de tareas.*

Un plan π devuelto por HTNP es una red de tareas primitiva. El plan es válido si, dada cualquier linearización del plan que respete el orden θ_π, partiendo del estado inicial ε_0 y aplicando cada una de las tareas primitivas σ_i todas las operaciones primitivas pueden ejecutarse de forma correcta, esto es, su precondición es válida en el estado actual.

- $\varepsilon_{i+1} = apply(\varepsilon_i, effects(\sigma_i))$
- $\forall_i, preconditions(\sigma_i, \varepsilon_i) = true$

HTNP solo tiene una linearización válida (el plan se devuelve como una única secuencia de acciones). Veremos más adelante que sus extensiones temporales no.

2.7.3. - *Redes de tareas en HTNP*

Antes de definir el algoritmo es necesario especificar el tipo de redes de acciones que el algoritmo tiene que tratar.

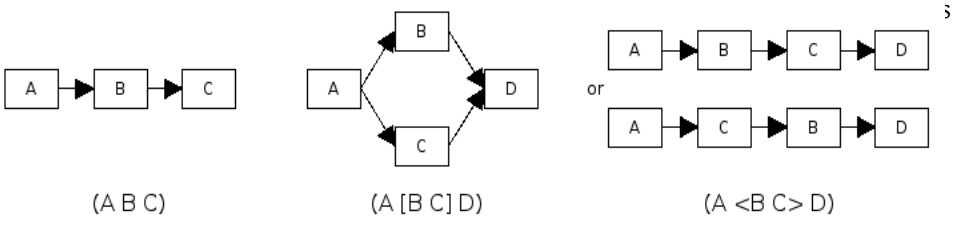


Ilustración 10: Diferentes redes de acciones soportadas en HTNP

- **Ordenación secuencial.** En esta ordenación las acciones deben realizarse en el plan final en el mismo orden en el que están declaradas. Usamos una notación donde las acciones son puestas entre paréntesis para indicar este tipo de ordenación. Por ejemplo dada la red de tareas (A B C), para poder realizar la acción B (o cualquiera de sus subtareas si se trata de una tarea compuesta) antes es necesario realizar A (todas las subtareas de A si A es una tarea compuesta). HTNP resuelve este tipo de ordenación no insertando B en la agenda hasta que A esté resuelta.

- **Ordenación paralela**. Se usarán corchetes para indicar este tipo de ordenación. Esta ordenación indica que en principio no hay ninguna dependencia entre las acciones involucradas (ni entre sus subtareas si se trata de tareas abstractas). HTNP por lo tanto puede escoger de forma no determinista cualquier posible ordenación entre las tareas (puede insertarlas en cualquier orden en la agenda). Cuando una de las acciones de la red falla, la red entera se considera fallida, con lo que se producirá un bactracking. Esto es equivalente a afirmar que HTNP solamente comprueba una posible ordenación en la agenda entre las tareas y si esta falla no intenta el resto. Que se suponga independencia entre las tareas no implica que HTNP no cree dependencias, como se discutirá más detenidamente a continuación. Dada la red de acciones (A [B C] D) de la Ilustración 10, una vez que el algoritmo HTNP ha explorado la acción A, insertará en la agenda las acciones B y C como pendientes y en principio escogerá de forma no determinista entre ellas. Hasta que B y C no hayan sido resueltas, el algoritmo no insertará D en la agenda. Si por ejemplo intenta primero B y falla automáticamente descartará también a C.

- **Ordenación permutable.** Usaremos la notación entre ángulos para indicarla. Este tipo de ordenación es muy similar a la ordenación paralela y se utiliza para que el planificador explore **todas las posibles ordenaciones entre las acciones** (y entre sus subtareas si estas son tareas compuestas), debido a que hay ocasiones en el escritor de dominios no puede conocer de antemano si existe algún tipo de interdependencia entre dos acciones y queremos que el planificador lo analice por nosotros. Por ejemplo dada la red de acciones (A <B C> D) el planificador probaría tanto la ordenación (A B C D) como la ordenación (A C B D). El costo computacional de analizar todas las posibles ordenaciones, es, como es obvio, muy grande y si no se usa con cuidado se pierde en gran medida la ventaja que tiene la planificación HTN de evitar la exploración combinatoria por medio de heurística, frente a otros modelos de planificación.

2.7.4. - Discusión de independencia y completitud

Cuando se afirma que dos ramas dentro de una red de tareas con una ordenación paralela son independientes, no quiere decir que HTNP no pueda establecer relaciones de dependencia entre ellas durante el proceso de planificación. Supongamos que tenemos la siguiente red de tareas [A B] y que A se descompone en la red de tareas primitiva (a1 a2 a3) y B en (b1 b2 b3). Supongamos también que el planificador decide primero resolver la subred de tareas de A y que el operador primitivo a2 añade un literal al estado actual. Es perfectamente válido que ese literal pudiera ser consumido por otro operador de la subred de B por ejemplo b1, estableciéndose una dependencia entre las tareas A y B (b1 no se podrá ejecutar si antes no se ejecuta a2).

Por esta misma razón es posible que alterándose el orden en el que el planificador considera las tareas de la agenda una descomposición (B A) para la red de tareas [A B] se pueda realizar, mientras que una (A B) falle. El problema aquí no está en la completitud del algoritmo de planificación, sino que es un fallo de concepto en la escritura del dominio ya que esto indica que si que existe algún tipo de dependencia entre A y B y el escritor de dominios debería haber escrito <A B>.

El modelo de planificación HTN permite al escritor de dominios, como ya se ha comentado, introducir mucha información de control de la búsqueda en el dominio. El escritor de dominios es muchas veces reticente a usar la ordenación permutable y prefiere usar la paralela a sabiendas de que pueden existir dependencias entre las subtareas con el fin de evitar la explosión combinatoria que se produce (sobre todo si la red permutable está en un nivel alto del árbol de expansión). El lenguaje HTN-PDDL proporciona herramientas sintácticas para controlar (en cierto grado) esta explosión combinatoria como se verá más adelante.

2.7.5. - Concepto de contexto de planificación

El algoritmo HTNP realiza una búsqueda en el espacio de posibles árboles de expansión de tareas (planes jerárquicos), pero al mismo tiempo va generando todos los estados intermedios. Se utiliza un algoritmo de backtracking que utiliza una búsqueda primero en profundidad en la selección de métodos, unificaciones y tareas con orden paralelo (por defecto).

Cuando se analizó el rendimiento de las primeras versiones del algoritmo HTNP una de las mayores causas de consumo de memoria y de tiempo de procesamiento era la gestión de las estructuras de datos asociadas al estado y a la red de tareas en expansión, que se clonaban en cada ciclo del algoritmo. Para tratar

de paliarlo se adoptaron varias medidas:

- Una representación de literales y acciones mucho más optimizada basada en vectores de enteros.
- Se sustituyó el algoritmo recursivo por uno iterativo.
- Se utilizó un enfoque "incremental" del algoritmo de manera que solamente se almacenan los cambios que se producen de ciclo en ciclo del algoritmo de planificación. La estructura que almacena la información del cambio se llama *contexto.*

El algoritmo HTNP mantiene una pila de *contextos* por los que va pasando el planificador a fin de simular la recursividad. Cada *contexto* almacena los cambios sufridos en un paso por el plan en expansión Π_i, en el estado ε_i y en otras estructuras auxiliares en un paso de ejecución. Cuando se hace bactracking se coge el contexto en el tope de la pila ω_i y se intenta seguir hacia adelante, si no es posible, se deshacen los cambios realizados para ese contexto se quita el contexto del tope de la pila ω_i y se intenta de nuevo con el anterior ω_{i-1}.

Cada *contexto* ω_i almacena la siguiente información:

- $\alpha_i^{pending}$: La agenda de acciones (tareas abstractas y operadores primitivos) pendientes de ejecución.
- $\alpha_i^{explored}$: La lista de acciones de la agenda que ya han sido descartadas.
- $\rho_i^{pending}$: Una lista de las acciones que unifican con la tarea actualmente seleccionada en la agenda que están pendientes de analizar. En HTN-PDDL el lenguaje usado por HTNP se permite la declaración de tareas con el mismo nombre y número de argumentos.
- $\rho_i^{explored}$: De la lista anterior las acciones que ya han sido descartadas por el algoritmo.
- $\mathcal{U}_i^{pending}$: La lista de sustituciones de variables por constantes o unificaciones. Las unificaciones pueden darse bien con las precondiciones de un método de expansión asociado a una tarea abstracta o bien con las precondiciones de un operador primitivo en expansión.
- $\mathcal{U}_i^{explored}$: De la lista de unificaciones las que ya han sido descartadas por el algoritmo.
- $\mu_i^{pending}$: Si se está expandiendo una tarea abstracta, de esa tarea, la lista de métodos que quedan por considerar.

- $\mu_i^{explored}$: Los métodos que ya han sido considerados y descartados por el algoritmo.
- *diff*(Π_{i-1},Π_i): La lista de cambios realizados sobre el plan en expansión desde el contexto anterior hasta este.
- *diff*(ε_{i-1},ε_i): Los cambios realizados desde el estado en el contexto anterior hacia el actual.

2.7.6. - Bucle principal de HTNP

Teniendo en cuenta la forma incremental de operar del algoritmo, el concepto de *contexto,* y sabiendo que el planificador recibe como entradas el dominio $\mathcal{D}$=<$\mathcal{V}$,$\mathcal{C}$,$\mathcal{P}$,δ,$\mathcal{A}$> y el problema $\mathcal{P}$ = <ε_0, Γ_g>, el bucle principal utilizado por el algoritmo de planificación HTNP se puede expresar de la siguiente forma:

```
HTNP(<V, C, P, δ, A>, <ε0, Γg>)
1.   // Inicialización de ω0
2.   αi^pending = first(Γg) //Inicializar la agenda con la red de
     tareas objetivo. (Las acciones de la red de tareas que no
     tienen dependencias de orden).
3.   Π0 = Γg  //El plan en expansión es la red de tareas objetivo
4.   push(stack,ω0)
5.   while not empty(stack) do: // Mientras queden contextos
     por analizar
6.          ωi= top(stack)
7.          ωi+1 = generate_new_context(ωi) // Generar a partir del
            contexto actual uno nuevo (dar un paso en la
            planificación) tomando las decisiones que sean
            adecuadas.
8.          if ωi+1 == FAIL then: // No quedan decisiones por
            tomar o ninguna es válida
9.                 undo(ωi) // deshacer los cambios provocados en
                   el plan en construcción y en el estado
10.                pop(stack) // backtrack
11.         else:
12.                if empty(αi+1^pending) then: // en el contexto que se
```

```
                acaba de generar no quedan acciones por
                expandir en la agenda, eso significa que
                tenemos un plan
13.                     return Π_{i+1} // devolver el plan
14.             else: // se dio un paso correctamente
15.                     push(stack,ω_{i+1})
16.  return FAIL // no se encontró plan
```

Algoritmo 6: bucle principal de HTNP

El bucle de HTNP permite de una forma secuencial implementar un algoritmo que es recursivo mediante el uso de una pila y los contextos. La función más interesante es la que genera un nuevo contexto a partir del actual *generate_new_context* ya que es la que toma las decisiones del planificador. Esta función, cuando se la llama, simplemente analiza las decisiones que quedan por tomar en las listas *pending* y las va aplicando cuando es posible. El orden en el que va analizando estas listas es el orden en el que se reintentan otras decisiones cuando el algoritmo vuelve por backtracking y es el siguiente:

- Escoger otra posible unificación de la lista de pendientes $\mathcal{U}_i^{\text{pending}}$.
- Escoger un nuevo método si estamos descomponiendo una tarea compuesta de la lista μ_i^{pending}.
- Escoger otra acción de las que unifiquen con la acción actualmente seleccionada en la agenda ρ_i^{pending}.
- Escoger una nueva acción de las disponibles en la agenda $\alpha_i^{\text{pending}}$.

Teniendo este orden en cuenta, la función generate_new_context, tiene la siguiente estructura:

```
generate_new_context(ω_i)
1.  while not empty(α_i^pending) then:
2.        α_j = first_of(α_i^pending)
3.        if empty(ρ_i^explored) and empty(ρ_i^pending) then:
4.              ρ_i^pending = matching_actions(α_j,δ)
5.        while not empty(ρ_i^pending) then:
6.              ρ_j = first_of(ρ_i^pending)
7.              if compound(ρ_j) then:
```

```
8.                    if empty(μi^pending) and empty(μi^explored) then:
9.                        μi^pending = generate_methods(ρj)
10.                   while not empty(μi^pending) then:
11.                       μj = first_of(μi^pending)
12.                       if empty(Ui^explored) and empty(Ui^pending)
                          then:
13.                           Ui^pending =
                              generate_unifications(μi,εi)
14.                       while not empty(Ui^pending) then:
15.                           Uj = first_of(Ui^pending)
16.                           Π^i+1 = replace(Πi,ρj,Γj in μj)
                              //Añadir nuevas tareas al
                              plan en construcción
17.                           αi+1^pending = αi^pending - ρj + first(Γj)
                              //Actualizar la agenda
18.                           delete(first_of(Ui^pending))
19.                           return  ωi+1
20.                       clear(Ui^explored), clear(Ui^pending) //Se
                          exploraron todas las unificaciones
                          se prueba un nuevo método
21.                       delete(first_of(μi^pending))
22.               else: // es un operador primitivo
23.                   if empty(Ui^explored) and empty(Ui^pending) then:
24.                       Ui^pending = generate_unifications(μi,εi)
25.                   while not empty(Ui^pending) then:
26.                       Uj = first_of(Ui^pending)
27.                       αi+1^pending = αi^pending - ρj //Actualizar la
                          agenda
28.                       εi+1 = εi + apply_effects(ρj)
29.                       delete(first_of(Ui^pending))
30.                       return  ωi+1
31.               delete(first_of(ρi^pending))
```

```
32.        clear(ρ_i^pending), clear(ρ_i^explored) //Se exploraron todas los
           matching con la agenda. Probar una nueva acción de la
           angenda
33.        delete(first_of(α_i^pending))
34. return FAIL
```

Algoritmo 7: Generar un nuevo contexto

2.8. - El lenguaje HTN-PDDL

Hasta ahora se ha presentado el algoritmo HTNP centrándose en las características técnicas del mismo. En esta sección trataremos de presentar las mejoras de expresividad que se le ofrecen al escritor de dominios y de las que no disponen otros planificadores.

Para explicar estas mejoras nos apoyaremos en las herramientas sintácticas que ofrece el lenguaje HTN-PDDL que en principio es el lenguaje de descripción de dominios y de problemas que usa HTNP[9].

HTN-PDDL no debe verse como un lenguaje de planificación diseñado en exclusiva para el planificador HTNP. La intención desde el principio es dotar al lenguaje PDDL de las construcciones sintácticas necesarias para poder hacer planificación HTN de una forma lo suficientemente genérica, para que cualquier planificador HTN pudiera utilizar esta extensión. De hecho las necesidades expresivas que se han detectado durante la escritura del dominios en HTN-PDDL, que han provocado cambios en el lenguaje, son las que posteriormente han obligado a realizar cambios en HTNP para soportar estas nuevas características.

La sintaxis de HTN-PDDL se describe en detalle en el Anexo II, esta sección no pretende realizar un análisis exhaustivo del lenguaje, únicamente trata de destacar sus características más relevantes y novedosas.

2.8.1. - *Motivación*

En la literatura ya existen lenguajes para la escritura de dominios HTN, se tratará de explicar las razones que motivaron la creación de un lenguaje nuevo.

La mayoría de los lenguajes HTN existentes, no se piensan como un lenguaje general sino que se diseñan para un planificador en concreto.

9 Como ya se ha mencionado, está previsto el soporte de otros lenguajes en la versión en desarrollo del planificador.

- UMCP [33] es un planificador HTN muy orientado a demostrar la completitud de los algoritmos HTN, su sintaxis es muy sencilla y al mismo tiempo limitada, sobre todo en la flexibilidad a la hora de definir redes de tareas. El lenguaje está totalmente descrito en LISP, de forma que es el propio intérprete de LISP el que construye las estructuras de datos.

- SHOP [97] y SHOP2 [28], son en gran medida sucesores de UMCP y su sintaxis es similar, sin embargo permiten un mayor control sobre la búsqueda.

- HYBIS [17], al ser un planificador puramente jerárquico, permite usar distintas representaciones a distintos niveles en la jerarquía. Esto le da una expresividad muy potente, pero tiene el inconveniente de complicar bastante la escritura de dominios.

- OPLAN [23] usa un lenguaje de representación de tareas abstractas (esquemas) muy limpio y bastante cercano al usuario, pero bastante distinto de PDDL.

- SIPE [59] es quizá el planificador que tiene un lenguaje de representación más completo. No solamente incluye la representación HTN, sino que por ejemplo el lenguaje también soporta definir mecanismos de reparación y adaptación durante la ejecución.

- PASSAT [94] es un planificador jerárquico que tiene la posibilidad de incluir preferencias y restricciones de usuario en su lenguaje. También tiene la posibilidad de almacenar trozos de planes parcialmente especificados para su posterior uso. Muchos de los conceptos que tiene el lenguaje de PASSAT son muy interesantes y existe una extensión de HTN-PDDL actualmente en desarrollo que introduce muchas de estas ideas.

El principal inconveniente de estos lenguajes es que ninguno a sido aceptado de forma general por la comunidad de planificación. El lenguaje de SIPE es tremendamente completo pero esto hace al mismo tiempo que su sintaxis sea complicada y los dominios escritos en el lenguaje de SIPE difíciles de leer.

Puesto que ya existe un lenguaje universalmente aceptado por la comunidad de planificación como es PDDL[10] se apostó por partir de ese lenguaje, para el cual ya existían múltiples dominios y todo la comunidad entiende.

10 Están apareciendo lenguajes en un áreas afines a la planificación, como es la de la web semántica y la de gestión de procesos de negocio (Bussiness Process Management (BPM)), apoyados por consorcios de empresas y organismos públicos muy grandes, que están siendo estandarizados (BPEL [14]). Para estos lenguajes existen herramientas visuales muy potentes para la descripción de procesos y su validación. Además son usados en una gran cantidad de plataformas. La comunidad de planificación debería plantearse trabajar en planificadores que fuesen capaces de procesar estos lenguajes.

La primera versión de PDDL [49] incluía soporte para planificadores HTN y aunque se planteaba incluir dominios en la IPC del 2000 finalmente se descartó y las sucesivas versiones del lenguaje no incluyen soporte HTN. El principal problema radica en la falta de una sintaxis y una semántica clara en esta parte del lenguaje y en la falta de una filosofía clara de como modelar dominios con éste lenguaje hasta el punto de que ningún dominio de la IPC se ha escrito con esta extensión.

Han surgido iniciativas, tratando de describir extensiones alternativas [127], pero con poco éxito.

El tema de disponer de una extensión de HTN para PDDL sigue estando abierto entre los miembros del comité de la IPC, pero los planificadores HTN son muy heterogéneos y es difícil encontrar una sintaxis y una semántica común que de cabida a todos ellos.

El lenguaje HTN-PDDL probablemente tampoco será el que adopte el comité de la IPC, si finalmente se consigue una extensión HTN en PDDL, pero nuestra intención es dar un paso en esa dirección, haciendo que el lenguaje sea lo más genérico y sencillo que nos sea posible.

Las mejoras expresivas introducidas en HTN-PDDL que soporta HTNP van orientadas en tres sentidos. Primero un mayor control sobre la búsqueda, segundo una mejor integración del planificador con otros actores, ya sean humanos o otros sistemas automáticos y tercero dar herramientas expresivas para la inferencia temporal. En este capítulo nos centraremos en los dos primeros aspectos, dejando las capacidades temporales para el siguiente capítulo, que se dedica de fondo a ello.

2.9. - Herramientas expresivas para mejorar la eficiencia

Una preocupación siempre importante es que el planificador tenga un tiempo de respuesta aceptable, para poder así hacer frente a problemas de mayor envergadura. Si el planificador toma buenas decisiones obtendrá un plan muy rápidamente. El problema surge cuando el planificador toma una mala decisión que le lleva a realizar mucho bactracking hasta que finalmente vuelve a escoger el camino correcto. El planificador debería ser capaz de darse cuenta lo antes posible de cuales son las mejores decisiones de forma automática o cuando esto no es posible, al menos disponer de herramientas expresivas que, por un lado, permitan al escritor de dominios guiar al planificador hacia buenas soluciones y por otro, en caso de que haya backtracking traten de reducirlo al mínimo.

El éxito de la planificación HTN se basa principalmente en esta segunda

opción, dar al escritor de dominios la posibilidad de que sea él el que oriente al planificador hacia las mejores soluciones.

Los primeros planificadores jerárquicos se basaban en el concepto de *esquemas de reducción de acciones* que eran simplemente una red de tareas por las que se sustituía la tarea abstracta y una serie de críticos [118] [132] decidían cual de los esquemas candidatos era el más adecuado. Con el tiempo la filosofía del modelo HTN ha ido evolucionando hacia representaciones más expresivas y legibles [23] [25]. En cualquier caso para la mayoría de los planificadores HTN el concepto principal de su lenguaje es el método, plantilla o esquema de reducción.

El lenguaje HTN-PDDL centra su representación en base a dos conceptos fundamentales, el concepto de *objeto* y el concepto de *acción.*

El concepto de objeto representa a los elementos del dominio (constantes de PDDL). Todos los objetos heredan por defecto de la clase abstracta "object", aunque pueden ser de una clase más específica. La jerarquía de clases viene marcada por la jerarquía de tipos de PDDL. En PDDL se permite herencia múltiple, de forma que un objeto puede pertenecer a varias clases (no emparentadas por una relación de herencia) de forma simultánea.

Los objetos se relacionan unos con otros a través del concepto de *literal.* Un literal es un predicado que define atributos y relaciones entre los objetos. Se permiten relaciones de cualquier aridad, atributos multievaluados, y estructurados.

Ejemplo:

```
(:objects
    maria – persona
    juan – persona
    mabel – persona
)
(:init
    ;; atributos simples
    (edad maria 27)
    (peso mabel 8)
    ;; relaciones entre objetos
    (casados juan maria)
    (hijo-de maria mabel)
    ;; relación ternaria
    (padres-de mabel juan maria)
    ;; atributo multievaluado
    (nacionalidad mabel españa)
    (nacionalidad mabel argentina)
    ;; atributo estructurado
    (medidas mabel 15 17 15)
)
```

Esta forma de estructurar la información de los objetos en PDDL viene dada por su herencia de la lógica de primer orden. Quizás se podrían encapsular más los objetos y sus propiedades siguiendo el paradigma de la orientación a objetos [111] [13], para la cual existen una gran cantidad de herramientas de diseño y modelado. Estos objetos, sin embargo, a diferencia de lo que ocurre por ejemplo en la programación orientada a objetos, no tienen entidades de procesamiento (funciones o métodos), simplemente porque estas entidades en la planificación tienen un peso más importante y central.

En la planificación automática las entidades de procesamiento son las *acciones* que son el elemento central en el lenguaje HTN-PDDL. Hay que notar sin embargo que el concepto de entidad de procesamiento es distinto en la programación orientada a objetos que en la planificación automática. En la planificación automática las acciones describen, cuando se puede ejecutar una acción y qué efectos tiene, pero no se describe el cómo hacerla (del cómo ya se encargará el actor ejecutor). En la programación orientada a objetos por el contrario se centra en cómo realizar una determinada funcionalidad y los efectos de la misma están implícitos.

Las acciones en HTN se dividen en dos grupos, tareas abstractas y operadores primitivos.

Las tareas abstractas en HTN-PDDL tienen una estructura similar a una clase en un lenguaje de programación. Dentro de una tarea abstracta se encapsulan los parámetros que utiliza y los distintos métodos de descomposición de tareas que puede utilizar. Hay que destacar que en HTN-PDDL el concepto de acción (tarea abstracta) toma un papel más importante que el de él método de descomposición. Cada método de descomposición encapsula la red de tareas por la cual la tarea abstracta se va a sustituir. De esta forma se va construyendo un *árbol de expansión* cuyas raíces son las tareas a descomponer que son objetivo del plan.

Ejemplo:

```
;; Un ejemplo de tarea abstracta en HTN-PDDL
(:task moverse
  :parameters (?quien ?destino)
  (!
    (:method en-destino
     :precondition (= (posicion ?quien) ?destino)
     :tasks ()
    )
    (:method andando
     :precondition (and (bind ?donde (posicion ?quien)) (<= (distancia ?
donde ?destino) (limite_andando ?quien)))
     :tasks (ir_caminando ?quien ?destino)
```

```
    )
    (:method en_taxi
     :precondition ()
     :tasks (ir_en_taxi ?quien ?destino)
    )
  )
)
```

En HTN-PDDL (y en cualquier enfoque HTN) la jerarquía de tareas es asimétrica. Los *operadores primitivos* (hojas en el árbol de expansión) son las únicas acciones que tienen capacidad para realizar cambios efectivos en el estado (cambios en las propiedades y en las relaciones entre los objetos del dominio). Los operadores primitivos en HTN-PDDL tienen una sintaxis muy similar a las acciones del lenguaje PDDL.

Ejemplo:

```
;; Descripción de un operador primitivo en HTN-PDDL
(:action caminar
  :parameters (?quien ?origen ?destino ?camino)
  :precondition (camino ?origen ?destino ?camino)
  :effect (assign (posicion ?quien) ?destino)
 )
```

Las tareas abstractas únicamente codifican la forma en la cual se construye el árbol de expansión y las relaciones de las tareas entre sí. Podemos distinguir dos tipos de relaciones entre las tareas:

- Relaciones de herencia. Codifican la relación tarea padre, tarea hija. Una tarea hija, únicamente puede tener una tarea padre (no hay herencia múltiple).
- Relaciones de orden. Relaciones de precedencia, sucesión con otras tareas del árbol.

Las relaciones de orden entre las tareas se definen en la codificación de la red de tareas, que es una regla que nos dice como sustituir una tarea abstracta por una red de tareas asociadas.

Las relaciones de herencia se establecen mediante un método, que es una regla que dice cuando o en qué condiciones de instanciación de variables asociadas a las tareas del árbol de expansión, es posible realizar la sustitución.

El árbol de expansión que devuelve el planificador cómo solución no es único. El planificador HTN busca en el espacio de árboles de expansión, uno que

sea válido y que esté totalmente expandido (sus hojas sean todas operadores primitivos). Para ello toma una serie de decisiones en cuanto a cómo instanciar las variables y qué métodos de expansión aplicar entre otras. A continuación presentaremos una serie de herramientas que permiten al diseñador de dominios el control de cómo se toman estas decisiones.

2.9.1. - El corte

El corte es una herramienta sintáctica importada del lenguaje PROLOG [20]. PROLOG es un lenguaje de programación genérico que utiliza un mecanismo de inferencia muy similar al que usa un planificador. El concepto de base de datos en PROLOG es similar al de estado en un planificador. Ambos infieren conocimiento nuevo usando reglas descritas en lógica de primer orden y un mecanismo de refutación contra la base de datos o el estado respectivamente.

PROLOG permite utilizar el corte como herramienta de control para cortar el backtracking cuando se refuta una regla. Veámoslo con un ejemplo, supongamos que describimos en PROLOG el siguiente conjunto de reglas, que transforman una nota numérica en su correspondiente textual:

```
nota(Nota,sobresaliente) :- Nota >= 9.
nota(Nota,notable) :- Nota < 9, Nota >= 7.
nota(Nota,bien) :- Nota <7, Nota >=6.
nota(Nota,suficiente):- Nota < 6, Nota > 5.
nota(Nota,insuficiente):- Nota < 5, Nota >= 3.
nota(Nota, muy_deficiente) :- Nota < 3.
```

Supongamos que preguntamos a PROLOG que nota corresponde a un 9.5, y que el algoritmo de inferencia va probando las distintas reglas en el orden en el que están escritas en la base de datos. PROLOG probará con a la primera regla, que será válida y nos dirá que es un “sobresaliente”:

```
?: nota(9.5,X)
X = sobresaliente
retry(Y,n)?
```

Pero ¿qué pasa si intentamos obtener otra solución , o lo que es lo mismo, forzamos un backtracking?. Pues PROLOG realizará 5 disparos de regla más que son innecesarios y finalmente fallará en la inferencia.

Si usamos el operador corte (!) podemos evitar este esfuerzo extra del algoritmo. El operador corte siempre hace que el algoritmo de inferencia tenga éxito al probarlo cuando va hacia adelante y hace que “olvide” el resto de

unificaciones que le quedan por probar, de forma que cuando se vuelve a él por backtracking falla. Por tanto en nuestro ejemplo si describimos ahora la base de datos de la situiente forma:

```
nota(Nota,sobresaliente) :- Nota >= 9,!.
nota(Nota,notable) :- Nota < 9, Nota >= 7,!.
nota(Nota,bien) :- Nota <7, Nota >=6,!.
nota(Nota,suficiente):- Nota < 6, Nota > 5,!.
nota(Nota,insuficiente):- Nota < 5, Nota >= 3,!.
nota(Nota, muy_deficiente) :- Nota < 3,!.
```

Y repetimos la misma consulta, ahora el algoritmo alcanza el corte y olvida el resto de las cinco reglas que le quedan por probar, de forma que cuando digamos que reintente, ahora la respuesta de fallo será inmediata, sin tener que probar ninguna regla más.

El ejemplo de corte que se ha descrito es muy simple, pero se pueden construir reglas mucho más complejas, con una inferencia mucho más costosa, en donde se puede apreciar mejor su potencial.

El corte es una herramienta controvertida porque si es mal usado puede hacer que el algoritmo de refutación no encuentre soluciones que sin él serían válidas. En cualquier caso su uso se deja bajo la responsabilidad del programador que, de esta forma, tiene cierto grado de control sobre la inferencia del algoritmo.

Esta misma idea de corte se puede trasladar al mecanismo de inferencia usado por algunos planificadores (en GRAPHPLAN por ejemplo este mecanismo sería en principio inútil puesto que considera todas las unificaciones de una vez). En concreto el planificador HTNP va realizando las unificaciones de izquierda a derecha según va analizando la cláusula de una precondición de la cual quiere encontrar las unificaciones. Éste conocimiento es útil para el escritor de dominios en cuanto puede utilizarlo para colocar las fórmulas que tienen más probabilidad de fallar en su refutación más a la izquierda, evitando de esta forma inferencia innecesaria de las fórmulas mas a la derecha que son más probablemente refutables. El corte igualmente se puede utilizar siempre que sea posible para reducir el número de puntos de backtracking.

Otro corte distinto, basado en la misma idea es el corte de métodos en una tarea abstracta. Este corte también puede ahorrar mucha inferencia si tenemos conocimiento de que los métodos son mutuamente excluyentes.

```
Ejemplo:
;; Un ejemplo de tarea abstracta con corte
(:task moverse
  :parameters (?quien ?destino)
  (!
```

```
    (:method en-destino
     :precondition (= (posicion ?quien) ?destino)
     :tasks ()
    )
    (:method andando
     :precondition (and (bind ?donde (posicion ?quien)) (<= (distancia ?
donde ?destino) (limite_andando ?quien)))
     :tasks (ir_caminando ?quien ?destino)
    )
    (:method en_taxi
     :precondition ()
     :tasks (ir_en_taxi ?quien ?destino)
    )
  )
 )
```

En esta tarea abstracta un turista desea desplazarse a un destino en concreto. Existen tres métodos distintos que codifican alternativas excluyentes, en la opinión del diseñador de dominios. Si el turista ya se encuentra en el destino no necesita desplazarse, en otro caso puede desplazarse en taxi si la distancia es mayor de un determinado umbral o en otro caso ir andando (se ha considerado que si la distancia es mayor que el umbral siempre se va en taxi y en otro caso siempre se va andando). El escritor de dominios juega con el hecho de que conoce el orden en el que el planificador prueba los métodos para no calcular de nuevo condiciones que ya han sido probadas. Cuando la precondición de un método es cierta y se realiza la expansión de la red de tareas asociada, el planificador automáticamente descarta el probar el resto de los métodos, de esta forma se ahorra el tiempo necesario en calcular condiciones y expansiones de métodos innecesarias.

Destacar un par de aspectos interesantes de este ejemplo. Primero, observar que se permite el uso de la red de tareas vacía como expansión de un método, en concreto del método "en-destino" y segundo el uso de la expresión *bind* dentro del método "andando" que es una expresión no incluida en el PDDL estándar pero que es muy útil. Esta expresión permite asignar a una variable libre el contenido de un fluent.

2.9.2. - El operador de ordenación

Otra forma posible de controlar el proceso de generación del plan es utilizar el operador de ordenación *sortby*. *sortby* se utiliza para dar una prioridad al orden en el cual se intentan las unificaciones. Este operador no está disponible en PDDL pero es muy útil para por ejemplo seleccionar por cercanía un medio:

```
(:task Enviar_ambulancia
 :parameters (?lx ?ly)
 (:method seleccionar
  :precondition (:sortby ?distancia :asc
                  (and
                    (ambulancia ?ambulancia)
                    (disponible ?ambulancia)
                    (localizacion ?ambulancia ?px ?py)
                    (bind ?distancia (distancia ?lx ?ly ?px ?py))
  )))
  :tasks (
     (Avisar_conductor ?ambulancia)
     (ira ?ambulancia ?lx ?ly)
  )
  ...
```

Hacer esto mismo en PDDL no es posible. Ni si quiera se puede hacer en PDDL 3.0 a pesar de poder ser vista esta característica como un tipo de preferencia y ser bastante corriente en problemas reales.

2.9.3. - *Las acciones inline*

Las acciones *inline* son, a efectos del proceso de planificación, acciones primitivas que no tienen nombre, usadas para introducir información de control en el estado del planificador. En el estado del planificador podemos distinguir dos tipos de literales. Los literales que representan el estado del universo en cada momento y otros literales que son puramente de procesamiento y que son utilizados por el planificador para ahorrar cálculos o guiar la búsqueda. Las acciones *inline* al igual que las acciones primitivas tienen una precondición que debe ser cierta para su ejecución y un efecto asociado que altera el estado. Sin embargo, las tareas *inline* no deben modificar el estado del universo.

Podemos distinguir dos tipos de acciones inline:

- Las "*:inline*": Estas acciones si sus precondiciones fallan, fallarán como una acción normal y producirán backtracking.
- Las "*:!inline*": Estas acciones no producen fallo, ni provocan backtracking aunque sus precondiciones sean falsas. Si la precondición es cierta la acción *inline* aplicará los efectos, en otro caso no hará nada. Observar que esta construcción es muy similar a tener una acción *inline* normal sin precondiciones y con un efecto condicional.

Las acciones *inline* no se incluyen en el plan resultante puesto que no tienen

la semántica de una acción primitiva real.

Un ejemplo en el que se pueden usar acciones inline es para por ejemplo realizar una serie de tareas un determinado número de veces, dependiendo del estado actual:

```
Ejemplo:
;; esquema de tarea iterativa
(:task Hacer_i_veces
   :parameters (?i -number)
   (:method inicializa
    :precondition (not (contador ?x))
    :tasks (
         (:inline () (contador ?i))
         (Hacer_i_veces)
   ))
   (:method bucle
    :precondition (> (contador ?x) 0)
    :tasks (
         (:inline  () (decrease (contador ?i) 1))
         (Hacer_esto)
         (Hacer_i_veces)
    ))
   (:method fin
    :precondition (<= (contador ?x) 0)
    :tasks (
         (:inline () (not (contador ?x))
   ))
)
```

Las tareas *inline* son también utilizadas para introducir información temporal como veremos en los próximos capítulos.

2.9.4. - *La cláusula maintain*

La cláusula *maintain* no es una herramienta expresiva para mejorar la eficiencia del proceso de planificación pero si es una herramienta de control del proceso de planificación. Se utiliza para proteger un literal de ser retirado o modificado el estado mientras no se levante su protección. De esta forma cualquier acción que intente modificar el literal mantenido fallará y se producirá backtracking. Esta herramienta es utilizada por el escritor de dominios para garantizarse que un literal crítico solamente se modifica por unas acciones del dominio y bajo unas circunstancias controladas.

PDDL 1.1 [49], que incluye soporte HTN tiene una cláusula similar para mantener el valor de verdad de un predicado durante toda la expansión de una red de tareas. La cláusula *maintain* de HTN-PDDL es mucho más fuerte ya que la

protección debe ser explícitamente retirada y se permite que se mantenga entre expansiones de redes de tareas (contextos) distintas. El maintain de PDDL es por el contrario mucho más genérico ya que permite el mantenimiento de cualquier precondición no de unos literales en concreto, pero garantizar su consistencia también requiere mucho más esfuerzo por parte del planificador.

El siguiente ejemplo muestra como declarar un literal maintain y como eliminar la protección.

```
Ejemplo:
;; Una pareja de operadores primitivos que protegen/desprogeten un
;; literal
(:action proteger
 :parameters (?x)
 :precondition ()
 :effect (:maintain (maintained ?x)))

(:action desproteger
 :parameters (?x)
 :precondition ()
 :effect (not (:maintain (maintained ?x))))
```

2.10. - Herramientas expresivas para mejorar la integración

Un planificador prácticamente nunca va a trabajar de forma aislada cuando se usa para la resolución de problemas no puzzle. Como el planificador va a tener que comunicarse con otros actores humanos o otros sistemas, es importante que se disponga de un lenguaje con capacidades expresivas para facilitar la integración.

2.10.1. - *Expresiones en otros lenguajes*

El lenguaje PDDL es un lenguaje de propósito específico que a veces no da toda la expresividad necesaria para hacer ciertos cálculos. Tampoco tiene herramientas para hacer llamadas o consultas externas.

Este problema no es nuevo, la mayoría de los planificadores están escritos en LISP y se apoyan en el parser de LISP para leer los dominos y los problemas, cuando necesitan alguna expresividad extra que no tiene su lenguaje se apoyan en LISP para conseguirla.

En HTN-PDDL se trata de conseguir una funcionalidad similar mediante la invocación de lenguajes de scripting. El lenguaje de scripting o el mecanismo de comunicación con este debe ser soportado por el planificador.

HTNP utiliza como lenguaje de scripting el lenguaje Python[11]. Python es un lenguaje de programación general orientado a objetos de uso muy extendido debido a la simplicidad de su sintaxis (es un lenguaje interpretado con tipos dinámicos), la facilidad para escribir prototipos rápidos, la enorme cantidad de librerías de código abierto existentes para este lenguaje y a su fácil integración con programas escritos en otros lenguajes.

HTN-PDDL permite la invocación de estos scripts al evaluar funciones o axiomas. De esta forma las funciones *fluents* pasan de ser simples contenedores de números a comportarse como verdaderas funciones que realizan cálculos.

El paso de parámetros se realiza utilizando las variables de la propia función. El valor devuelto se coge del devuelto por la cláusula return. Por ejemplo la siguiente función (fluent) calcula la distancia entre dos puntos:

```
Ejemplo:
;; función que calcula la distancia entre dos puntos
(distance ?x1 ?y1 ?x2 ?y2)
{
 import math
 return math.sqrt(math.pow((?x2 - ?x1),2) + math.pow((?y2 - ?y1),2))
}
```

Observar que el código del script se pone entre llaves. Si se quisiese codificar esta misma función como *fluents* normales de PDDL, habría que introducir los cálculos de todas las distancias posibles en el dominio. Observar que en PDDL es imposible hacer estos cálculos con *fluents* en otro sitio que no sean los efectos. En HTN-PDDL si se podrían hacer estos cálculos en las precondiciones mediante el uso de la cláusula *bind,* pero aún así, el cálculo es más lento que con el script.

```
Ejemplo:
;; fluents para calcular la distancia en PDDL estandar
(:init
    ((distance 2 2 2 2) = 0)
    ((distance 1 1 2 2) = 2)
    ((distance 1 2 2 2) = 1)
    ....
)

Ejemplo:
;; Uso de bind para calcular la distancia en una precondición en HTN-PDDL
(:action ir-andando
 :parameters (?p -persona ?l -lugar)
 :precondition (and
                 (posicion ?p ?px ?py)
                 (posicion ?l ?lx ?ly)
                 (maximo_andando ?max)
```

11 http://www.python.org

```
                (< ?max (sqrt (+ (* (- ?px ?lx) (-?px ?lx))
                                 (* (- ?py ?ly) (-?py ?ly)))))
             )
:effect (and
            (not (posicion ?p ?px ?py)
            (posicion ?p ?lx ?ly)
        )
```

El uso del lenguaje de scripting para realizar cálculos matemáticos es el ejemplo más inmediato, pero se pueden realizar tareas mucho más complejas (Python es un lenguaje con mucha flexibilidad en este sentido):

- Consultar una base de datos.
- Invocar un servicio web.
- Hacer una búsqueda en una página web de internet y recibir información.
- Escribir información de progreso en un fichero o en la pantalla.
- Invocar un algoritmo A* para el cálculo de la ruta óptima.
- Mostrar diálogos al usuario, con petición de información en tiempo de planificación.

El siguiente ejemplo muestra como hacer una consulta a una base de datos mysql:

```
(calcula_precio ?p_id)
{
import MySQLdb

mydb = MySQLdb.Connect(db='almacen')
cursor = mydb.cursor()
stmt = "select pvp from productos where id = " + str(?p_id)
cursor.execute(stmt)
resultSet = cursor.fetchall()
pvp = resultSet[0][0]

mydb.close()
return pvp
}
```

2.10.2. - *Incluir metainformación*

PDDL en particular y los lenguajes de descripción de dominios en general se

centran como es lógico en codificar la información necesaria para realizar la planificación, pero existe información no útil para el proceso de planificación, pero que es interesante que se encuentre en el plan final como información adicional o como información necesaria para una herramienta que postprocese el plan.

La metainformación en HTN-PDDL se introduce como una sección *meta* en operadores primitivos, tareas abstractas y métodos. El funcionamiento básico es muy simple, existen una serie de etiquetas a las cuales de les asocia una cadena de texto. Estas cadenas de texto pueden incluir variables que serán sustituidas por sus correspondientes valores. El siguiente ejemplo muestra el uso de estas etiquetas meta.

```
(:durative-action board
     :parameters (?p - person ?a - aircraft ?c – city)
     :meta (
        (:tag prettyprint "?start > Embarque de ?p en ?c, vuelo ?a")
        (:tag short "Embarque de ?p")
        (:tag resource "?a")
        (:tag monitoring "yes")
        (:tag cost "HIGH")
    )
     :duration (= ?duration (boarding-time))
     :condition (and (at start (at ?p ?c))
    (over all (at ?a ?c)))
     :effect (and (at start (not (at ?p ?c)))
    (at end (in ?p ?a))))
```

Como se ve la tarea *board* es etiquetada con distinta información. La tarea tiene una descripción larga y otra corta para hacerla más legible a los usuarios finales. Se le indica al monitor del plan que la monitorice y se indica su costo y los recursos que utiliza. Toda esta información puede ser utilizada en un postprocesamiento del plan.

El metatag prettyprint tiene un tratamiento especial en HTNP. Siempre que este tag exista se utiliza en la impresión del plan. Por ejemplo en la impresión de la tarea de embarque del plan anterior por pantalla se mostraría algo similar a lo siguiente:

```
...
17/05/2007 11:15:00 > Embarque de pedro_luque en madrid, vuelo ib510
...
```

El uso de una representación alternativa a las acciones, a la hora de mostrarlas al usuario, no es algo nuevo pero es muy importante que cualquier lenguaje de planificación y cualquier plan lo contemple. OPLAN [23] por ejemplo incluye una construcción llamada *expands* dentro de sus esquemas que hace algo

similar.

```
schema puton;
    vars ?x undef, ?y undef, ?z undef;
    expands {put ?x on top of ?y};
    only_use_for_effects
          {on ?x ?y} = true,
          {cleartop ?y} = false,
           {on ?x ?z} = false,
           {cleartop ?z} = true;
    conditions only_use_for_query {on ?x ?z} = true,
               achievable {cleartop ?y} = true,
               achievable {cleartop ?x} = true;
endschema;
```

HTNP es un planificador que puede obtener las salidas de sus planes en formato XML. Este formato de salida es ideal para los metatags que pueden representarse como tags de XML. El siguiente ejemplo muestra como sería la descripción XML de la tarea de embarque comentada en los anteriores ejemplos:

```
<primitive name="board" id="123" indx="144" start="16:05:2007 11:15:00"
end="17/05/2007 11:30:00" duration="15" start_point="18" end_point="19">
         <meta name="prettyprint">
         27/01/2006 08:11:00 &gt; Embarque de pedro_luque en madrid, vuelo
ib510
         </meta>
         <meta name=”short”>Embarque de pedro_luque</meta>
     <meta name=”resource”>ib510</meta>
     <meta name=”monitoring”>yes</meta>
     <meta name=”cost”>HIGH</meta>
         <parameters>
            <parameter pos="0" name="p">
               <constant name="perdor_luqye" id="125">
                  <type name="person" id="97">
                  </type>
               </constant>
            </parameter>
...
```

2.11. - El modelo de planificación en el sistema SIADEX

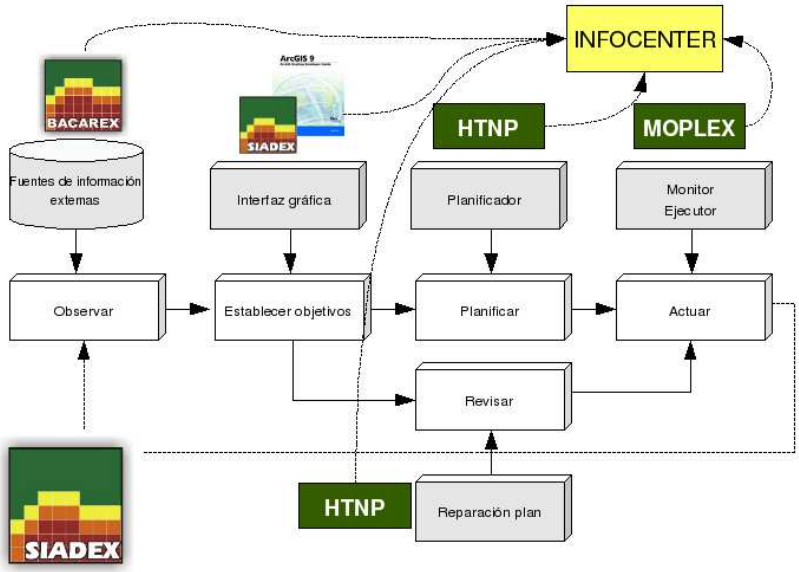


Ilustración 11: Arquitectura del sistema SIADEX

Una vez que hemos presentado el modelo general de planificación, se estudiará como este se a adaptado al caso concreto del sistema SIADEX (Ilustración 11).

El sistema SIADEX está compuesto por una serie de programas software autónomos, escritos en distintos lenguajes (actualmente Java, C++, Visual Basic, Python y PHP) que actúan como servicios web. La arquitectura está muy orientada a permitir la escalabilidad y el intercambio de módulos. Los módulos son coordinados a través de un módulo central llamado INFOCENTER.

2.11.1. - *Módulo INFOCENTER*

El módulo INFOCENTER [36] se basa en una arquitectura publicar suscribir, donde cada uno de los módulos publica sus capacidades y la información procesada cuando está disponible y se suscribe a la información que necesita para su funcionamiento. Los módulos se comunican entre sí utilizando el protocolo XML-RPC [30] (XML *Remote Procedure Call*), lo que permite su distribución en distintos

ordenadores y la comunicación con otros servicios web.

2.11.2. - *La base de conocimiento BACAREX*

BACAREX es la base de conocimiento que trata de aglutinar en un único servicio toda la información necesaria para realizar la planificación. Inicialmente el servicio estaba planteado como una ontología que unificara todos los sistemas del plan INFOCA y que tuviera todo el conocimiento tanto de los objetos del dominio como de las acciones (llamadas protocolos de actuación por los técnicos de extinción). Finalmente se constituyó como un repositorio de información centralizado que guardaba toda la información del plan INFOCA, ampliando la funcionalidad para la que inicialmente fue diseñada.

BACAREX está construida sobre el modelo de datos de Protégé[12] [103] que es una herramienta para el diseño y gestión de ontologías. La comunicación con la API de BACAREX se realiza a través de XML-RPC. El objetivo subyacente es que BACAREX sirva tanto como herramienta para el ingeniero de conocimiento que tenga que integrar nueva información, cómo para los operadores del plan INFOCA que utilizan el sistema.

BACAREX almacena una gran cantidad de información[13] que se organiza en base a una jerarquía de conocimiento de la que se muestra una pequeña parte en la Ilustración 12.

- **Información sobre medios:** El conocimiento principal se centra en los medios disponibles. Mientras que hay medios que son muy estáticos, como por ejemplo las bases (CEDEFOS Centros de defensa forestal) o los aeropuertos, hay otros medios que son muy dinámicos y cuyos atributos cambian a diario (grupos, vehículos aéreos y terrestres) cuya información tiene que estar perfectamente actualizada para el correcto funcionamiento del planificador. Hay atributos que van incluso cambiando conforme un plan de extinción se va ejecutando, como la posición del medio, el trabajo que está realizando, y la carga de combustible, agua, o extintor que tiene disponible.

12 http://protege.stanford.edu

13 Actualmente BACAREX tiene unas 130 clases y más de 2000 instancias de objetos. Hay que destacar que solamente en medios de lucha contra incendios tenemos recogidos unos 300 grupos de actuación repartidos por toda la comunidad (unos 6 u 8 retenes por grupo, dependiendo del medio que los transporte), 200 medios terrestres (todoterrenos, autobombas, camiones, bulldozers...) y 30 vehículos aéreos. Para la presente campaña del 2007 se espera que su número aumente.

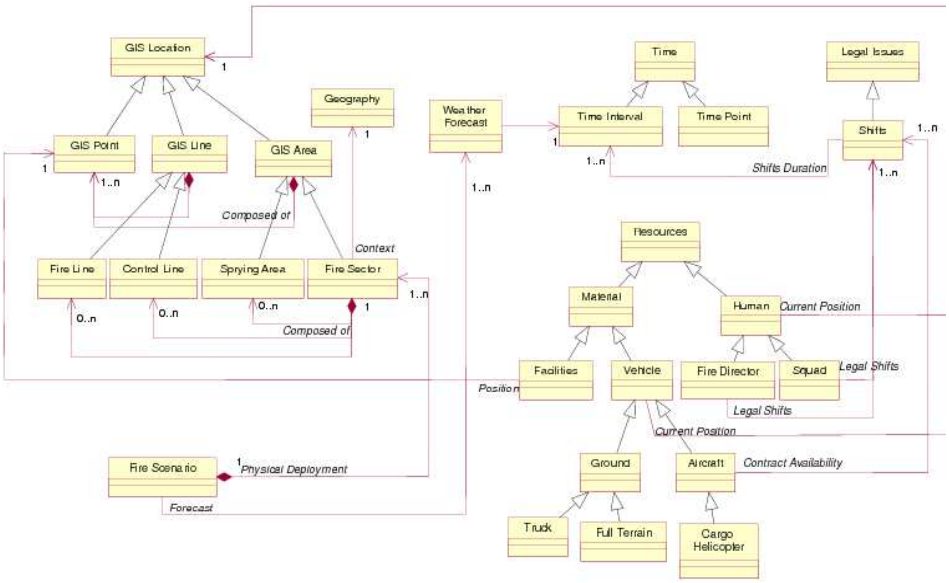


Ilustración 12: Jerarquía de conocimiento simplificada en BACAREX

- **Información temporal**: La información de tipo temporal es fundamental e impregna toda la ontología. Hay eventos que ocurren de forma periódica como el día y la noche o los turnos de los grupos o agentes de extinción (aunque estos cambian en cuanto actúan en un incendio). También hay eventos que ocurren de manera aleatoria (la aparición de un incendio, el cambio en las condiciones meteorológicas). Y eventos que ocurren durante un intervalo de tiempo (periodo de subcontratación de un medio aéreo).

- **Información geográfica:** Se caracteriza por tener unas coordenadas. Las coordenadas pueden hacer referencia a un único punto (la posición de un vehículo o de una base), o a varios puntos que pueden cerrar una figura poligonal o formar una recta (el área que ocupa un incendio, o una línea de defensa).

La **Información procedural,** representa los protocolos de extinción del plan INFOCA. Estos protocolos se describen como una jerarquía de tareas descritas en el lenguaje HTN-PDDL. Esta información está disponible en el servicio BACAREX aunque no se almacena de forma directa en la ontología.

Queda otro tipo de información crucial para la elaboración del plan que se actualiza de forma diaria (previsión meteorológica) o anual (puntos de recarga de

agua).

Para la actualización, consulta y mantenimiento de toda esta información la ontología de BACAREX tiene dos formas de acceso (directa) para dos perfiles de usuario distintos:

- **Ingenieros del conocimiento:** Como ya se ha comentado BACAREX está diseñada para integrar conocimiento de múltiples fuentes de información. Cuando aparece una nueva fuente de información debe existir un mecanismo fácil para poder integrarla. Protégé es una herramienta que permite la gestión, consulta y actualización de ontologías de una manera muy sencilla. Además dispone de una gran cantidad de herramientas para la importación y exportación de ontologías en otros formatos como OWL [31] [8] o DAML+OIL. Puesto que BACAREX está construida sobre el modelo de datos de Protégé el ingeniero de conocimiento puede aprovechar esta herramienta para realizar este tipo de tareas (Ilustración 13).

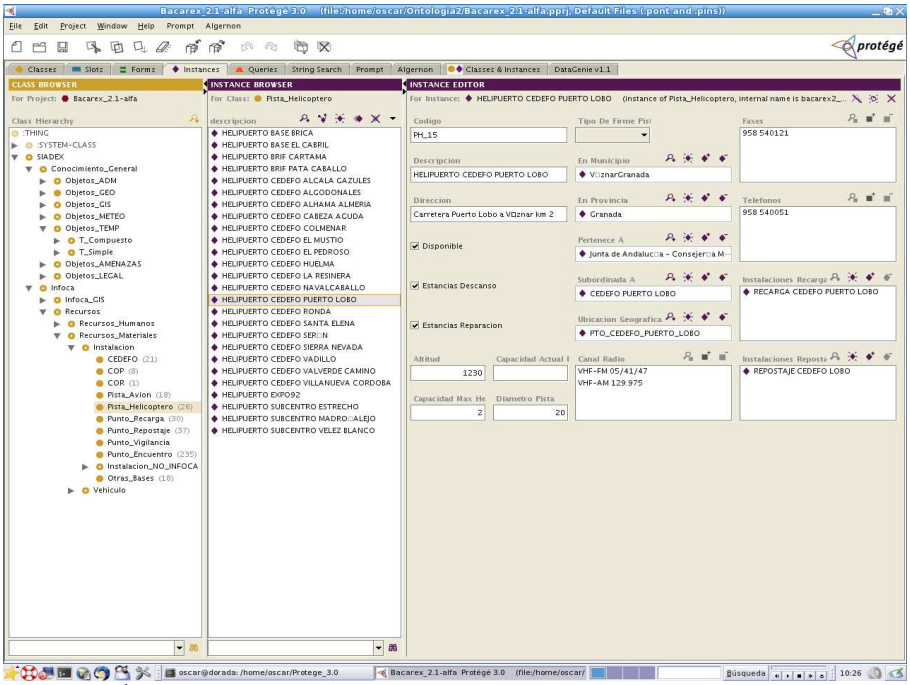


Ilustración 13: Ontología BACAREX en Protégé

- **Operadores:** Por otro lado están los operadores del plan INFOCA que mantienen actualizada la base de datos. No parece muy recomendable que estos usuarios tengan el mismo acceso a la base de conocimiento a través de Protégé que los ingenieros del conocimiento. Por otro lado uno de los objetivos principales del sistema SIADEX es que la información sea accesible desde cualquier lugar y plataforma de la forma más fácil posible. Por ello se ha construido una plataforma web sobre el servicio de web de BACAREX para la modificación y consulta de la información (ver Ilustración 14). Esta plataforma trata de ser poco exigente en cuanto a ancho de banda necesario y en cuanto a requisitos del navegador, de forma que puede ser visualizada desde un teléfono móvil o una PDA. Puesto que se tiene que dar servicio a un buen número de usuarios el modelo de datos de Protégé es guardado en una base de datos MySQL de forma que se mejora el acceso concurrente y la eficiencia de las consultas.

Es muy importante que la información se encuentre actualizada para que el planificador de planes correctos, pero el planificador tiene su propio lenguaje de descripción del conocimiento (que en SIADEX es HTN-PDDL), que es diferente del utilizado en la representación de BACAREX.

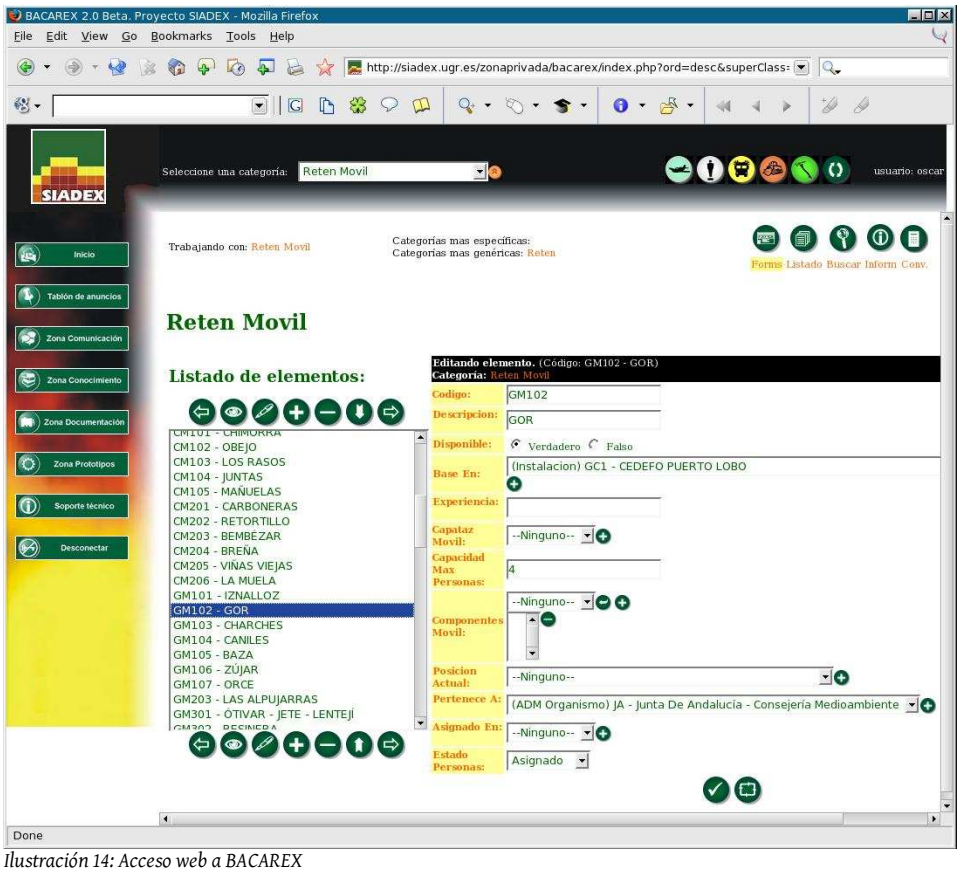


Ilustración 14: Acceso web a BACAREX

En la plataforma SIADEX existen dos pequeños servicios que hacen de pasarela y consultan la información de la ontología y obteniendo un fichero de dominio y de problema descritos HTN-PDDL. Estos servicios son el getProblem y el getDomain.

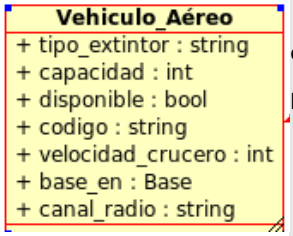
getDomain es el servicio que construye las distintas partes del fichero de dominio. getProblem hace lo propio con el fichero de problema extrayendo el objetivo y el estado inicial. En general los ficheros de dominio y problema se costruyen de la siguiente forma:

- Se hace una traducción directa de las tareas abstractas y de los operadores primitivos que están ya almacenados en formato HTN-PDDL.
- La jerarquía de tipos que es utilizada por los literales y las acciones

se construye recorriendo la jerarquía de clases almacenada en BACAREX, por ejemplo para la jerarquía mostrada en la Ilustración 12, se construiría una jerarquía de tipos como la siguiente:

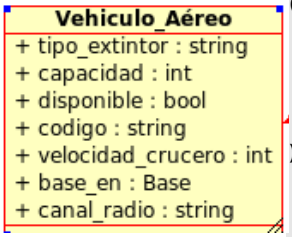
```
(:types
    resources – object
    material humano – resources
    facilities vehicles – material
    ground aircraft – vehicles
    truck full_terrain – ground
....
)
```

- La definición de los literales se obtiene recorriendo las clases y accediendo a su estructura. Para cada atributo del objeto de una clase se construye una definición de literal, atendiendo a su metainformación (tipo, cardinalidad). Los literales instanciados se obtienen recorriendo los objetos de la ontología.



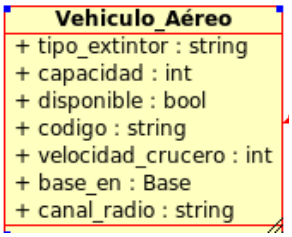

```
(:predicates
    (tipo_extintor ?cod – Vehiculo_Aereo ?
extintor - object)
    (disponible ?cod – Vehiculo_Aereo ?b -
boolean)
    (base_en ?cod – vehiculo_aereo ?b - Base)
   ...
)
(:functions
    (velocidad_crucero ?cod - Vehiculo_Aereo)
– number
    (capacidad ?cod - Vehiculo_Aereo ?y –
number)
    (canal_radio ?cod - Vehiculo_Aereo ?y –
string)
   ...
)
```

- Los literales binarios más comunes son del tipo *(nombre_atributo identificador_objeto valor_atributo)* que describen propiedades del objeto y su traducción es prácticamente directa.



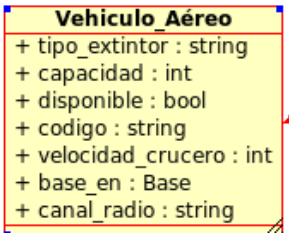

```
(:init
    (= (capacidad A1) 1500)
    (tipo_extintor A1 agua)
    (disponible A1 true)
    (= (velocidad_crucero A1) 300)
   ...
)
```

- Hay predicados que codifican relaciones con otros objetos del tipo *(nombre_atributo identificador_objeto identificador_objeto2).* Las relaciones pueden ser binarias cómo las del ejemplo o más complejas (ternarias, cuaternarias...), para lo cual se necesita de un esquema de traducción concreto que también se especifica en la ontología.




```
(:init
    (base_en A1 AP02)
  ...
)
```

- Hay valores de atributos que pueden tomar valores múltiples, éstos al generarse el problema producirán una lista de literales instanciados de la forma *(nombre_atributo identificador_objeto valor1), (nombre_atributo identificador_objeto valor2)...*




```
(:init
    (canal_radio A1 VHF-FM-03.70.99)
    (canal_radio A1 VHM-AM-123.425)
  ...
)
```

- Los literales que se construyen a partir de atributos numéricos son automáticamente construidos como *fluents* de HTN-PDDL (capacidad o velociad_crucero en el ejemplo).

- Los literales que están marcados como temporizados se construyen como *timed initial literals* de HTN-PDDL[14].

- El problema es extraído de los objetivos que el técnico en extinción marca a través de la interfaz gráfica y que se encuentran almacenados en unas clases conocidas de la ontología.

14 Todavía no se han presentado las capacidades temporales del planificador HTNP El sistema SIADEX utiliza dominios temporizados. Presentaremos las capacidades temporales del algoritmo de planificación se presentarán en los capítulos 3 y 4.

Debido a la gran cantidad de información getProblem es un proceso costoso. Para evitar la extracción de información innecesaria para resolver el problema en cuestión (por ejemplo información asociada a otro incendio distinto) getProblem utiliza una serie de heurísticas. De esta forma también se descarga al proceso de planificación de gestionar información innecesaria en el problema, que ya de por sí es suficientemente grande.

2.11.3. - *Interfaz gráfica*

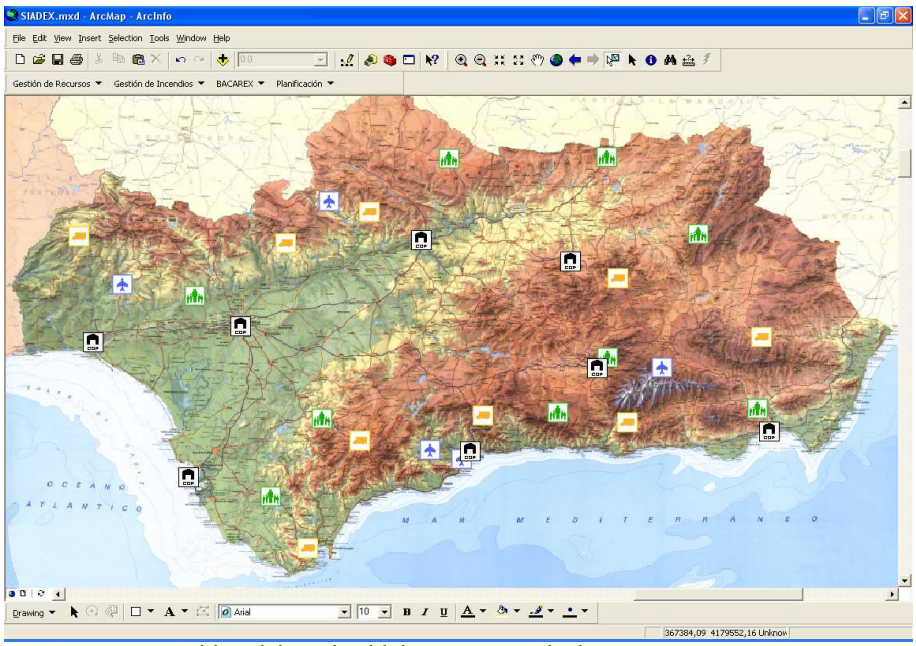


Ilustración 15: Disposición del estado los medios del plan INFOCA extraídos de BACAREX

La interfaz gráfica es quizá una de las partes menos interesante para un investigador en planificación (a pesar de que existe también una gran cantidad de investigación en el área), sin embargo es sin duda la parte más importante para la implantación de cualquier sistema [120], ya que es la parte con la que el usuario tiene un contacto directo (los procesos internos y toda la investigación en la que se apoya no son visibles para él).

La interfaz gráfica con la que los técnicos en extinción trabajan en SIADEX está construida como una extensión de la herrameinta ArcMap de ArcGIS[15] de ESRI.

15 http://www.esri.com/software/arcgis/

ArcMap es una herramienta para la gestión de información de tipo geográfico. La información georeferenciada (mapas, fotografías aéreas o cualquier otra información) es visualizada en la pantalla como distintas capas que los técnicos pueden superponer. Los técnicos en extinción usan mucho esta herramienta en su trabajo diario y están muy familiarizada con ella. Puesto que para los técnicos es muy importante reducir el número de aplicaciones con las que tienen que interactuar y el tipo de problemas a los que se enfrenta SIADEX tiene una gran cantidad de información de tipo geográfico se hacía imprescindible integrarse dentro de ArcMap.

El objetivo principal de la interfaz gráfica además de que el técnico pueda consultar en tiempo real el estado de los medios (al ser esta un servicio web más dentro de la arquitectura general) (ver Ilustración 15) es que este introduzca los objetivos estratégicos para que el planificador elabore el plan de extinción.

El técnico que puede utilizar una herramienta de simulación para estudiar cómo puede ser la evolución del incendio y presentarla dentro del propio ArcMap, puede dibujar incluso sobre el resultado de la simulación cuales son van a ser los sectores de actuación, las líneas de defensa que desea levantar y en que zonas van a actuar los medios aéreos, tal y como se aprecia en la Ilustración 16.

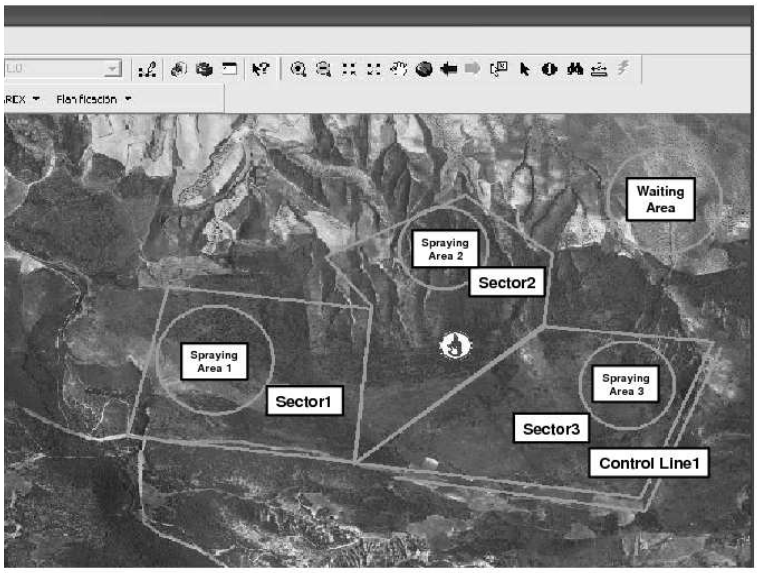


Ilustración 16: Un escenario simulado de incendio en ArcMap

En el ejemplo de la Ilustración 16 el técnico ha definido 3 sectores en el incendio y 3 zonas de descarga de vehículos aéreos una por cada sector. Ha establecido una línea de defensa en el sector oeste y ha colocado el centro de mando en una explanada en la cima de una montaña donde probablemente existan buenas comunicaciones.

Una vez hecho el despliegue geográfico el técnico establece una serie de actividades de tipo genérico que se deben realizar para ese objeto geográfico, utilizando una serie de menús y formularios (Ilustración 17).

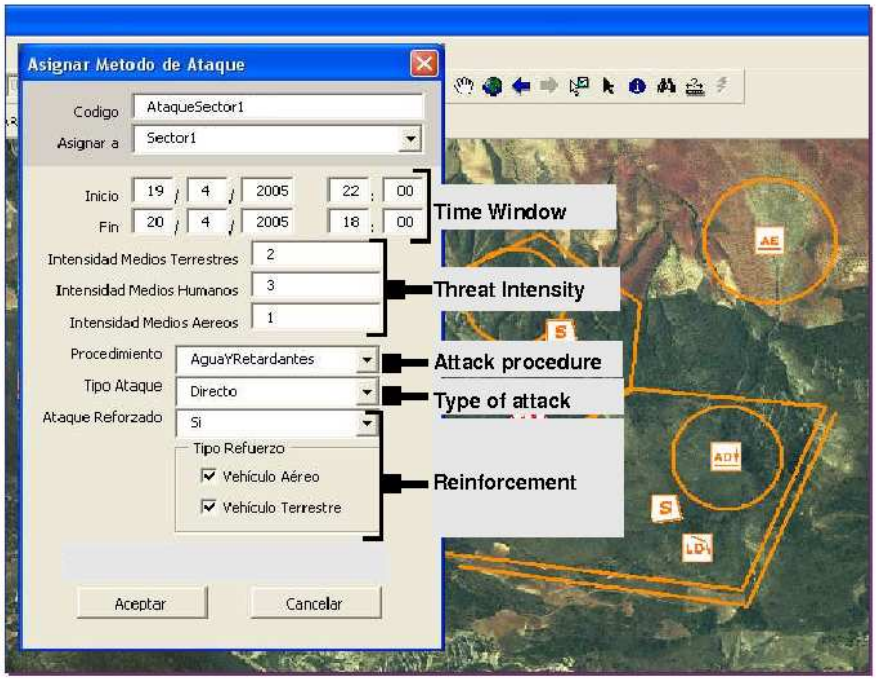


Ilustración 17: Introducción de la estrategia para un área geográfica en la extensión de SIADEX para ArcMap

El técnico establece la ventana de tiempo durante la cual quiere realizar un procedimiento en concreto y la cantidad de medios de distinto tipo que quiere dedicar a esa tarea (el planificador se encarga de asignarlos) así como otros parámetros de alto nivel.

Estas actividades son guardadas dentro de la ontología BACAREX para la posterior construcción del objetivo, como una red de tareas a resolver por el planificador.

El técnico puede iniciar el proceso de planificación (que se realiza en un servidor de planificación distinto como se explicará a continuación), simplemente pulsando un botón.

El plan resultante es presentado de vuelta al usuario en la propia aplicación ArcMap como un cronograma de acciones, o también puede visualizarse en Microsoft Excel, en Microsoft Project gracias a un traductor de los planes devueltos por HTNP al formato de Microsoft Project (Ilustración 18), o en una aplicación creada exclusivamente para el personal del plan INFOCA llamada la INFOMANTA (Ilustración 19). La INFOMANTA muestra una matriz, donde las columnas representan sectores en el incendio y en las filas se encuentran ubicados los medios. La idea está basada en una manta real con bolsillos que los técnicos cuelgan en el centro de mando. En cada bolsillo colocan una tarjeta con un código de colores que representa un medio (blanco agentes, verde grupos, amarillo terrestres, marrón maquinaria, azul aéreos). En la tarjeta describen toda la información acerca del medio y las labores que está desempeñando en el sector.

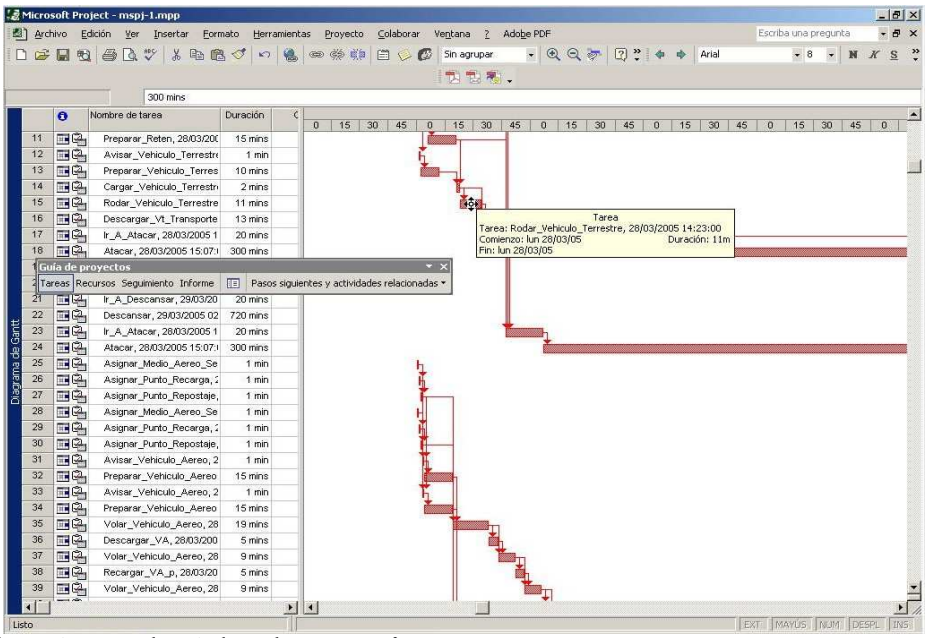


Ilustración 18: Visualización de un plan en Microsoft Project

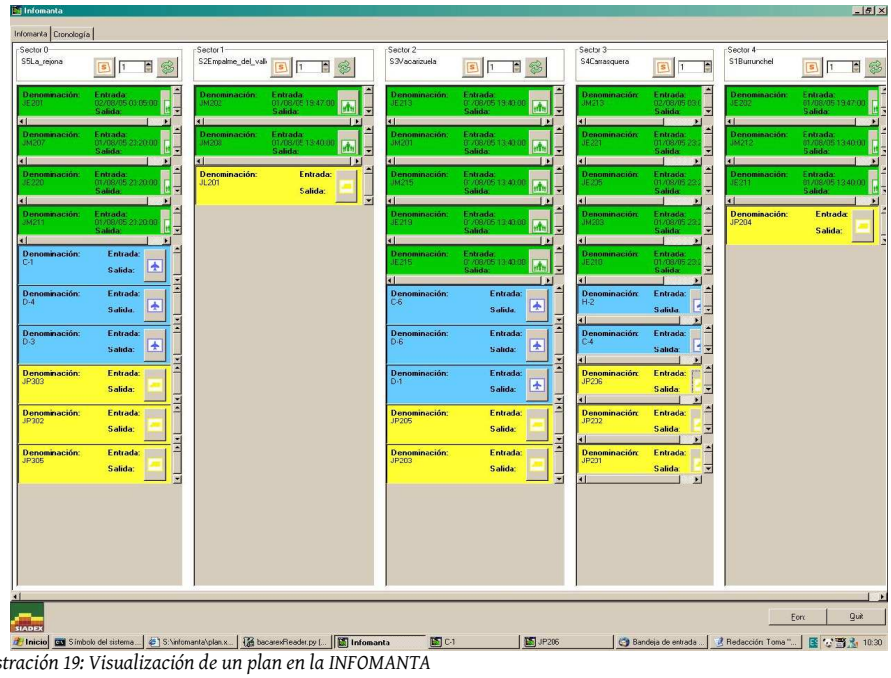


Ilustración 19: Visualización de un plan en la INFOMANTA

2.11.4. - *Planificación*

Como se ha comentado, cuando el técnico en extinción ha terminado de definir los objetivos, simplemente pulsa un botón y lanza el proceso de planificación, pero ese simple proceso involucra a varios servicios. Primero el INFOCENTER recoge la petición y buscar algún servicio que ofrezca servicios de planificación, en SIADEX este servicio es el METASERVER.

METASERVER es un servicio cuyo funcionamiento es muy similar al del proceso portmap en los sistemas operativos UNIX/LINUX. Puesto que se pueden requerir varios servicios de planificación de forma simultánea, METASERVER que siempre está escuchando en un puerto en concreto, recibe una petición de planificación, levanta un proceso/servicio de planificación, que en nuestro caso es el planificador HTNP cuyo funcionamiento se detallará más adelante, asignándole un puerto vacío y devolviendo este puerto al servicio que realizó la petición. El proceso es el siguiente (Ilustración 20):

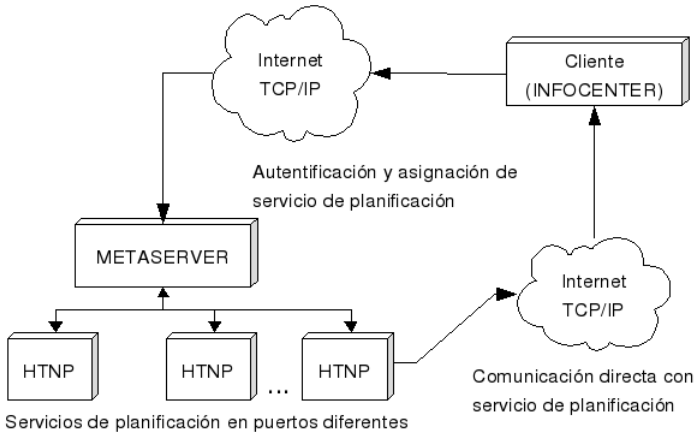


Ilustración 20: Comunicación con METASERVER

1. INFOCENTER realiza una petición de planificación a METASERVER, esta petición normalmente lleva asociado un proceso de autentificación (la comunicación se lleva a cabo a través de un túnel SSH).

2. METASERVER levanta el servicio de planificación devuelve el puerto que es asignado a dicho servicio y una *passphrase* o clave para que HTNP únicamente responda a las peticiones que provengan de ese cliente.

3. INFOCENTER empieza a comunicarse de forma directa con SHTN. La comunicación que se realiza hasta que SHTN termina el proceso de planificación es básicamente de *polling* o testeo del estado.

4. Cuando HTNP termina escribe la información del plan en el INFOCENTER y deja libre el puerto.

2.11.5. - *Ejecución y monitorización*

Una vez que se dispone de un plan y éste ha sido aceptado por el técnico comienza su ejecución y seguimiento. Esta parte del ciclo es al igual que la interfaz gráfica muy dependiente del dominio de aplicación. En el caso de SIADEX a la hora de extinguir un incendio es necesario apoyarse en la jerarquía de mando del plan INFOCA. El plan completo para la extinción del incendio únicamente se tiene que

acceder desde el Centro Operativo Regional en Sevilla, desde el Centro Operativo Provincial de la provincia donde el incendio se ha declarado y desde el Centro de Mando a pie de fuego, por el equipo de técnicos que tienen asignada la dirección de extinción. Aparte cada sector o área en la que se divide el incendio suele tener un agente de extinción que es responsable de las operaciones en ese sector. El responsable por lo tanto solamente necesita acceder a la parte del plan que se ejecuta en su área de trabajo. También hay un responsable o controlador de los medios aéreos. Cada uno del resto de los medios, bases, grupos y vehículos sólo necesitan acceder a las operaciones del plan que les afectan.

El motivo de reducir la visibilidad del plan al mínimo es doble. Por un lado por motivos de seguridad, no es bueno que el plan de extinción sea visible a todo el personal del INFOCA y por otro lado por simplicidad. Al patrón de un grupo de retenes lo único que le interesa es qué tiene que hacer, donde tiene que hacerlo y como llegar hasta allí, el resto de la información para él no es relevante.

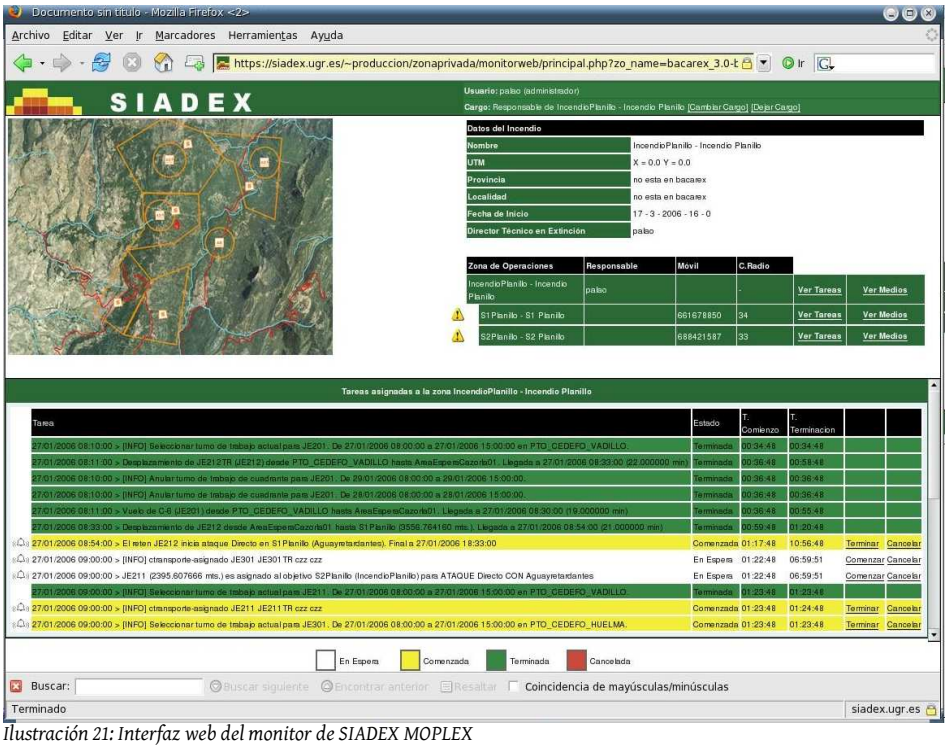


Ilustración 21: Interfaz web del monitor de SIADEX MOPLEX

Teniendo en cuenta estas restricciones en la accesibilidad al plan debemos distribuirlo a los medios involucrados. La filosofía de SIADEX es que la información

sea accesibles desde cualquier lugar en cualquier dispositivo, por lo tanto se ha diseñado un pequeño portal web que necesita de un acceso identificado en el que se muestre la información relativa al plan. En ese portal una vez que se accede se muestra un pequeño resumen de la situación del incendio (sectorización y medios involucrados) y las tareas sobre las que se tiene control dependiendo del ámbito de visibilidad o posición en la jerarquía de mando (ver Ilustración 21).

Los usuarios pueden a través de esta interfaz informar sobre la evolución de las tareas que tienen asignadas introduciendo de esta forma la información en el servicio de monitorización y seguimiento MOPLEX.

MOPLEX es el servicio web con el cual todas las interfaces gráficas se conectan (pasando a través del INFOCENTER) para informar sobre la evolución de las operaciones. MOPLEX informa sobre la correcta ejecución del plan y en caso de fallo debe lanzar el servicio de replanificación.

Pero este modelo no es únicamente adaptable al problema de los incendios forestales, también se puede utilizar en otras aplicaciones.

2.12. - Aprendizaje automático, ADAPTAPLAN

ADAPTAPLAN: Adaptación basada en aprendizaje, modelado y planificación para tareas complejas orientadas al usuario, es un sistema orientado a desarrollar planes educativos en una serie de materias de forma telemática adaptados a un perfil de usuario.

En ADAPTAPLAN nuevamente se puede usar HTNP y la arquitectura de planificación que se ha presentado. El planificador, dado un perfil de usuario, los recursos disponibles en forma de conocimientos adquiridos y materiales disponibles, y los propios objetivos de aprendizaje del estudiante, elabora un plan de estudios para aprender las materias necesarias.

Un sistema de aprendizaje asistido (Learning Management System LMS) se compone de distintos subsistemas:

- Un repositorio de objetos de aprendizaje, que contiene todos los recursos educativos (temarios, vídeos, fotografías, diagramas...) necesarios para la impartición de un curso. Cada objeto se encuentra etiquetado con metainformación, de forma que se describe las características de ese objeto de aprendizaje y su adecuación a distintos perfiles de usuario.

- Una base de datos de perfiles de usuario que contienen información exhaustiva sobre cada usuario, conocimientos previos, experiencia, material disponible, historial académico etc. La anotación de esta

información sigue el estándar IMS-LIP[16].

- Una base de datos con los objetivos de aprendizaje para cada curso que son establecidos por el tutor del curso, de forma que todos los estudiantes que comparten el mismo curso deben alcanzar los mismos objetivos.

- La base de datos de diseño de cursos, que contiene las secuencias de objetos de aprendizaje que un usuario debe de utilizar para realizar el curso, adaptadas a las necesidades/características de cada estudiante.

La Ilustración 22 muestra como podemos integrar un planificador dentro de un LMS, en concreto en ADAPTAPLAN se usa ILIAS[17]. Para ello al igual que ocurre con SIADEX es necesario disponer de un sistema de traducción automática de los objetos de la base de conocimiento del LMS para construir el dominio y el problema de planificación, así como traducir el plan resultante obtenido por el planificador a un curso del LMS.

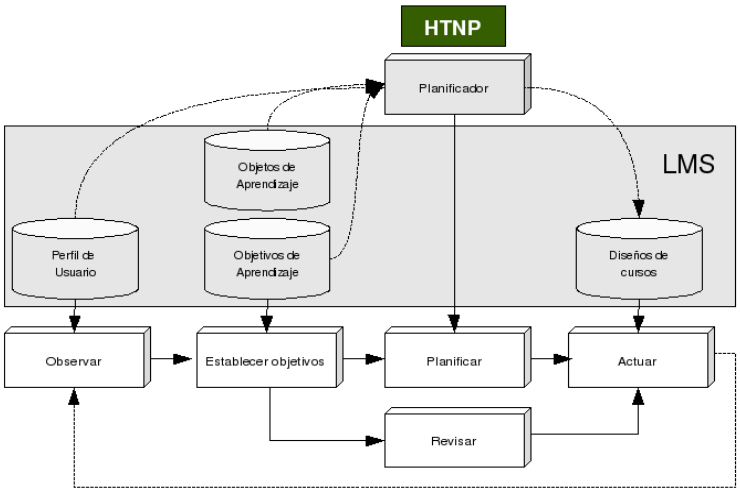


Ilustración 22: Modelo de planificación para ADAPTAPLAN

Los componentes de ILIAS están diseñados como servicios web que pueden ser invocados usando SOAP (Simple Object Access Protocol), por lo que se adapta perfectamente a la filosofía de arquitectura de planificación que presenta al

16 IMS-LIP: IMS Global Learning Consortium Inc, Learner Information Package http://www.imsglobal.org

17 http://www.ilias.de/ios/index-e.html

planificador HTNP como un servicio más.

El incluir un planificador automático como HTNP dentro de la arquitectura es transparente para los distintos usuarios, tanto para los estudiantes como para los tutores que se ven descargados de la tarea de generar cursos personalizados para las necesidades de cada estudiante.

2.13. - Análisis de eficiencia, comparativa HTNP vs SHOP2

Para terminar el capítulo se tratará de demostrar que HTNP es además de un planificador que permite construcciones expresivas complejas y de adaptarse bien al conocimiento de los expertos es un planificador muy eficiente.

Es difícil hacer un análisis comparativo entre planificadores jerárquicos ya que estos suelen tener expresividad bastante distinta y precisamente hacen uso de dicha expresividad para definir heurísticas que mejoren su rendimiento a la hora de resolver un problema en concreto.

Se ha escogido SHOP2 [28] como planificador con el que comparar HTNP por varias razones. Primera aunque los métodos de búsqueda de HTNP y SHOP2 no son exactamente iguales, si comparten un modelo de planificación común y es posible describir un dominio plenamente funcional pero simple y equivalente para SHOP2 y HTNP que no haga uso de construcciones complejas en sus lenguajes. Segundo, SHOP2 ya ha demostrado que es un planificador plenamente solvente en la IPC del 2002, por lo que es un digno rival para HTNP.

Como dominio de pruebas hemos usado una implementación jerárquica muy simple del problema del mundo de los bloques. Aunque el dominio de los bloques es un problema de los llamados tipo puzzle, para los cuales la planificación jerárquica no es especialmente indicada, es muy fácilmente entendible por todo el mundo. La implementación jerárquica del mundo de bloques que usaremos es muy sencilla y utiliza una heurística que se basa en despejar los bloques poniéndolos encima de la mesa. Se ha implementado el dominio en el lenguaje HTN-PDDL y en el lenguaje de SHOP2 de forma que ambos planificadores sigan el mismo camino a la hora de construir los árboles de expansión con la intención de que la comparativa sea lo más justa posible. Por lo tanto ninguno de los planificadores usa heurísticas especiales ni una búsqueda distinta.

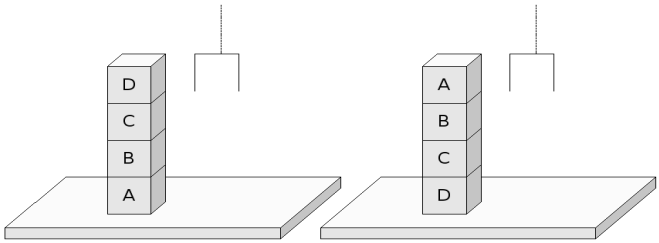
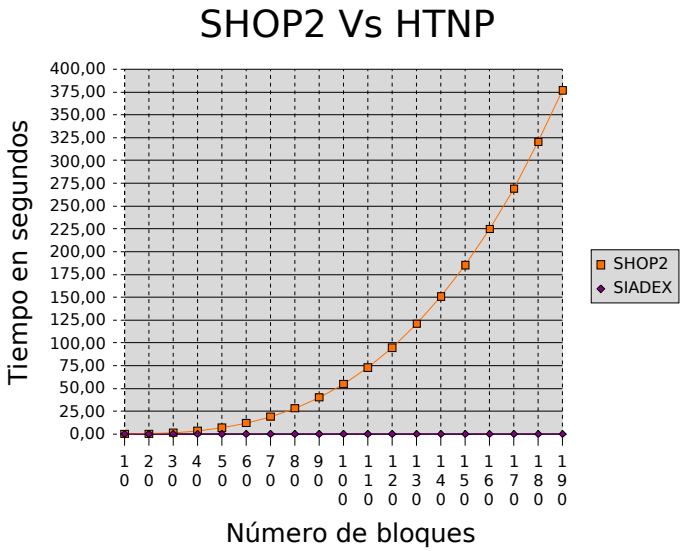


Ilustración 23: Ejemplo de problema en el mundo de bloques

La batería de problemas que se ha preparado consiste simplemente en dada una torre de bloques de diferente tamaño, el planificador tiene que conseguir como objetivo dar la vuelta completamente a dicha torre. La Ilustración 23 muestra el estado inicial y el objetivo de un problema con cuatro bloques.

La gráfica 1 muestra la comparativa en eficiencia de HTNP y SHOP2. En el eje y se muestra el tiempo consumido en segundos mientras que en el eje x se muestra el tamaño del problema en número de bloques que forman la torre. Los experimentos se han realizado en un Intel Centrino 1.3 Ghz con 1GB de RAM. HTNP está escrito en c++ y se ha compilado en Linux con el compilador de la GNU, SHOP2 está escrito en LISP y ha sido compilado en Allegro Common Lisp.



Gráfica 1: Comparativa en eficiencia de shop2 contra HTNP

Se puede ver que en ninguno de los ejemplos HTNP tarda más de un segundo, de hecho para el problema más grande que es de 190 bloques HTNP tarda 0,060 sg mientras que tarda 376,860 sg, lo cual es una diferencia muy considerable y sorprendente dado que como se ha explicado ambos algoritmos realizan la expansión de la misma forma y el único backtracking se produce en la selección de los métodos de expansión.

Se han realizado pruebas con problemas más grandes en HTNP, en concreto para un problema de 2500 bloques HTNP emplea 5.01 segundos, el plan resultante tiene 5000 operadores primitivos y se realizan 15000 expansiones (aplicaciones de métodos).

3. - *Extensión temporal de HTNP el algoritmo HTNP2*

Nuestros objetivos solamente pueden ser conseguidos a través de la vía de un plan, en el cual debemos creer fervientemente, y según el cual debemos actuar de forma vigorosa, no hay otro camino hacia el éxito.

***Pablo R. Picasso**, Pintor y escultor español (1881 - 1973)*

Ser capaz de hacer razonamientos con el tiempo es un aspecto fundamental para cualquier planificador, ya que es necesario obtener planes temporizados en la mayoría de los dominios de aplicación.

Históricamente los planificadores han enfrentado el problema de razonar con el tiempo adoptando distintos modelos:

- Los planificadores que realizan una búsqueda hacia adelante, pueden ir calculando la cantidad de tiempo consumido en cada acción (*makespan).* Suelen utilizar heurísticas con el objetivo de reducir al mínimo el tiempo consumido por el plan [89] [69] [107].

- Otros planificadores utilizan la estructura de grafo del plan (GRAPHPLAN y similares) para razonar con el tiempo [1] [123].

- Por último hay algunos planificadores que utilizan algún tipo de modelo temporal basado en redes temporales [72] [60].

El modelo que mejor se adapta a HTNP y que además ofrece la mayor flexibilidad es el tercero. En este capítulo introduciremos HTNP2, un planificador basado en HTNP que utiliza un modelo temporal para basado en redes de restricciones temporales para realizar razonamientos con el tiempo.

3.1. - Funcionamiento básico del algoritmo temporal

Para manejar el tiempo el algoritmo temporal mantiene una base de datos para cada acción que cambia con el tiempo (base de conocimiento temporal). Esas referencias pueden ser representadas como instantes temporales o intervalos. En cada paso el planificador añade nuevas relaciones entre estas referencias temporales que pueden ser expresadas como restricciones binarias en el modelo temporal. El modelo temporal debe garantizar que la base de conocimiento temporal es siempre consistente. Las relaciones temporales entre las distintas proposiciones las describe implícitamente el escritor de dominios, que de esta forma expresa el momento de tiempo en que una proposición es válida con respecto a otra. Las relaciones como ya se argumentó pueden ser de dos tipos:

- *Relaciones temporales cualitativas:* Este tipo de restricciones establecen el orden relativo entre proposiciones y acciones. Los modelos formales que se pueden utilizar para manejar estas restricciones son el álgebra de puntos y el álgebra de intervalar [5] [6].

- *Relaciones temporales cuantitativas:* Este tipo de restricciones son numéricas. Estas restricciones se pueden manejar usando un modelo de redes de restricciones temporales (*Temporal Constraint Networks TCN*) [110].

En las próximas secciones formalizaremos el modelo utilizado en el algoritmo HTNP2 que se basa en redes TCN.

3.2. - Modelo temporal del algoritmo HTNP2

La mayoría de los planificadores basados en estados que avanzan hacia adelante ya sean HTN o no, obtienen un plan como una secuencia totalmente ordenada de operadores primitivos:

$$(1)\quad \pi = \sigma_1, \sigma_2, \ldots, \sigma_n$$

La aplicación de cada acción va haciendo que el plan avance hacia adelante a partir del estado inicial.

$$(2)\quad \varepsilon_0 \circ \sigma_1 = \varepsilon_1, \varepsilon_1 \circ \sigma_2 = \varepsilon_2, \ldots, \varepsilon_{n-1} \circ \sigma_n = \varepsilon_n$$

Donde suponemos que el operador º es el operador que aplica los efectos de un operador primitivo y ε_0 y ε_n son el estado inicial y el final respectivamente. Este enfoque no es realista en la mayoría de los dominios de aplicación donde es común que las acciones se ejecuten en paralelo. Nos apoyaremos en el modelo temporal y en una estructura de vínculos causales para romper esta linealidad en el plan.

Como se ha indicado el modelo se apoya en el concepto de red de restricciones temporales. Podemos definir una red de restricciones temporales TCN [32] de la siguiente forma:

> **Definición 1:** *(Red de restricciones temporales):* Una red de restricciones temporales STN=(X, D, C) es un grafo dirigido cuyo conjunto de nodos X representan referencias temporales o *timepoints,* que toman valores en un rango concreto D, y el conjunto de los arcos C representan restricciones temporales entra las referencias temporales.

Usamos los *timepoints* para marcar los puntos de inicio y fin de los eventos de la base de datos temporal que tienen una duración en el tiempo. Los arcos entre los *timepoints* restringen por un lado la duración de los eventos y por otro lado indican la distancia en el tiempo de los distintos eventos. Los nodos o *timepoints* representan el instante de tiempo en que ocurre un evento. Al propagar las restricciones codificadas en los arcos generalmente quedan restringidos a un intervalo de valores válidos.

Lo primero que necesitamos definir en el modelo es cuales son los objetos que van a formar parte de la base de conocimiento temporal.

3.2.1. - *Modelado de tareas temporizadas*

Cada una de las tareas en HTNP2 representa un evento que tiene una duración determinada. Cada tarea tendrá asociados dos puntos temporales *(timepoints),* el *start* y el *end.*

Notaremos como start(σ_i) como el momento inicial en el que comienza una tarea y end(σ_i) como el momento en el que finaliza. $tp_{start}(\sigma_i)$ y $tp_{end}(\sigma_i)$ son los *timepoints* que marcan el inicio y el final de la tarea σ_i y t() es una función capaz de obtener el instante de tiempo en el que está previsto que ocurra un evento asociado a un *timepoint.*

De esta forma el tiempo en el que se tiene que ejecutar una acción *t*(σ_i) de la siguiente forma:

$$(3)\quad \text{start}(\sigma_i) = t(tp_{start}(\sigma_i))$$

Y la duración de una acción viene indicada por el arco que une esos dos *timepoints* y que notaremos como:

$$(4)\quad \text{duracion}(\sigma_i) = t(tp_{end}(\sigma_i)) - t(tp_{start}(\sigma_i))$$

Ilustración 24: Timepoints asociados a una tarea

En la ilustración 24 se muestran los *timepoints* asociados a una tarea o a un evento (*start y end*). Observar que la duración de una tarea, no es un valor único, sino que viene dada por un intervalo [d,D] o d ≤ duración ≤ D. Cuando la tarea tiene una duración exacta entonces se marcará con un intervalo de tipo [d,d] donde d es la duración de la tarea. De la misma forma el inicio y el fin de la tarea o evento viene dada por un intervalo en el que el valor inferior indica el momento más temprano en el que puede empezar el evento y el valor superior indica el momento mas tardío. El *schedule* del plan deberá asignar a cada *timepoint* del plan un valor concreto único de dichos intervalos. En general [d,D] marca la distancia entre dos *timepoints* cualquiera en la red.

En general el proceso general para el cálculo del tiempo asociado a las acciones del plan es el siguiente:

1 Definir los *timepoints* necesarios (nodos en la red).

2 Añadir restricciones al modelo (arcos entre los nodos).

3 Propagar las restricciones, o recalcular los intervalos de valores válidos asociados a los nodos, tras la adición de las nuevas restricciones.

4 Una vez que se ha terminado de añadir todos los puntos temporales y todas las restricciones y que la red temporal es consistente, es necesario realizar un proceso de *scheduling* para seleccionar un conjunto de valores concretos para los nodos.

En el modelo temporal se permite etiquetar los arcos y los nodos con intervalos donde el límite inferior está en 0 y el límite superior en ∞. Aunque se permite también como límite inferior el -∞ para indicar que no hay ninguna restricción. En la implementación del modelo sin embargo existen las limitaciones propias de la precisión de los valores numéricos que usemos. La implementación usa números enteros sin signo y ∞ toma el valor del entero máximo que podemos representar.

Se utilizan dos *timepoints* especiales en el modelo que marcan el inicio y el

fin del plan, que notaremos como $tp_{start}(plan)$ y $tp_{end}(plan)$. El *timepoint* que marca el inicio del plan viene etiquetado con el intervalo [0,0] y el que marca el fin va etiquetado con el intervalo [0,h]. Existe un arco que une ambos puntos que está marcado con el intervalo [0,h]. h denota el horizonte temporal máximo del plan, es decir el tiempo máximo durante el cual deben ocurrir todos los eventos. Por defecto se asume que h = ∞. Todo evento en el modelo temporal debe cumplir las siguientes restricciones:

$$(5)\ \forall_i\, t(tp_{start}(\sigma_i)) \geq t(tp_{start}(plan))$$

$$(6)\ \forall_i\, t(tp_{end}(\sigma_i)) \leq t(tp_{end}(plan))$$

Esta restricción se puede representar indicando que hay una restricción (arco) entre el comienzo del plan y el comienzo del evento, y otra restricción entre el fin del evento y el fin del plan (Ilustración 25).

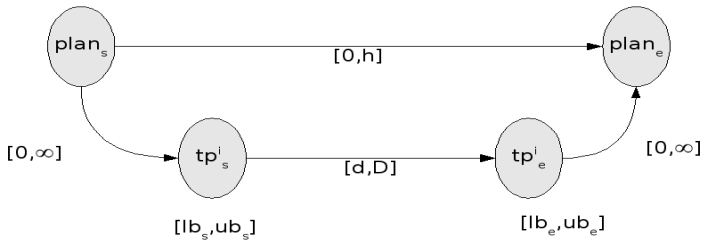


Ilustración 25: Un evento y sus restriccions con respecto al inicio y al fin del plan

El intervalo con el que va etiquetado un nodo marca el rango de valores temporales en los que puede ocurrir el inicio o el fin de un evento. Para calcular éste intervalo se toma como referencia el *timepoint* de inicio del plan y se utiliza un algoritmo de propagación de restricciones. Dicho algoritmo de propagación de restricciones utiliza la información de las restricciones marcadas por los arcos, para limitar el rango de valores que puede tomar un nodo.

El algoritmo de propagación de restricciones por tanto se encarga de garantizar la *consistencia* de la red de restricciones temporales. Definimos el concepto de consistencia como sigue:

> **Definición 2:** (*Consistencia temporal*): Una red de restricciones temporales conexa es temporalmente consistente si y solamente si existe una asignación de valores a los timepoints que es compatible con todas las restricciones de la red.

3.2.2. - *Modelado de literales*

Las tareas son representadas mediante eventos que ocurren durante un intervalo de tiempo, pero esas tareas producen unos efectos que cambian el estado en un intervalo determinado y sólo pueden ejecutarse cuando sus precondiciones son ciertas.

Los operadores primitivos pueden tener dos tipos de precondiciones en nuestro modelo:

- Precondiciones *at-start,* que deben ser válidas en el momento que se inicia la acción.

$$(7)\quad \forall \sigma_i \in \Pi, l_j^i \in \text{prec}_{\text{atstart}}(\sigma_i) \rightarrow \text{start}(\sigma_i) \geq \text{start}(l_j^i) \leq \text{end}(\sigma_i)$$

- Precondiciones *overall*, que deben ser ciertas durante toda la ejecución del operador primitivo.

$$(8)$$
$$\forall \sigma_i \in \Pi, l_j^i \in \text{prec}_{\text{overall}}(\sigma_i) \rightarrow \text{start}(\sigma_i) \geq \text{start}(l_j^i) \leq \text{end}(\sigma_i) \wedge \text{end}(l_j^i) \geq \text{end}(\sigma_i)$$

Se conoce como calcular el inicio y el fin de una tarea utilizando para ello los *timepoints* que tiene asociados dicha tarea, pero hasta ahora no se ha analizado como calcular ni el inicio ni el fin de un literal en nuestro modelo. Al igual que ocurre con las tareas se podrían asociar a los literales del estado dos *timepoints* que codificaran su comienzo y su finalización, esto nos conduciría hacia un modelo temporal muy general que nos daría gran expresividad, sin embargo dicho modelo temporal tiene una serie de inconvenientes. Primero, se complica el manejo del estado del planificador al poder no ser cierta la asunción de STRIPS [37]. Cuando un literal se niega en el efecto de una acción (se encuentra en la lista de supresión en el modelo de STRIPS) es quitado del estado actual del planificador. El estado actual del planificador únicamente mantiene los literales afirmados que son válidos en ese momento. Sin embargo se podrían producir inconsistencias temporales si el literal ha sido declarado válido durante un intervalo temporal cuyo máximo es mayor que el instante en el que el operador primitivo lo retira del estado. Segundo, introducir más puntos temporales en la red STN incrementa el tiempo necesario para que se realice la propagación de restricciones disminuyendo la eficiencia del planificador.

Se usará un modelo temporal equivalente en expresividad al anterior que sin embargo no necesita que todos los literales tengan asociados un par de *timepoints.*

La idea para modelar los literales se basa en que su inicio se puede

establecer tomando como referencia otros *timepoints*. Cualquier literal que se encuentre en el estado del planificador cumple la siguiente propiedad:

Como el planificador va hacia adelante:

$$(9)\ \ \pi = \sigma_1, \sigma_2, \ldots, \sigma_n$$

$$(10)\ \ \varepsilon_0 \circ \sigma_1 = \varepsilon_1, \varepsilon_1 \circ \sigma_2 = \varepsilon_2, \ldots, \varepsilon_{n-1} \circ \sigma_n = \varepsilon_n$$

Cualquier literal existente en un estado fue añadido por los efectos de un operador primitivo anterior o ya se encontraba en el estado inicial.

$$(11)\ \ \forall_{j=(1,\ldots,n)}\, l_k^i \in \varepsilon_i,\ j \le i,\ l_k^i \in \text{effects}(\sigma_j) \vee l_k^i \in \varepsilon_0$$

Si se añade a cualquier literal que se encuentre en el estado un valor que notaremos como Δt, que representa el tiempo en que cada acción tarda en conseguir sus efectos o la diferencia con el estado inicial en caso de que el literal pertenezca al estado inicial. Usando Δt es posible calcular el instante de tiempo en que comienza a ser válido dicho literal utilizando como referencias los *timepoints* asociados o bien al estado inicial o bien al operador primitivo que lo produjo.

El *timepoint* que utilizaremos como inicio para el estado inicial es el de inicio del plan o instante 0 o instante inicial $tp_{start}(plan)$. Así pues para cualquier literal que se defina en el estado inicial ε_0 podemos calcular su inicio de la siguiente forma:

$$(12)\ \ \forall_k\, l_k^0 \in \varepsilon_0,\ \text{start}(l_k^0) = t(tp_{start}(\text{plan})) + \Delta t_k$$

Con la restricción:

$$(13)\ \ \forall_k\, l_k^0 \in \varepsilon_0,\ t(tp_{start}(\text{plan})) + \Delta t_k \le t(tp_{end}(\text{plan}))$$

Generalmente Δt=0, es decir, los literales son ciertos desde el comienzo del plan, pero el modelo permite definir literales que se hacen ciertos tras una cantidad fija de tiempo tras el inicio del plan.

De manera similar los efectos de un operador se pueden conseguir en cualquier instante de tiempo tras el inicio de la acción y antes de su finalización:

$$(14)\ \ \forall \sigma_i \in \pi,\ l_k^i \in \text{effects}(\sigma_i),\ \text{start}(l_k^i) = t(tp_{start}(\sigma_i)) + \Delta t_k$$

Con la restricción:

$$(15)\ \ \forall \sigma_i \in \pi,\ l_k^i \in \text{effects}(\sigma_i),\ t(tp_{start}(\sigma_i)) + \Delta t_k \le t(tp_{end}(\sigma_i))$$

Este modelo permite una temporización más general que la de los efectos *at_start* y *at_end* de PDDL.

$$(16)\ \forall \sigma_i \in \pi, l_k^i \in effects_{atstart}(\sigma_i), start(l_k^i) = t(tp_{start}(\sigma_i)) + \Delta t_k, \Delta t_k = 0$$

$$(17)$$

$$\forall \sigma_i \in \pi, l_k^i \in effects_{atend}(\sigma_i), start(l_k^i) = t(tp_{start}(\sigma_i)) + \Delta t_k, \Delta t_k = duracion(\sigma_i)$$

Donde d es la duración de la tarea, aunque los efectos *at_end* también podrían ponerse como referencia al punto final de la tarea:

$$(18)\ \forall \sigma_i \in \pi, l_k^i \in effects_{atend}(\sigma_i), start(l_k^i) = t(tp_{end}(\sigma_i))$$

3.2.3. - *Concepto de timeline*

El modelo de temporización de los literales del estado presentado hasta ahora es capaz de representar el momento inicial en el que un literal comienza a ser válido, pero hasta ahora no se ha tenido en cuenta como modelar su instante final. Para facilitar el algoritmo de planificación y la representación del estado es interesante seguir manteniendo la asunción de STRIPS. Por lo tanto se asumirá que un literal es válido mientras que no haya un operador primitivo en el plan que lo niegue (Ilustración 26).

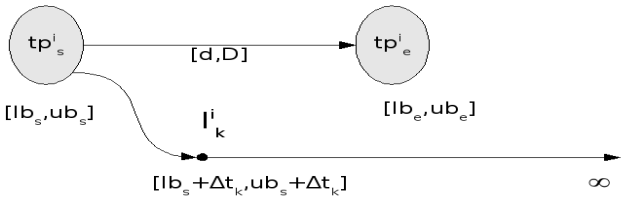


Ilustración 26: Periodo de validez de un literal conseguido por un operador o en el estado inicial

Esto no supone problema alguno ya que puesto que el planificador HTNP va hacia adelante y mantiene un orden total, cuando un operador primitivo retira un literal del estado, este ya no podrá ser utilizado en otro operador.

Sin embargo existen ciertos eventos que ocurren durante un determinado periodo de tiempo prefijado, independientes de los operadores primitivos, que también es necesario modelar. Es el caso de literales que modelan eventos exógenos no controlables.

> **Definición 3:** (*Evento exógeno*): Son eventos que ocurren durante un intervalo temporal que podemos predecir de forma exacta, pero que no podemos controlar ni alterar.

Definición 4: (*Timeline*): Definimos un timeline (línea de tiempo) como el conjunto de eventos exógenos que pueden influir sobre el plan resultante.

De esta forma el *timeline* es el siguiente conjunto de eventos que son válidos durante un intervalo de tiempo determinado:

$$\text{TIMELINE} = \langle([\text{start}(\text{event}_1), \text{end}(\text{event}_1)] \rightarrow \text{event}_1), \ldots, ([\text{start}(\text{event}_n), \text{end}(\text{event}_n)] \rightarrow \text{event}_n)\rangle \quad (19)$$

El modelo temporal que usa HTNP2 permite definir los eventos exógenos que forman parte del *timeline* mediante literales declarados en el estado inicial. Cada evento hace uso de un *timepoint* para marcar su inicio. Su finalización es calculada tomando como referencia el inicio y la duración del evento. Por tanto estos literales son ciertos en un intervalo fijo $[t_x,t_x+d]$, donde d es la duración. De esta forma podemos calcular el inicio y el fin de un evento de la siguiente forma:

$$(20)\quad \forall \text{event}_k \in \text{TIMELINE}, \text{start}(\text{event}_k) = t(tp_{start}(\text{event}_k))$$

$$(21)\quad \forall \text{event}_k \in \text{TIMELINE}, \text{end}(\text{event}_k) = t(tp_{start}(\text{event}_k)) + d$$

Donde el timepoint del inicio debe cumplir las propiedades:

$$(22)\quad \forall \text{event}_k \in \text{TIMELINE}, tp_{start}(\text{event}_k) - tp_{start}(\text{plan}) = t_x$$

$$(23)\quad \forall \text{event}_k \in \text{TIMELINE}, tp_{start}(\text{event}_k) \leq tp_{start}(\text{plan})$$

La Ilustración 27 muestra como se representaría uno de estos eventos en la red de restricciones temporales.

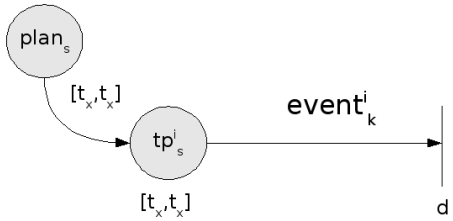


Ilustración 27: Nodos de la TCN asociados a un evento

Observar que esta representación permite un nivel de expresividad para los literales exógenos similar al del modelo general en donde todos los literales tienen asociados un par de *timepoints* con la ventaja de que la gestión del estado es más

sencilla y la red TCN más pequeña.

En la naturaleza son muy comunes los eventos cíclicos o periódicos. Estos eventos también suelen ser exógenos e incontrolables pero predecibles (por ejemplo el día y la noche). Se podrían representar estos eventos cíclicos mediante eventos del *timeline* como los estudiados hasta el momento, pero esto obligaría a introducir una gran cantidad de *timepoints* en la red TCN (en teoría infinitos) lo que complicaría la gestión del timeline.

Se propone añadir la generalización de que los eventos del *timeline* dispongan de un *periodo T* tras el cual vuelven a ser válidos. Los eventos que son ciertos en un único intervalo de tiempo como los estudiados hasta el momento tendrían $T=\infty$, lo que se interpreta como que solamente ocurren una vez. De esta forma se pueden calcular los distintos inicios y sus correspondientes finalizaciones de la siguiente forma:

(24)

$$\forall_{i=1,\dots,n} \forall\, event_k \in TIMELINE, T>0, start_i(event_k) = t(tp_{start}(event_k)) + i\cdot(T+d)$$

(25)

$$\forall_{i=1,\dots,n} \forall\, event_k \in TIMELINE, T>0, end_i(event_k) = t(tp_{start}(event_k)) + d + i\cdot(\Delta t + d)$$

Donde el *timepoint* del inicio tiene las mismas propiedades que en un evento no cíclico. La Ilustración 28 muestra de forma gráfica como se modelarían esta clase de eventos.

Dado un instante de tiempo t_i se puede conocer que eventos exógenos son válidos en ese instante intersectando con el *timeline,* de la siguiente forma:

(26) $$t_i, \forall_k\, event_k = true \text{ if } t_i \geq start(event_k) \wedge t_i \leq end(event_k)$$

Los literales del *timeline* unifican de la misma forma que los literales normales con las precondiciones de un operador primitivo y producen las mismas restricciones temporales (Ec. 7 y 8) , la diferencia básica se encuentra en la gestión que hace el planificador de los eventos cíclicos, para los cuales, si dado un intervalo del evento cíclico, al añadir las restricciones pertinentes al modelo temporal, se produce una inconsistencia, el planificador intentará de nuevo el anclaje con el intervalo siguiente, hasta que se consiga un anclaje válido o se alcance el horizonte superior de planificación[18].

18 El horizonte superior de planificación, es un valor temporal máximo hasta el cual se permite la generación de intervalos temporales, para evitar un backtracking infinito. Este valor se establece como un parámetro para el planificador (Anexo II).

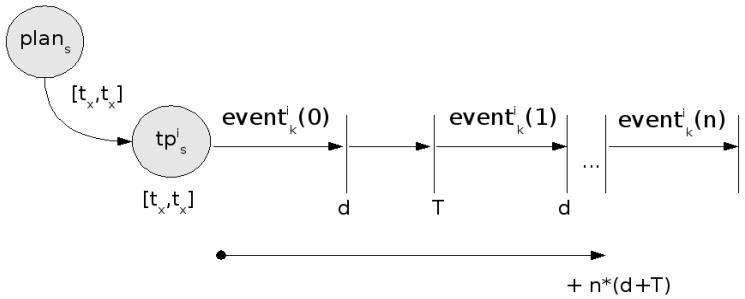


Ilustración 28: Representación de un evento cíclico

3.3. - Rompiendo la linealidad del plan

El modelo temporal presentado es suficiente para dar soporte temporal a un planificador basado en estados hacia adelante, sin embargo se quiere aprovechar la riqueza que da esta estructura temporal para poder romper la linealidad del plan y que sea por lo tanto posible obtener planes con ramas de acciones que se puedan ejecutar en paralelo.

Los planificadores basados en estados que avanzan hacia adelante, obtienen los planes como una secuencia de acciones con un orden total. De esta forma se pueden asociar un instante de tiempo a cada estado, de una forma sencilla:

$$(27)\quad \pi = \sigma_1, \sigma_2, \ldots, \sigma_n$$

$$(28)\quad \varepsilon_0 \circ \sigma_1 = \varepsilon_1, \varepsilon_1 \circ \sigma_2 = \varepsilon_2, \ldots, \varepsilon_{n-1} \circ \sigma_n = \varepsilon_n$$

$$(29)\quad t_{makespan}(\varepsilon_i) = t_{makespan}(\varepsilon_{i-1}) + duracion(\sigma_{i-1})$$

$$(30)\quad t_{makespan}(\varepsilon_0) = 0$$

Así como encontrar un *schedule* o asignación de tiempo a acciones también de una forma relativamente simple.

$$(31)\quad \begin{aligned} start(\sigma_0) &= t_{makespan}(\varepsilon_0) = 0 \\ start(\sigma_i), i>0 &= \sum_{0<j\le i-1} duracion(\sigma_j) \\ end(\sigma_i) &= start(\sigma_i) + duracion(\sigma_i) \end{aligned}$$

El orden total además garantiza que las precondiciones de un operador σ_i si son ciertas en el estado ε_{i-1} nunca van a ser violadas ya que no puede haber otro

operador primitivo que se interponga y las haga falsas.

El modelo basado en TCN planteado sin embargo permite un orden parcial PO (*Partial Order*) ya que es perfectamente válido que dos operadores primitivos se solapen en el tiempo, siempre y cuando las restricciones impuestas en la red temporal lo permitan. Para evitar la violación de precondiciones se pueden añadir restricciones a la red TCN que fuercen un orden total o bien introducir estas restricciones únicamente cuando sean necesarias para lo cual el modelo se apoya en una estructura de vínculos causales λ.

Definiremos el plan parcial que obtiene HTNP2 como una tupla POHTN = <π,θ,λ,Π> donde:

- π: Es el conjunto de operadores primitivos del árbol de expansión HTN.
- θ: Es el orden entre los operadores primitivos.
- Π: Es el árbol de expansión HTN resultante al obtener un plan. Dicha expansión influye en el orden en el que se pueden ejecutar los operadores.
- λ: Es la estructura de vínculos causales que relacionan los efectos de un operador con las precondiciones de otro $\sigma_i \rightarrow^{l_k} \sigma_j$. Dicha estructura también impone restricciones de ordenación.

3.3.1. - *Restricciones temporales causales*

HTNP2 mantiene una estructura de vínculos causales λ entre los operadores primitivos del plan en descomposición. Cada uno de los vínculos causales añadirá una restricción a la STN de forma que se garantiza una correcta ordenación causal entre los operadores primitivos, de forma que ningún operador σ_i se pueda ejecutar antes de que se consigan los literales l_k^i de los cuales depende. De esta forma se evitan las posibles violaciones de precondiciones producidas por el solapamiento de eventos en la STN. Destacar que a pesar de que se mantenga un orden parcial temporal en la STN entre los operadores, el funcionamiento del planificador sigue siendo el mismo, hacia adelante y añadiendo operadores uno a uno y en orden. Esto garantiza que los literales que retira (niega) un operador del estado, no son usados por un operador que se añade al plan posteriormente.

Formalizando esta idea, cualquier literal del que dependa un operador primitivo, se hizo cierto en un estado anterior:

$$(32)\quad \pi = \sigma_1, \sigma_2, \ldots, \sigma_n$$

$$(33)\quad \varepsilon_0 \circ \sigma_1 = \varepsilon_1, \varepsilon_1 \circ \sigma_2 = \varepsilon_2, \ldots, \varepsilon_{n-1} \circ \sigma_n = \varepsilon_n$$

$$(34)\quad \forall_{i=(1,..,n)} \sigma_i \in \pi, \forall\, l_k^i \in \left[\text{prec}_{\text{atstart}}(\sigma_i) \cup \text{prec}_{\text{overall}}(\sigma_i) \right], l_k^i \in \varepsilon_j, j < i$$

El inicio de cualquier acción siempre debe ser posterior al momento en el cual se consigue cualquiera de los literales de los que depende.

$$(35)\quad \forall_{i=(1,..,n)} \sigma_i \in \pi, \forall\, l_k \in \left[\text{prec}_{\text{atstart}}(\sigma_i) \cup \text{prec}_{\text{overall}}(\sigma_i) \right], \text{start}(\sigma_i) \geq \text{start}(l_k^i)$$

En cualquier caso ya sea un literal perteneciente al estado inicial, al *timeline,* o un literal conseguido como efecto de otro operador anterior, se conoce como calcular su momento de inicio (Ec: 12,14,20,24) y también de finalización en el caso de los literales del *timeline* (Ec: 21,25). Lo único que queda por hacer es garantizar la propiedad (35) añadiendo las restricciones temporales convenientes a la STN. También tenemos que garantizar el anclaje correcto con los eventos del *timeline*:

$$(36)\quad \begin{array}{c} \forall\, \sigma_i \in \pi, \text{event}_k \in \text{prec}_{\text{atstart}}(\sigma_i), \text{event}_k \in \text{TIMELINE}, \\ \exists_{j=0,\ldots,n} \text{start}(\sigma_i) \geq \text{start}_j(\text{event}_k) \end{array}$$

$$(37)\quad \begin{array}{c} \forall\, \sigma_i \in \pi, \text{event}_k \in \text{prec}_{\text{overall}}(\sigma_i), \text{event}_k \in \text{TIMELINE}, \\ \exists_{j=0,\ldots,n} \text{start}(\sigma_i) \geq \text{start}_j(\text{event}_k) \wedge \text{end}(\sigma_i) \leq \text{end}_j(\text{event}_k) \end{array}$$

El cumplimiento de estas restricciones obligan al planificador en el caso de los eventos cíclicos (j>0) a buscar un intervalo de tiempo donde el anclaje del operador no viole las restricciones sobre el inicio y el fin del operador.

De forma gráfica podemos apreciar como se introducen estas restricciones

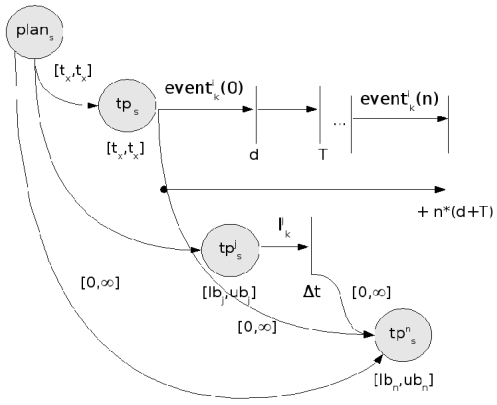


Ilustración 29: Restricciones impuestas por la dependencia de literales

en la red STN en la Ilustración 29. El *timepoint* n referencia el inicio de un operador,

que por un lado depende de un literal cíclico del *timeline,* que usa el *timepoint* i para marcar su inicio y por otro lado de un literal proporcionado por otro operador primitivo cuyo inicio viene dado por el *timepoint* j.

Los vínculos causales nos permiten por tanto restringir el instante de tiempo en el cual se tiene que ejecutar un operador, de forma que todos los literales de los que este dependa estén disponibles en ese momento.

Basándonos en las restricciones temporales impuestas sobre el inicio de un operador primitivo σ_i podemos definir el *timestamp* causal del inicio del operador *tsc(start(σ_i))* como el instante de tiempo válido a partir del cual se puede ejecutar dicho operador sin dañar sus dependencias causales:

$$(38)\quad \mathrm{tsc}(\mathrm{start}(\sigma_i)) = \mathrm{Max}(\mathrm{start}(l_k^i)), l_k^i \in \mathrm{prec}_{\mathrm{atstart}}(\sigma_i) \vee l_k^i \in \mathrm{prec}_{\mathrm{overall}}(\sigma_i)$$

El *timestamp* del final del operador se puede calcular fácilmente a partir de su *timestamp* inicial de la siguiente forma:

$$(39)\quad \mathrm{tsc}(\mathrm{end}(\sigma_i)) = \mathrm{tsc}(\mathrm{start}(\sigma_i)) + \mathrm{duracion}(\sigma_i)$$

Observar que de esta forma el cálculo del inicio del operador es distinto del utilizado con el cálculo del *makespan* utilizado por otros planificadores hacia adelante que manejan el tiempo como por ejemplo SHOP2 [28] en el que el manejo del tiempo se hace mediante la codificación directa en el dominio. El cálculo no depende tanto del orden de inserción en el plan sino que depende más de la estructura causal del plan. De esta forma el manejo de tiempo es mucho más preciso y flexible que con el cálculo de *makespan.* De hecho se puede demostrar que:

$$(40)\quad \mathrm{tsc}(\mathrm{start}(\sigma_i)) \leq \mathrm{t}_{\mathrm{makespan}}(\mathrm{start}(\sigma_i))$$

Demostración: Supongamos que queremos calcular el momento en el que comienza la última acción en el plan, entonces si (40) no es cierto:

$$(41)\quad \mathrm{tsc}(\mathrm{start}(\sigma_n)) > \mathrm{t}_{\mathrm{makespan}}(\mathrm{start}(\sigma_n))$$

El cálculo del *makespan* depende del orden en el cual los operadores primitivos son insertados en el plan:

$$(42)\quad \mathrm{t}_{\mathrm{makespan}}(\mathrm{start}(\sigma_i)) = \mathrm{t}_{\mathrm{makespan}}(\mathrm{start}(\sigma_{i-1})) + \mathrm{duracion}(\sigma_{i-1}), i > 2$$

$$(43)\quad \mathrm{t}_{\mathrm{makespan}}(\mathrm{start}(\sigma_1)) = 0$$

De esta forma igualmente la duración del plan es siempre la suma de las duraciones de todos los operadores que lo componen (estamos ignorando el uso de eventos exógenos para simplificar la demostración que además generalmente no son soportados por estos planificadores). De esto se deduce que:

$$(44)\quad \mathrm{tsc}(\mathrm{start}(\sigma_i)) > \sum_{i=0,\ldots,n} \mathrm{duration}(\sigma_i)$$

Definamos una cadena causal como una lista ordenada de operadores primitivos en la que el operador que ocupa la posición i en la lista tiene al menos un vínculo causal con el operador que ocupa la posición (i-1). La duración de la cadena causal es la suma de todas las duraciones de los operadores al igual que ocurre con el *makespan.*

Si se rompe este encadenamiento causal, por que por ejemplo el operador n, no tenga un vínculo con el (n-1), cosa que es frecuente en los planes, entonces el plan puede tener una duración de *d* unidades temporales menos que el calculado por el makespan, siendo *d* la duración de la tarea (n-1), ya que n y (n -1) se podrían ejecutar en paralelo, de donde se deduciría para este caso:

$$(45)\quad (\sum_{i=0,\ldots,n} \mathrm{duration}(\sigma_i)) - \mathrm{duration}(n) > \sum_{i=0,\ldots,n} \mathrm{duration}(\sigma_i)$$

Lo cual es falso y por contraposición al absurdo demuestra (40)

En el caso del *timestamp* por tanto el orden de ejecución de los operadores primitivos no viene determinado de forma estricta por el orden en el que son planificados a pesar de que HTNP2 construye el plan siguiendo un orden total.

3.3.2. - *Temporización del estado múltiple*

En el modelo temporal seguido por HTNP2 el estado que mantiene el planificador de un paso de planificación al siguiente no tiene una única marca de tiempo, debido por un lado a que los literales almacenados en el estado pueden tener momentos de consecución distintos y también a los literales del TIMELINE.

De esta forma se puede decir que el estado que mantiene el planificador en un paso del algoritmo representa múltiples estados en el tiempo de forma simultánea. De esta forma no tenemos un estado único ni una marca de tiempo exacta para ese estado. Lo más que podemos calcular es la ventana de tiempo durante la cual están vigentes una serie de literales.

Sin embargo en HTNP2 no es necesario disponer de una marca de tiempo única para el estado, esto es únicamente necesario para el cálculo del *makespan*. Se puede seguir haciendo inferencia sobre el momento en el que va a ocurrir un evento en concreto, como hemos visto, teniendo en cuenta las restricciones temporales y realizando inferencia sobre la red STN.

El problema de realizar inferencias sobre la red STN, cuando entre cada dos timepoints, únicamente hay un arco que los une (no hay disyunciones en el

modelo), es una instancia de un problema temporal simple STP (*Simple Temporal Problem*) [110] que es un caso especial de los problemas GTP (*General Temporal Problem*).

3.4. - Herencia de restricciones a través de la estructura jerárquica

El modelo presentado hasta el momento no hace uso de las características jerárquicas del planificador, de hecho el modelo podría utilizarse con un planificador no jerárquico basado en estados y hacia adelante.

En esta sección se tienen en cuenta las restricciones explícitas cualitativas y cuantitativas impuestas por el escritor de dominios sobre las redes de tareas, que añaden otro tipo de restricciones distintas de las que tienen carácter causal.

Para mantener la generalidad del modelo se dará a las tareas abstractas un tratamiento equivalente al de los operadores primitivos. Cada tarea abstracta tendrá dos *timepoints* asociados uno al inicio *start*(τ_i) y otro al fin *end*(τ_i). Podemos utilizar estos *timepoints* para, al igual que con los operadores primitivos, definir restricciones temporales sobre las tareas abstractas.

Aparecen ahora las relaciones de herencia entre una una tarea abstracta τ_i y las tareas por las cuales es sustituida, y que pertenecen a una red de tareas Γ_j, tras la aplicación de un método. Lo natural y lo que el escritor de dominios espera es que todas las tareas pertenecientes a esta red de tareas hereden las restricciones impuestas sobre los *timepoints* de la tarea padre. Por eso definimos las siguientes restricciones entre una tarea padre y sus tareas hijas:

$$(46)\quad \forall \rho_k \in (\Gamma_j \in \tau_i)\, \text{start}(\rho_k) \geq \text{start}(\tau_i) \wedge \text{end}(\rho_k) \leq \text{end}(\tau_i)$$

Estas restricciones son las que se muestran con línea punteada en la Ilustración 30. Cada tarea tiene asociados dos *timepoints* que representan su inicio y su finalización. La tarea abstracta τ_0 se descompone en la red de tareas (τ_1 [τ_2 σ_1] σ_2). Todas las tareas de la red heredan las restricciones de la tarea padre, además deben definir restricciones para garantizar el orden entre las mismas (flechas continuas).

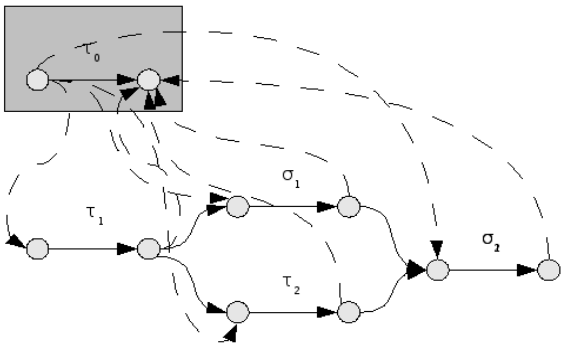


Ilustración 30: Herencia de restricciones temporales

El conjunto θ_j mantiene las relaciones de orden entre las tareas de Γ_j, en el ejemplo de la red de tareas de la Ilustración 30 el conjunto tiene las siguientes relaciones de orden: θ_j= {before(τ_1,σ_1), before(τ_1,τ_2), before(τ_2,σ_2),before(σ_1,σ_2)}. De forma general, podemos definir que para mantener el orden entre las tareas dentro de la red de tareas es necesario garantizar las siguientes restricciones:

$$(47) \quad \forall i,j,\text{before}(\rho_i,\rho_j)\in\theta\rightarrow \text{end}(\rho_i)\leq\text{start}(\rho_j)$$
$$\forall i,j,\text{before}(\rho_i,\rho_j)\in\theta\rightarrow \text{end}(\rho_i)-\text{start}(\rho_j)\in[0,\infty)$$

Obsérvese que las tareas independientes o paralelas no añaden restricciones para mantener ningún tipo de orden.

Realmente una vez que se tienen las restricciones de orden no es necesario añadir todas las restricciones de herencia. En el ejemplo de la Ilustración 30 basta con añadir dos restricciones (start(τ_1) >= start(τ_o), end(σ_2) <= end(τ_o)), para conseguir el mismo efecto con un número menor de restricciones. Si suponemos que *first* y *last* son los operadores que calculan el conjunto de tareas de Γ_j que cumplen las propiedades:

$$(48) \quad \text{first}(\Gamma_j)=\{\rho_k,...,\rho_n\}\neg\exists i,\text{before}(\rho_i,\rho_k)$$

$$(49) \quad \text{last}(\Gamma_j)=\{\rho_k,...,\rho_n\},\neg\exists i,\text{before}(\rho_k,\rho_i)$$

Es decir, son los operadores que calculan que tareas de una red de tareas no tienen una dependencia de orden con otra tarea anterior y las tareas con las cuales ninguna otra tiene una dependencia de orden, podemos reducir el número de restricciones de herencia impuestas para una tarea abstracta τ_i usando las siguientes restricciones en lugar de las definidas en (46):

$$(50)\quad \forall \rho_k \in \text{first}(\Gamma_j)\ \text{start}(\rho_k) \geq \text{start}(\tau_i)$$

$$(51)\quad \forall \rho_k \in \text{last}(\Gamma_j)\ \text{end}(\rho_k) \leq \text{end}(\tau_i)$$

En la Ilustración 31 se muestra como queda la red TCN mostrada en la Ilustración 30 con el número de restricciones de herencia reducidas.

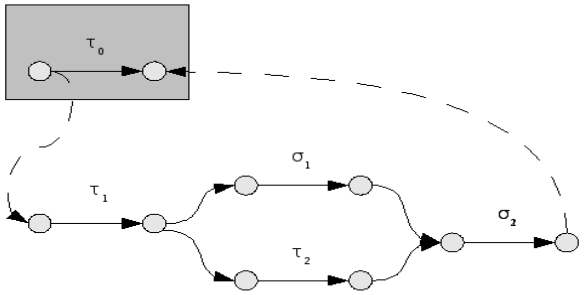


Ilustración 31: Herencia de restricciones temporales

El *start* y el *end* de una tarea compuesta o un operador primitivo también se pueden utilizar para definir restricciones temporales para el comienzo, finalización o duración de una tarea. Analicemos las distintas posibilidades:

- Restricciones sobre el inicio, que pueden definir de forma exacta el inicio de la tarea o bien acotarla:
  - Comienzo a una hora exacta t_x:

$$(52)\quad \begin{array}{c} \text{start}(\rho_i) = t_x \\ tp_{start}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, t_x] \end{array}$$

  - Comienzo en un intervalo de tiempo $[t_x,t_y]$:

$$(53)\quad \begin{array}{c} \text{start}(\rho_i) \geq t_x \wedge \text{start}(\rho_i) \leq t_y \\ tp_{start}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, t_y] \end{array}$$

  - Comienzo máximo, la tarea debe comenzar antes del instante de tiempo t_x:

$$(54)\quad \begin{array}{c} \text{start}(\rho_i) \leq t_x \\ tp_{start}(\rho_i) - tp_{start}(\text{plan}) \in [0, t_x] \end{array}$$

  - Comienzo mínimo, la tarea debe comenzar como mínimo

después del instante t_x:

$$(55)\quad \begin{matrix} \text{start}(\rho_i) \geq t_x \\ tp_{start}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, \infty) \end{matrix}$$

- Comienzo desconocido, el comienzo de la tarea no está delimitado:

$$(56)\quad \begin{matrix} \text{start}(\rho_i) = ¿? \\ tp_{start}(\rho_i) - tp_{start}(\text{plan}) \in [0, \infty) \end{matrix}$$

- Se pueden definir un conjunto de restricciones simétricas sobre la finalización de una tarea:
  - Finalización a una hora exacta t_x:

$$(57)\quad \begin{matrix} \text{end}(\rho_i) = t_x \\ tp_{end}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, t_x] \end{matrix}$$

  - Finalización en un intervalo de tiempo $[t_x,t_y]$:

$$(58)\quad \begin{matrix} \text{end}(\rho_i) \geq t_x \wedge \text{end}(\rho_i) \leq t_y \\ tp_{end}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, t_y] \end{matrix}$$

  - Finalización máxima, la tarea debe terminar antes del instante de tiempo t_x:

$$(59)\quad \begin{matrix} \text{end}(\rho_i) \leq t_x \\ tp_{end}(\rho_i) - tp_{start}(\text{plan}) \in [0, t_x] \end{matrix}$$

  - Finalización mínima, la tarea debe terminar como mínimo después del instante t_x:

$$(60)\quad \begin{matrix} \text{end}(\rho_i) \geq t_x \\ tp_{end}(\rho_i) - tp_{start}(\text{plan}) \in [t_x, \infty) \end{matrix}$$

  - Finalización indeterminada, el fin de la tarea no está delimitado:

$$(61)\quad \begin{matrix} \text{end}(\rho_i) = ¿? \\ tp_{end}(\rho_i) - tp_{start}(\text{plan}) \in [0, \infty) \end{matrix}$$

- También se pueden definir restricciones sobre la duración de una tarea. Estas restricciones afectan al *timepoint* del *start* o del *end* o a ambos de forma simultánea.
  - Duración de la tarea fija, t_x:

$$(62)\quad \begin{array}{c} \mathrm{end}(\rho_i)-\mathrm{start}(\rho_i)=t_x \\ tp_{end}(\rho_i)-tp_{start}(\rho_i)\in[t_x,t_x] \end{array}$$

- Duración de la tarea indeterminada:

$$(63)\quad \begin{array}{c} \mathrm{end}(\rho_i)-\mathrm{start}(\rho_i)=¿? \\ tp_{end}(\rho_i)-tp_{start}(\rho_i)\in[0,\infty) \end{array}$$

3.5. - Deslinearización del plan

En un plan totalmente ordenado no existe el concepto de amenaza a la estructura causal del plan. Al deslinearizar el plan, se rompen muchas relaciones de orden previamente existentes, que se sobreentendían de forma implícita, de forma que será necesario comprobar, y proteger en su caso, aquellos literales cuya satisfacción pudiese verse amenazada.

- Por un lado están los literales que aparecen en las precondiciones de una acción y que fueron conseguidos por los efectos de alguna acción previa. Estos literales están protegidos por los vínculos causales que se crean durante la planificación y que añaden restricciones de orden sobre los operadores en el modelo temporal.

  - Los literales de una acción que fueron satisfechos por axiomas generan también vínculos causales del operador con dichos literales, ya que un axioma de PDDL puede verse como una forma de englobar o abstraer una condición bajo el nombre de un literal nuevo, condición que es dependiente del estado actual.

  - Se considera que las precondiciones de los métodos no deben generar vínculos causales pues no restringen la estructural causal del plan, tan solo deben servir para guiar el proceso de descomposición. Se considera por ejemplo un fallo en la escritura del dominio suponer en los operadores primitivos condiciones que hayan sido satisfechas en tareas abstractas. Eliminar esta suposición no es un problema grave, el modelo está preparado para ello (es un flag del algoritmo HTNP2) pero origina una mayor carga computacional tanto sobre la estructura que mantiene los vínculos causales como sobre la red temporal.

  - Las precondiciones y efectos, comprobadas o generadas en una acción *inline* no generan vínculos causales, ni por lo tanto restricciones temporales. Como se comentó las tareas *inline* están diseñadas para introducir información de control en el estado del

planificador, por lo tanto utilizable únicamente en las precondiciones de los métodos de una tarea abstracta, que como ya se ha dicho, tampoco generan vínculos causales.

- Los literales del *timeline* que representan eventos exógenos no pueden ser negados (eliminados) por los efectos de un operador primitivo. Esto tiene sentido puesto que estos literales representan eventos incontrolables, por lo tanto no gestionables por los operadores primitivos. De esta forma los vínculos causales contra este tipo de literales, siempre se realizan con el *timeline.* Puesto que ninguna acción puede eliminar estos literales, su satisfacción está garantizada por las restricciones temporales que se introducen entre las acciones y el *timeline.*

- Por otro lado están los literales tipo *fluent* en los que se basa un operador primitivo. Estas precondiciones específicas pueden ser utilizadas para representar restricciones sobre recursos, entre otras cosas. Sin el uso de vínculos causales la deslinearización de un plan totalmente ordenado podría provocar una reordenación de las acciones de forma que se produjese una violación de recursos que, en el plan totalmente ordenado no se produciría. Protegeremos por tanto los literales *fluent* con vínculos causales de forma que entre dos acciones consecutivas que hacen uso del mismo literal, se establece un vínculo causal.

  - Un caso especial de funciones en HTNP2 es cuando utilizamos scripts en un lenguaje de programación genérico, por ejemplo Python, para hacer determinados cálculos. Este tipo de fluents sólo se pueden utilizar en las precondiciones de un operador, por lo tanto no pueden ser cambiadas por un operador y no producen vínculos causales.

- A veces es necesario asegurarse de la negación de un literal. Supongamos que un operador tiene en sus precondiciones la negación de un literal. Para garantizar que el literal se encuentre efectivamente negado cuando comience la acción es necesario insertar un vínculo causal entre el operador que negó dicho literal y el operador que lo requiere en sus precondiciones.

Teniendo en cuenta estas consideraciones no es necesario "forzar" el orden total en el plan, ya que el planificador añadirá restricciones temporales de orden entre dos operadores que tienen un vínculo causal entre sí, garantizando de esta forma que el plan parcial obtenido es totalmente consistente.

3.6. - Expresividad de HTN-PDDL para el manejo del tiempo

Hasta ahora se ha presentado el modelo temporal subyacente basado en redes temporales STN, pero es necesario dotar al escritor de dominios de las herramientas expresivas necesarias para poder hacer uso de una forma natural de estas capacidades temporales.

HTN-PDDL, el lenguaje utilizado por el planificador HTNP2, permite la escritura de dominios temporales. En esta sección estudiaremos los mecanismos de los que dispone el lenguaje para hacerlo.

Como se ha explicado HTN-PDDL es una extensión para dominios jerárquicos HTN del lenguaje PDDL 2.1 [81]. PDDL 2.1 incluye capacidades temporales mediante el uso de literales temporizados y acciones durativas. Estas construcciones son compatibles con el lenguaje HTN-PDDL de forma que las acciones durativas, por ejemplo, de un dominio PDDL 2.1 nivel 3 pueden ser utilizadas de forma directa en HTN-PDDL, pero no a la inversa, ya que HTN-PDDL incluye algunas expresiones temporales no incluidas en PDDL. Es decir, las capacidades temporales de PDDL 2.1 nivel 3 son un subconjunto de las soportadas por HTN-PDDL.

Ejemplo:

```
;;Acción durativa de HTN-PDDL válida también en PDDL
(:durative-action drive-to
 :parameters(?destination)
 :duration (= ?duration
              (/ (distance ?current ?destination) (average-speed my-car)))
 :condition(and
              (current-position ?current)
              (available my-car))
 :effect(and
              (current-position ?destination)
              (not (current-position ?current))))
```

3.6.1. - *Orden cualitativo entre las tareas*

El orden temporal entre las tareas viene dado por el orden relativo entre las tareas de una red de tareas, que llamaremos orden cualitativo, y las necesidades de realizar una tarea en un determinado rango de tiempo que llamaremos restricciones de orden cuantitativas.

Recordemos que las redes de tareas en HTNP al igual que en HTNP2 son de tres tipos:

- **Secuenciales:** Representadas entre paréntesis, por ejemplo (A B). Como se ha expuesto, cuando el planificador encuentra una red de este tipo, automáticamente introduce una restricción temporal entre las tareas involucradas, de forma que en el ejemplo dado el comienzo de B no podría producirse antes de la finalización de A.

- **Paralelas o independientes:** Representadas entre corchetes, por ejemplo [A B]. Cuando el planificador encuentra una red de tareas de este tipo no introduce ninguna restricción de orden temporal entre las tareas, de esta forma en nuestro ejemplo se puede ejecutar primero A, primero B o ambas de forma simultánea. En esta característica estriba la capacidad del algoritmo HTNP2 para encontrar planes parcialmente ordenados.

- **Permutables:** Representadas entre ángulos, por ejemplo <A B>. En este tipo de redes de tareas se supone que existe algún tipo de dependencia entre las tareas involucradas, lo que pasa es que el diseñador de dominio la desconoce y quiere que sea el algoritmo de búsqueda empleado en el planificador el que la descubra. El planificador probará por lo tanto todas las posibles permutaciones de las tareas de la red, introduciendo las restricciones de ordenación temporal necesarias entre ellas.

Obviamente las estructuras básicas para construir redes de tareas se pueden combinar para construir redes de tareas más complejas, la Ilustración 32 muestra la obtención de un plan parcialmente ordenado a partir de la red de tareas (T1 [T2 (T3 T4)] T5).

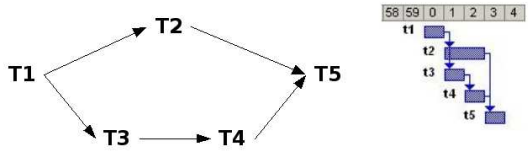


Ilustración 32: Ejemplo de parcialización de la red de tareas (T1 [T2 (T3 T4)] T5)

Destacar que el planificador HTNP2, y a pesar de que las redes se hayan declarado como paralelas, puede decidir una ordenación determinada entre las tareas de las redes a fin de mantener la consistencia de la estructura causal.

3.6.2. - *Literales temporizados*

Como se ha explicado los literales en el estado están temporizados, de una forma similar a los *timed initial literals* de PDDL 2.2 [128]. La temporización se realiza de forma explícita en la declaración del problema en HTN-PDDL. En general HTNP2 supone que todos los literales están temporizados. Si en la declaración del problema un literal no está temporizado entonces se supone que es válido desde el instante 0 (instante inicial) y hasta que un operador primitivo lo niegue.

Podemos declarar tres tipos de literales distintos:

- Literales que se hacen válidos en un instante de tiempo determinado hasta que el efecto de un operador primitivo lo niegue. Se declaran utilizando la cláusula de PDDL *at:* (at *instante_temporal* (*literal_temporizado*)).

- Literales que son ciertos durante un intervalo de tiempo determinado. Se declaran utilizando la cláusula de HTN-PDDL *between:* (between *instante_temporal_1* and *instante_temporal_2* (*literal_temporizado*)).

- Literales que son ciertos de forma periódica. Se declaran utilizando la cláusula de HTN-PDDL *between:* (between *instante_temporal_1* and *instante_temporal_2* and every *unidades_de_tiempo* (*literal_temporizado*)).

Veamos ejemplos concretos de cada uno de ellos:

```
Ejemplo:
;;Declaración de distintos tipos de literales temporizados
(:init
  ;; literales que son válidos desde el instante 0
  (= (precio linea_bus11 110))
  (en alhambra_palace alhambra)
  ;; literales ciertos en un instante de tiempo determinado
  ;; literal válido 120 unidades de tiempo después del inicio
  (at 600 (disponible coche_alquiler))
  ;; literal disponible en un instante temporal fijo
  (at “12:00:00 12/05/2007” (disponible habitacion_101 alhambra_palace))
  ;; literal válido solamente en un intervalo de tiempo
  (between 600 and 120 (fiesta fiesta_cerveza alhambra_palace))
  ;; literal periodico
  (between “08:30:00 12/05/2007” and “18:00:00 12/05/2007” and every 870
(abierto alhambra))
...
)
```

Hay que hacer una serie de consideraciones sobre el ejemplo mostrado. Primero se observa como en HTN-PDDL es posible mezclar referencias a un instante temporal relativas al instante de tiempo con referencias absolutas

expresadas como cadenas fecha/hora. De esta forma se facilita la escritura de dominios temporales y la legibilidad de los mismos, pero también nos obligan a introducir una serie de parámetros en el planificador para que este sea capaz de interpretarlos. Estos parámetros son introducidos en la sección de personalización (*customization*) del fichero del problema y marcan la unidad temporal que usaremos (en los literales del ejemplo minutos), el formato de fecha/hora, y el instante temporal que se toma como inicio.

```
Ejemplo:
;; Ejemplo de sección de personalización, para los literales mostrados en
;; el ejemplo anterior
(:customization
   (= :time-format "%H:%M:%S %d/%m/%Y")
   (= :time-horizon-relative 1000)
   (= :time-start " 00:00:00 12/05/2007")
   (= :time-unit :minutes)
)
```

Observar que se define un horizonte temporal relativo al inicio de 1000 minutos. Este horizonte temporal sirve para limitar el backtracking que un operador primitivo puede realizar al unificar contra un literal periódico, que de otra forma sería infinito. Por ejemplo en el literal periódico expuesto en el ejemplo que modela el horario de visita a la Alhambra, que se repite día tras día, sin ese límite un operador primitivo que hiciera uso de ese literal en una de sus precondiciones caería en un backtracking infinito (Ilustración 33).

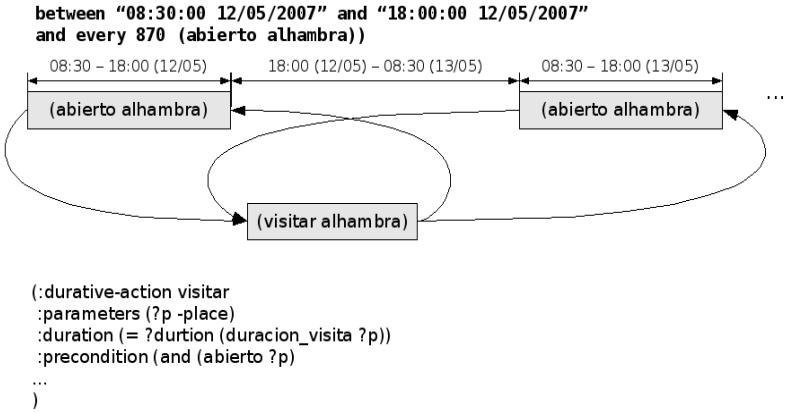


Ilustración 33: La tarea (visitar alhambra) puede ser satisfecha en múltiples instantes de tiempo en el timeline

Los literales temporizados en el estado “empujan” a los operadores primitivos hacia el futuro para garantizar que en el momento en que estos se

ejecuten todos los literales son válidos (Ec: 38).

3.6.3. - *Orden cuantitativo entre las tareas*

Se ha analizado como las redes de tareas y los literales contenidos en el estado fuerzan de forma implícita a una tarea o a un operador a ejecutarse en un instante determinado. A veces hay restricciones más complejas que las causales o las que se puedan definir mediante el orden de descomposición que requieren de un tratamiento mas específico en el dominio. Se estudiarán a continuación las herramientas expresivas con las que cuenta HTN-PDDL para que el escritor de dominios pueda definir de forma explícita cuando quiere que dichas tareas se realicen.

HTN-PDDL permite el uso de las variables *?start, ?end y ?dur* para la definición de restricciones temporales en una red de tareas. La variable *?start* hace referencia al *timepoint* que marca el inicio de una acción, mientras que la variable *?end* marca su final. ?dur se usa para definir restricciones sobre la duración de la tarea. Se pueden utilizar los operadores >=, <=, =, <, y > junto con estas variables, y una expresión numérica para definir las restricciones.

Las restricciones tienen la forma:

```
;; una restricción
((restricción (tarea))
;; varias restricciones
((and restricción_1 restricción_2 ... restricción_n (tarea))
```

Donde una restricción es una expresión con la estructura:

```
(operador variable_temporal expresión_numérica)
```

De forma que, por ejemplo, para indicar que una tarea T debe comenzar al menos 5 unidades temporales después del punto de inicio se escribiría la restricción (>= ?start 5 (T)).

Se usará el siguiente ejemplo simplificado para analizar de forma un poco más detallada estos conceptos:

```
Ejemplo:
;; Tres tareas abstractas con relaciones de herencia
;; La tarea A (MA) no impone más restricciones que las de ordenación
(:task A
 :parameters ()
 (:method MA
  :precondition ()
  :tasks ((A1)(A2))
```

```
))

;; La tarea B (MB1) pone restricciones en el inicio y el fin de la
;; subtarea A2
(:task B
 :parameters ()
 (:method MB1
  :precondition ()
  :tasks ((A1)
          ((and (>= ?start 3)(<= ?end 5)) (A2)))
))

;; Definición de la tarea A2 utilizada por las tareas
;; abstractas anteriores
(:task A2
 :parameters ()
 (:method A2
  :precondition ()
  :tasks ((a21)(a22))
))
```

Observar cómo en el método MB1 de la tarea del ejemplo B, se impone la restricción de que la tarea A2 se tiene que ejecutar en el intervalo de tiempo [3,5]. Estas restricciones son codificadas en el modelo temporal basado en redes de restricciones temporales STN explicado con anterioridad. Como también se explicó existe un mecanismo de herencia que hace que las tareas hijas en una red de tareas hereden las restricciones de sus tareas abstractas padre. De esta forma, en nuestro ejemplo si las tareas a21 y a22 durasen entre las dos (se tienen que ejecutar en secuencia) más de dos unidades temporales, se produciría una violación en la red

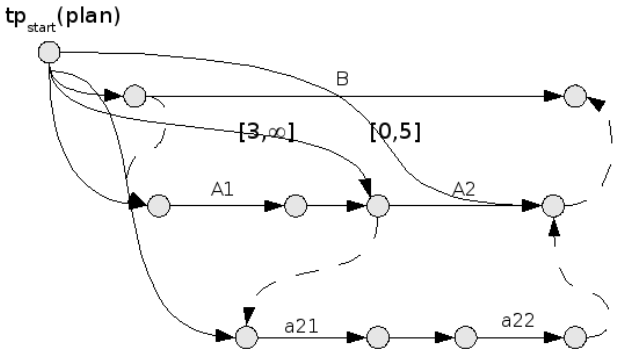


Ilustración 34: Herencia de restricciones y restricciones explícitas

STN que provocaría un backtracking en el algoritmo. En la Ilustración 34 con flechas punteadas se marcan las restricciones heredadas, mientras que con flechas

continuas se marcan las restricciones de orden y explícitas del ejemplo planteado.

3.6.4. - *Temporización de los objetivos*

El objetivo para un planificador HTN consiste en partir de una red de tareas objetivo e ir expandiendo todas sus tareas abstractas hasta que todas las hojas del árbol en expansión sean operadores primitivos.

En HTN-PDDL la definición de la red de tareas en un método es exactamente igual a definir la red de tareas objetivo. Por eso las tareas que forman la red de tareas a resolver pueden ser restringidas en el problema, de la misma forma que se restringen en la red de tareas de un método (de hecho la red de tareas objetivo podría verse como perteneciente a una tarea abstracta ficticia "*goal*" con un sólo método de descomposición).

3.6.5. - *Sincronización entre las tareas*

Mediante el orden definido en las redes de tareas y las restricciones explícitas impuestas sobre las mismas, se dispone de unos mecanismos de control sobre la red de tareas bastante adecuados, pero no se dispone de herramientas para controlar las relaciones entre dos redes de tareas distintas.

Los *timepoints ?start* y *?end* de una tarea también se pueden utilizar para definir sincronizaciones complejas entre tareas o límites máximos o mínimos en la distancia temporal entre dos tareas.

Internamente HTNP2 instancia las variables *?start* y *?end* como referencias a los *timepoints start* y *end* de su tarea asociada en la STN. Estas referencias tipo *timepoint* (o *landmarks*) forman parte de la estructura de tipos básica gestionada por HTNP2. De este modo se permite que formen parte de los argumentos de un literal y por lo tanto pueden ser insertadas en el estado de planificación para ser posteriormente ser consultadas más adelante en el proceso de planificación y servir para definir restricciones en otra red de tareas. Por comodidad los *landmarks* suelen ser definidos en la descripción de dominios como *fluents* que pueden ser utilizados directamente en los operadores de comparación al describir una restricción.

El siguiente ejemplo muestra la inserción en el estado del planificador de los landmarks:

```
Ejemplo:
;; inserción de los landmarks de una tarea abstracta y de un operador
primitivo en el estado
```

```
(:task A2
 :parameters ()
 (:method MA2
  :precondition ()
  :tasks ((:inline () (and (assign (start A2) ?start)
                           (assign (end A2) ?end)))
          (a21)
          (a22))
))

(:durative-action b
 :parameters()
 :duration (= ?duration 1)
 :condition()
 :effect(and (assign (start b) ?start)
         (assign (end b) ?end)
))
```

Observar cómo se hace uso de las acciones *inline* para introducir los *landmarks* de la tarea abstracta A2. Recordemos que las acciones *inline* se usan en HTN-PDDL para introducir conocimiento de control en el estado del planificador, como es la información temporal, y que estas acciones no son contempladas en el plan final, ni generan vínculos causales. Al operador primitivo b en cambio no le queda más remedio que insertar la información temporal como efectos.

Esta información temporal puede ser utilizada para definir restricciones en la red de tareas de un método distinto, como se aprecia en el siguiente ejemplo:

Ejemplo:

```
;; Uso de un landmark guardado con anterioridad para hacer una
;; sincronización.
(:task A3
 :parameters ()
 (:method MA3
  :precondition ()
  :tasks (((= ?start (start A2)) (b)))
))
```

En el ejemplo la tarea A3 en su método MA3 utiliza el landmark guardado en el ejemplo anterior del inicio de la tarea A2, para sincronizar el inicio de la acción b con dicha tarea.

Destacar el abanico de posibilidades que se presentan gracias a esta expresividad y al modelo basado en STN subyacente. En general en HTN-PDDL se pueden representar todas las restricciones recogidas en el álgebra de Allen [5], combinando tareas abstractas y operadores primitivos.

En la tabla 1 se muestran dichas relaciones, tomando como ejemplo, por un lado una tarea abstracta A2, que se descompone en la secuencia de operadores

primitivos a21 y a22 que tienen una duración de 1 unidad temporal y por otro lado un operador primitivo b que tiene una duración de 5 unidades temporales.

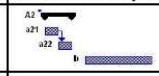
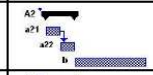
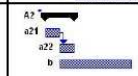
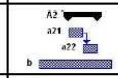
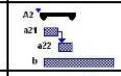
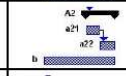
Allen's relation	SIADEX encoding	SIADEX output
A2 b **A2 BEFORE b**	((> ?start (end A2)) b)	
A2 b **A2 MEETS b**	((= ?start (end A2)) b)	
A2 b **A2 OVERLAPS b**	(and (> ?start (start A2)) (< ?start (end A2)) (> ?end (end A2)) b)	
A2 b **A2 DURING b**	(and (< ?start (start A2)) (> ?end (end A2)) b)	
A2 b **A2 STARTS b**	((= ?start (start A2)) b)	
A2 b **A2 FINISHES b**	((= ?end (end A2)) b)	
A2 b **A2 EQUAL b**	(and (= ?start (start A2)) (= ?end (end A2)) b)	

Tabla 1: Relaciones del álbebra de Allen expresadas en HTN-PDDL

3.7. - Algoritmos de propagación de restricciones "camino consistentes" (*path consistency*)

Dado el modelo temporal de HTNP2 expresado en una red de restricciones temporales TCN necesitamos disponer de un algoritmo de propagación de restricciones temporales. Dicho algoritmo es necesario por dos razones, primero para detectar posibles inconsistencias en la TCN durante la construcción del plan y segundo para hacer más eficiente la posterior búsqueda de una solución o schedule en el plan resultante.

Los algoritmos de propagación de restricciones trabajan de forma incremental, añadiendo una a una las nuevas restricciones a la red TCN, en cada paso nuevo conocimiento puede ser inferido de la red. En el momento que se infiere conocimiento que es inconsistente con el que existe previamente en la red, se dice que la red TCN se encuentra en estado inconsistente, lo que provocará una vuelta atrás o backtracking en el algoritmo de planificación. Esta forma de trabajar incremental se adecúa perfectamente a la forma en la que el planificador

construye el plan, que es también paso a paso, añadiendo una acción cada vez y enviando las restricciones asociadas a la acción a la TCN también una a una.

Se distinguen dos tipos de algoritmos de propagación de restricciones, los que se basan en la propiedad de "arco consistencia" y los que se basan en la propiedad de "camino consistencia" [110] .

Los algoritmos basados en arco consistencia infieren las restricciones basándose en parejas de variables (puntos temporales) de la red. La arco consistencia de una red asegura que dados un par de variables, y un rango de valores que pueden tomar cada una de las variables, para cualquiera de los valores que pueda tomar la primera variable existe otro valor válido dentro del rango de valores para la otra variable.

En cambio los algoritmos basados en camino consistencia (*path consistency*) infieren restricciones basándose en subredes que involucran tres variables. La camino consistencia asegura que cualquier solución consistente para una subred de tamaño dos sea extensible a una tercera variable.

En general se pueden construir algoritmos que garanticen la consistencia para subredes de tamaño i (i consistencia), el problema es que garantizar la i consistencia es exponencial en el tamaño de i.

HTNP2 usa un algoritmo de propagación de restricciones de camino consistencia llamado PC-2 (*Path Consistency 2*), que describiremos a continuación.

> **Definición 5:** (*Camino consistencia de una red STN*): Dado un plan temporal $\Pi = \sigma_1, \sigma_2, ..., \sigma_n$ y una red STN={X,D,C} construida sobre el, donde X es el conjunto de *timepoints* (start y end de todas las tareas), D es el rango donde esos *timepoints* son válidos (en principio $[0,+\infty)$) y C el conjunto de restricciones que el planificador va añadiendo en cada paso (Ec. 5, 6, 7, 8, 35, 36, 37), dos *timepoints* $\{x_i, x_j\}$ son camino consistentes con relación a una variable x_k si y sólo si para cada asignación de un instante temporal $\{(x_i,t_i), (x_j,t_j)\}$ existe otro instante temporal $t_k \in D_k$ tal que las asignaciones $\{(x_i,t_i), (x_k,t_k)\}$ y $\{(x_k,t_k), (x_j,t_j)\}$ son consistentes. De forma alternativa, una restricción binaria entre los *timepoints* $\{x_i, x_j\}$ R_{ij} es camino consistente con respecto a x_k si y solo si para cada pareja $(t_i,t_j) \in R_{ij}$ que verifique la restricción hay un valor t_k tal que las parejas $(t_i,t_k) \in R_{ik}$ y $(t_k,t_j) \in R_{kj}$ son también consistentes con la restricción.

> **Definición 6:** (*Camino consistencia local*): Una subred de tres variables $\{x_i, x_j, x_k\}$ es camino consistente si y solo si para cada

permutación de (i,j,k), R_{ij} es camino consistente con relación a x_k.

Definición 7: (*Red STN camino consistente*): Una red STN es camino consistente si para cada restricción R_{ij} y para cada $k \neq i,j$ R_{ij} es camino consistente con relación a x_k.

Definamos un procedimiento llamado REVISE-PC que toma como argumentos una restricción R*ij* entre dos timepoints i,j y una tercera variable k y devuelve la restricción mas relajada que satisface la propiedad de camino consistencia. REVISE-PC puede ser calculado utilizando la siguiente fórmula:

$$(64)\quad R'_{ij} = R_{ij} \cap (R_{ik} \circ R_{kj})$$

Usando esta función podemos definir un algoritmo de fuerza bruta que recibe el nombre de PC-1 que llama a REVISE-PC para cada tripleta de variables hasta que no se produce ningún cambio en la red STN.

```
PC-1(STN)
1.  repeat:
2.     for k=1 to n:
3.        for i,j=1 to n:
4.           REVISE-PC(STN,(i,j),k)
5.  until no changes in STN
```

Algoritmo 8: Path Consistency 1

Se demuestra [110] que PC-1 tiene una eficiencia de $O(n^5)$, siendo n el número de *timepoints* en la red STN. Existe un algoritmo más eficiente que es en el que se basa el algoritmo de propagación de restricciones temporales de HTNP2 llamado PC-2 (*Path Consistency 2*). PC-2 utiliza una cola que mantiene en cada ciclo exclusivamente las tripletas de variables que es necesario chequear porque pueden quedar inconsistentes al cambiar una restricción. Inicialmente la cola contiene todas las tripletas de variables de la red.

```
PC-2(STN)
1.  Q ← {(i,k,j), 1≤i<j≤n, 1≤k≤n, k≠i, k≠j}
2.  while not empty Q:
3.      (i,k,j) = pop_front(Q)
4.      REVISE-PC((i,j),k,STN)
```

```
5.      if changed(Rij) then:
6.       for l= 1 to n:
7.          if l!= i and l != j then:
8.              Q←Q∪{(l,i,j)(l,j,i),∀l 1≤l<n,l≠i,l≠j}
```

Algoritmo 9: Path Consistency 2

PC-2 es bastante más eficiente que PC-1, en concreto se puede demostrar [110] que su eficiencia es $O(n^3)$ donde n marca el número de *timepoints* en la STN.

3.8. - Modificaciones sobre el algoritmo básico HTNP para dar soporte temporal

Una vez que disponemos del modelo temporal y del algoritmo de propagación de restricciones, pasaremos a describir las modificaciones que se han llevado sobre el algoritmo básico de HTNP y que nos llevan hasta la definición algoritmo HTNP2.

Recordar que el algoritmo HTNP mantiene una pila de *contextos* por los que va pasando el planificador a fin de simular la recursividad que almacenan los cambios sufridos en un paso por el plan en expansión Π_i, en el estado ε_i y en otras estructuras auxiliares en un paso de ejecución. Es necesario ahora ampliar la información almacenada en un contexto ω_i a fin de poder deshacer no sólo los cambios en el estado y en el plan en expansión sino también en la red STN.

Añadimos al *contexto* ω_i la siguiente información:

- STP_{size}: Que mantiene el número de variables temporales *timepoints* de la red temporal en el contexto anterior.
- $intervals_{pending}$: En el caso de una tarea que haga uso de un literal periódico en el tiempo, se generan una serie de intervalos temporales durante los cuales ese literal es cierto (literales de la forma *(between* t_1 *and* t_2 *and every* t_n *(l))*) . Esta es la lista de intervalos que quedan por intentar.
- $intervals_{explored}$: La lista de intervalos en el *timeline* del literal ya probados.
- λ_{size}: Almacena el tamaño de la tabla de vínculos causales, para poder devolverla a su estado anterior al hacer backtracking.

Como se ha estudiado se van añadiendo a la STN restricciones una a una

mientras no se detecten inconsistencias, pero es necesario también disponer de un algoritmo que permita devolver a la red a su estado anterior, ya sea porque se detectó una inconsistencia o por que se vuelve al contexto ω_i por un backtracking provocado en cualquier otro punto de decisión.

Al mismo tiempo que se añaden restricciones a la STN y se verifican usando PC-2, se van añadiendo a una pila la información necesaria (matriz de adyacencia) para poder devolver la red a su estado anterior. Llamaremos undoSTP(STP_{size}) al algoritmo que realiza dicha vuelta atrás.

Las listas de intervalos se utilizan para gestionar un nuevo punto de decisión del algoritmo que se discutirá más adelante.

El bucle principal de HTNP2 es exactamente igual que el de HTNP con la única diferencia de que al realizar el undo (backtrack) de un contexto ω_i también se llama a la función undoSTP. El algoritmo de planificación recibe como entradas el dominio $\mathcal{D}$=<$\mathcal{V}$,$\mathcal{C}$,$\mathcal{P}$,δ,$\mathcal{A}$> y el problema $\mathcal{P}$ = <ε_0, Γ_g> y en pseudocódigo se describe de la siguiente forma:

```
HTNP2(<V, C, P, δ, A>, <ε0, Γg>)
1.  // Inicialización de ω0
2.  αi^pending = first(Γg) //Inicializar la agenda con la red de
    tareas objetivo. (Las acciones de la red de tareas que no
    tienen dependencias de orden).
3.  Π0 = Γg  //El plan en expansión es la red de tareas objetivo
4.  push(stack,ω0)
5.  while not empty(stack) do: // Mientras queden contextos
    por analizar
6.          ωi= top(stack)
7.          ωi+1 = generate_new_context2(ωi) // Generar a partir del
            contexto actual uno nuevo (dar un paso en la
            planificación) tomando las decisiones que sean
            adecuadas.
8.          if ωi+1 == FAIL then: // No quedan decisiones por
            tomar o ninguna es válida
9.                  undo(ωi) // deshacer los cambios provocados en
                    el plan en construcción y en el estado
10.                 pop(stack) // backtrack
```

```
11.        else:
12.              if empty(α_{i+1}^pending) then: // en el contexto que se
                 acaba de generar no quedan acciones por
                 expandir en la agenda, eso significa que
                 tenemos un plan
13.                    return Π_{i+1} // devolver el plan
14.              else: // se dio un paso correctamente
15.                    push(stack,ω_{i+1})
16.  return FAIL // no se encontró plan
```

Algoritmo 10: bucle principal de HTNP2, en color más oscuro se marcan las líneas que cambian sobre el algoritmo HTNP

Las principales modificaciones se encuentran en el algoritmo de generación de contextos que, recordemos, tenía los siguientes puntos de decisión:

- Escoger otra posible unificación de la lista de pendientes $\mathcal{U}_i^{pending}$.
- Escoger un nuevo método si estamos descomponiendo una tarea compuesta de la lista $\mu_i^{pending}$.
- Escoger otra acción de las que unifiquen con la acción actualmente seleccionada en la agenda $\rho_i^{pending}$.
- Escoger una nueva acción de las disponibles en la agenda $\alpha_i^{pending}$.

Ahora se tienen que tener en cuenta también los nuevos puntos de decisión que se crean por la inferencia temporal. En concreto el punto de decisión lo genera la posibilidad de anclar una precondición que hace uso de un literal periódico en distintos intervalos temporales del período. Así pues la nueva decisión es:

- Escoger un nuevo intervalo de entre los posibles si la acción hace uso de un literal periódico.

Teniendo en cuenta este nuevo punto de decisión y la necesidad de actualizar la STN con las restricciones temporales, la nueva función generate_new_context2, se define como sigue:

```
generate_new_context2(ω_i)
1.  while not empty(α_i^pending) then:
2.     α_j = first_of(α_i^pending)
3.     if empty(ρ_i^explored) and empty(ρ_i^pending) then:
4.        ρ_i^pending = matching_actions(α_j,δ)
```

```
5.     while not empty(ρi^pending) then:
6.       ρj = first_of(ρi^pending)
7.       if compound(ρj) then:
8.         if empty(μi^pending) and empty(μi^explored) then:
9.           μi^pending = generate_methods(ρj)
10.        while not empty(μi^pending) then:
11.          μj = first_of(μi^pending)
12.          if empty(U_i^explored) and empty(U_i^pending) then:
13.            U_i^pending = generate_unifications(μi,εi)
14.          while not empty(U_i^pending) then:
15.            U_j = first_of(U_i^pending)
16.            postFAIL = PostSTP(ρj,Uj,Γj)
17.            if not postFAIL then:
18.              Π^(i+1) = replace(Πi,ρj,Γj in μj) //Añadir nuevas
                 tareas al plan en construcción
19.              α_(i+1)^pending = αi^pending - ρj + first(Γj) //Actualizar la
                 agenda
20.            delete(first_of(U_i^pending))
21.            return  ωi+1
22.        clear(U_i^explored), clear(U_i^pending) //Se exploraron todas las
           unificaciones se prueba un nuevo método
23.        delete(first_of(μi^pending))
24.      else: // es un operador primitivo
25.        if empty(U_i^explored) and empty(U_i^pending) then:
26.          U_i^pending = generate_unifications(μi,εi)
27.        while not empty(U_i^pending) then:
28.          U_j = first_of(U_i^pending)
29.          if timelined(U_j) and empty(intervals) then:
30.                  intervals_pending = generate_timeline(U_j)
31.          if timelined(U_j)
```

```
32.                while not empty(intervals_pending) then:
33.                   interval = top(intervals_pending)
34.                   postFAIL = PostSTP(ρ_j,U_j,Γ_j,interval)
35.                   if not postFAIL then:
36.                      α_{i+1}^pending = α_i^pending − ρ_j //Actualizar la agenda
37.                      ε_{i+1} = ε_i + apply_effects(ρ_j)
38.                      push(intervals_explored,pop(intervals_pending))
39.                      return  ω_{i+1}
40.                   push(intervals_explored,pop(intervals_pending))
41.             else:
42.                   postFAIL = PostSTP(ρ_j,U_j,Γ_j)
43.                   if not postFAIL then:
44.                      α_{i+1}^pending = α_i^pending − ρ_j //Actualizar la agenda
45.                      ε_{i+1} = ε_i + apply_effects(ρ_j)
46.                      delete(first_of(U_i^pending))
47.                      return  ω_{i+1}
48.          delete(first_of(ρ_i^pending))
49.     clear(ρ_i^pending), clear(ρ_i^explored) //Se exploraron todas los
        matching con la agenda. Probar una nueva acción de la
        angenda
50.     delete(first_of(α_i^pending))
51.  return FAIL
```

Algoritmo 11: Generar un nuevo contexto en HTNP2, en color más oscuro se marcan las líneas que cambian sobre el algoritmo HTNP

Destacar que el modelado de los literales del estado es también distinto en HTNP2, ya que éstos están temporizados. Por eso las funciones de aplicación de los efectos y de unificación son ligeramente distintas. La función de unificación además también actualiza la tabla de vínculos causales cada vez que se hace uso de un operadorprimitivo.

La principal novedad del algoritmo está en la función que realiza actualización de la red STN PostSTP de la que se presenta su algoritmo de forma general a continuación:

Sea:

- STP: La estructura global que mantiene la estructura temporal.
- λ: La tabla global de vínculos causales

```
PostSTP(ρi,𝒰i,Γi,[interval])
1.  if not compound(ρi) then:
2.     dur = duration(ρi)
3.     ρi.start = STP.registerTP()
4.     ρi.end = STP.registerTP()
       //Fijar los timepoints de la tarea
5.     STP.addTConstraint(ρi.start,ρi.end,[dur,dur]) or FAIL
       //Fijar la duración de la tarea
6.     STP.addTConstraint(ρi.start,parent(ρi).start,[0,∞]) or FAIL
7.     STP.addTConstraint(parent(ρi).end,ρi.end,[0,∞]) or FAIL
       //Añadir las restricciones de herencia de timepoints con
       la tarea padre
8.     foreach ρj in first(Γi,ρi) then:
9.        STP.addTConstraint(ρj.start,ρi.start,[0,0]) or FAIL
10.    foreach ρj in last(Γi,ρi) then:
11.       STP.addTConstraint(ρi.end,ρj.end,[0,0]) or FAIL
          //Añadir las restricciones de orden con las
          predecesoras y las sucesoras en la red de tareas
12.    if interval then:
13.       STP.addTConstraint(ρi.start,interval.start,[0,∞]) or
          FAIL
14.       STP.addTConstraint(interval.end,ρi.end,[0,∞]) or FAIL
          //Tratar de anclar con el timeline de los eventos
          periodicos
15.    foreach l in λ(ρi,𝒰i) then:
16.       STP.addAppropiateTConstraint(l) or FAIL
          //Añadir las restricciones temporal apropiada
          dependiendo del vínculo
17.    return SUCCESS
```

```
18. else:
19.     ρi.start = STP.registerTP()
20.     ρi.end = STP.registerTP()
        //Fijar los timepoints de la tarea
21.     STP.addTConstraint(ρi.start,parent(ρi).start,[0,∞]) or FAIL
22.     STP.addTConstraint(parent(ρi).end,ρi.end,[0,∞]) or FAIL
        //Añadir las restricciones de herencia de timepoints con
        la tarea padre
23.     foreach C in Γi then:
24.         STP.addAppropiateTConstraint(C) or FAIL
            //Añadir las restricciones temporal adecuada a la
            restricción definida en la red de tareas del método
            aplicado
25.     return SUCCESS
```

Tabla 2: PostSTP registro de restricciones temporales en la red STN

3.9. - Scheduling

Una vez que disponemos de un plan es necesario asignarle un scheduling o asignación precisa de tiempos a los operadores primitivos que lo componen.

En la red STN cada arco que une dos timepoints i,j está etiquetado con el intervalo $[a_{ij}, b_{ij}]$ que representa la restricción:

$$(65)\quad a_{ij} \leq x_j - x_i \leq b_{ij}$$

O bien las desigualdades:

$$(66)\quad x_j - x_i \leq b_{ij}, x_i - x_j \leq -aij$$

Podemos transformar la red STP resultante en un grafo dirigido ponderado llamado grafo de distancias. Cada arco del grafo, que une dos timepoints i,j es etiquetado con el valor a_{ij}, que representa la desigualdad $x_j - x_i \leqslant t_{ij}$. Por ejemplo dado el grafo de la Ilustración 35, la distancia entre los timepoints 1 y 2 es de [30,40].

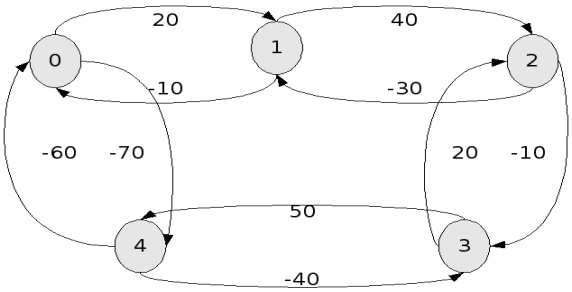


Ilustración 35: Grafo de distancias

Para cualquier distancia desde entre dos puntos i j en el grafo de distancias se induce la siguiente restricción en la distancia $x_j - x_i$:

$$(67)\quad x_j - x_i \leq \sum_{j=1}^{k} a_{i(j-1)}, i_j$$

Si hay más de un camino desde i hasta j, entonces se verifica que la intersección de todas las restricciones inducidas cumple:

$$(68)\quad x_j - x_i \leq d_{ij}$$

Donde d_{ij} es la distancia del camino más corto desde i hasta j. De esta forma se puede demostrar [80] [121] que una red STN es consistente si y solo si el grafo de distancias no tiene ciclos negativos.

Así pues podemos calcular un scheduling (solución de la red STN), si la siguiente la tupla ($x_1 = d_{01}$, $x_2 = d_{02}$, ..., $x_n=d_{0n}$) que es una asignación de tiempos concretos a los timepoints, no genera un ciclo negativo. Esta asignación es además la asignación (schedule) que ejecuta las acciones del plan lo más pronto posible.

La asignación inversa que asigna las distinacias negativas más cortas es también una solución que ejecuta las acciones lo más tarde posible ($x_1 = -d_{10}$, $x_2 = -d_{20}$, ..., $x_n=-d_{n0}$).

Podemos calcular un grafo de distancias mínimas aplicando el algoritmo de Floyd-Warshall [22]. Este algoritmo tiene una eficiencia $O(n^3)$ en el peor de los casos y detecta inconsistencias (ciclos negativos) en la red STN simplemente comprobando el signo en la diagonal principal de la matriz de adyacencia. Una vez que se dispone del grafo de distancias calcular una solución se puede realizar en un tiempo acotado por $O(n^2)$, simplemente chequeando cada asignación con las asignaciones realizadas previamente.

```
ALL-PAIRS-SHORTEST-PATHS(STN)
1.  for i=1 to n: d_ij=0
2.  for i,j=1 to n: d_ij = a_ij
3.  for k=1 to n:
4.     for i,j=1 to n:
5.        d_ij = min(d_ij, d_ik + d_kj)
```

Algoritmo 12: Algoritmo de Floyd-Warshall para el cálculo de la matriz de distancias de un grafo

Así pues con el algoritmo de Floyd-Warshall podemos calcular el momento preciso en que se puede ejecutar una acción y también cuando esta terminará. Este mismo algoritmo se puede utilizar para realizar un *reschedule* del plan cuando ocurre un retraso.

3.10. - Floyd-Warshall vs PC-2

Como se ha visto el propio algoritmo de Floyd-Warshall se puede utilizar para propagar las restricciones temporales en lugar del algoritmo PC-2. Floyd-Warshall está acotado en $O(n^3)$ donde n es el número de timepoints al igual que PC-2. Se podría pensar entonces en utilizar el algoritmo de Floyd-Warshall en lugar de PC-2, más teniendo en cuenta que Floyd-Warshall tiene una serie de ventajas.

Floyd-Warshall es un algoritmo completo y se puede utilizar para determinar la red STN mínima. Si hay una inconsistencia siempre la detecta. Por el contrario PC-2 es un algoritmo aproximado. PC-2 es un algoritmo aproximado porque como se ha mostrado utiliza propagación de restricciones local y en esto estriba precisamente su ventaja frente a Floyd-Warshall.

Floyd-Warshall es un algoritmo que propaga *siempre* todas las restricciones en la red por lo tanto en media es siempre un algoritmo de orden $O(n^3)$, mientras que PC-2 solamente propaga las restricciones locales (en el peor de los casos puede propagarlas todas y ser incluso más lento que Floyd-Warshall) con lo que en promedio es un algoritmo muy rápido. Además se demuestra que PC-2 detecta todas las posibles inconsistencias, es decir toda inconsistencia detectada por Floyd-Warshall es también detectada por PC-2.

3.11. - Comparativa con otros planificadores temporales

El uso de redes STN para la codificación de restricciones temporales en planificación no es nueva y existen diversos planificadores que hacen uso de ellas.

En planificación jerárquica HTN uno de los primeros planificadores que hacen uso de ellas es Oplan [133] en que a diferencia de HTNP2 la mayoría de las restricciones temporales tenían que codificarse de forma explícita en la red STN. En HTNP2 solamente se hacen explícitas las restricciones de sincronización de tareas de redes distintas, el resto de restricciones son manejadas de forma automática por el planificador simplificando mucho la escritura de dominios.

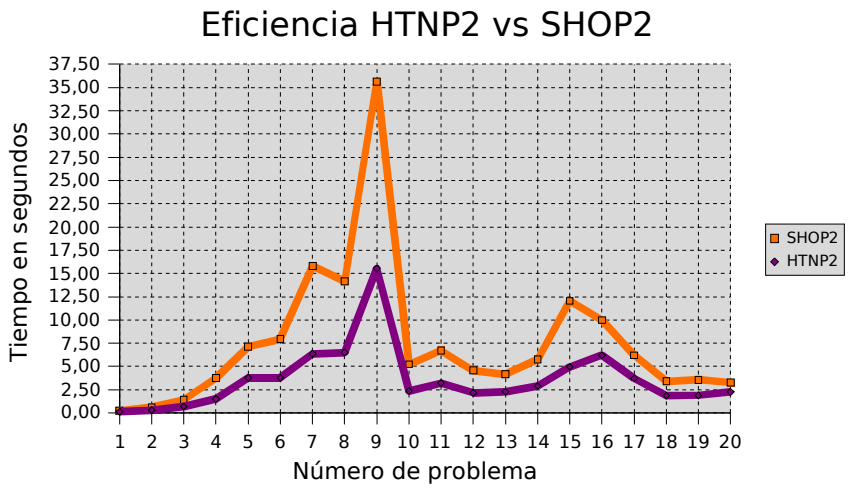
SHOP2 [28] no hace uso de una red STN, pero provee un mecanismo llamado MTP (*Muti-Timeline Preprocessing*) que mediante el uso de literales *fluents* en el estado va calculando el *schedule* del plan. SHOP2 es incapaz de calcular múltiples *schedules* como HTNP2 y es incapaz de representar *deadlines* y hacer sincronización entre tareas, además que el peso de llevar el cálculo de tiempo recae sobre el escritor de dominios. De esta forma HTNP2 produce un conjunto de *schedules* flexibles que se pueden adaptar a posibles retrasos, mientras que SHOP2 calcula un único schedule [72].

Además en SHOP2 el plan obtenido es lineal, mientras que HTNP2 gracias al uso que hace de la estructura de vínculos causales el plan obtenido es un plan parcial.

Ixtet [79] es un planificador que también hace uso de redes STN. Las relaciones temporales causa efecto son codificadas mediante dos predicados sencillos que representan eventos o cambios instantáneos en el estado y *assertions* que representan literales que se mantienen a lo largo del tiempo.

A pesar del costo que supone al algoritmo de planificación la gestión del modelo temporal y la propagación de las restricciones temporales asociadas HTNP2 sigue siendo un planificador bastante eficiente. La Gráfica 2 muestra una comparativa de uso de tiempo de CPU entre el planificador HTNP2 y SHOP2. Se ha escogido este planificador por que es el que quizás es más parecido a HTNP2 en cuanto a la estructura del dominio y por tanto se pueden escribir dominios que son aproximadamente equivalentes. En el eje x se muestran distintos problemas para el dominio del *zeno-travel (hard time)* temporal adaptado por SHOP2 para su participación en la IPC del 2002 mientras que en el eje y se muestra el tiempo consumido por los planificadores en segundos. Se puede apreciar claramente que HTNP2 es más rápido que SHOP2 en todos los dominios. Puesto que los dominios de HTNP2 se han escrito siguiendo la estructura de los dominios utilizados por SHOP2

(no se ha hecho uso de expresiones o construcciones en HTNP2 no soportadas por SHOP2 que pudieran acelerar el tiempo de procesamiento) también se aprecia como las curvas son simétricas.



Gráfica 2: Comparativa en tiempo de CPU entre HTNP2 y SHOP2

4. - *Mejorando la eficiencia del planificador temporal*

Un objetivo sin un plan es sólo un deseo.

***Antonie de Saint-Esupery**, (1900-1944), escritor francés.*

Durante el capítulo 3 se ha presentado un modelo de planificación temporal basado en el paradigma de planificación jerárquica HTN y en el uso de redes STN. Se ha presentado el planificador HTNP2 que hace uso de un algoritmo de satisfacción de restricciones PC-2 (*Path Consistency* 2) que se basa en mantener la propiedad de camino consistencia de las redes STN.

A pesar de que se ha demostrado que PC-2 es un algoritmo eficiente en problemas complejos, la carga computacional necesaria para mantener el modelo temporal es elevada y sería deseable disponer de algoritmos más eficientes. En este capítulo se presenta el algoritmo de planificación HTNP2CL que hace uso de una inferencia temporal distinta basada en la estructura causal del plan que lo hace más eficiente.

4.1. - Usando la estructura del plan para mejorar la eficiencia

El algoritmo PC-2 es un algoritmo de propagación de restricciones incremental. Cada vez que se añade una restricción temporal a la red temporal puede ocurrir uno de los siguientes tres casos:

- Se reduce el intervalo que restringe la distancia entre dos *timepoints,* mediante la intersección con la nueva restricción.
- Se añade un nuevo arco entre dos *timepoints* con la nueva

restricción.

- Se añade un nuevo nodo (*timepoint)* a la red STN y uno o más arcos que lo conectan con el resto de la red.

Una vez hecho esto es necesario propagar los cambios aplicados al resto de la red para comprobar si esta sigue siendo consistente.

Como se explicó PC-2 utiliza una estructura tipo cola para ir almacenando las restricciones que es preciso revisar. PC-2 hace propagación local en caminos de longitud dos (parten de un timepoint i pasan por uno intermedio j y terminan en otro k) para detectar inconsistencias. Cada vez que una restricción es modificada en la red STN, todos los *timepoints* y sus conexiones con la restricción modificada son revisadas, de esta forma la modificación se va extendiendo al resto de la red:

```
5.      if changed(Rij) then:
6.        for l= 1 to n:
7.           if l!= i and l != j then:
8.              Q←Q∪|(l,i,j)(l,j,i),1≤l<n,l≠i,l≠j|
```

Algoritmo 13: Revisión de restricciones en PC-2

Por ejemplo dada la red temporal mostrada en la Ilustración 36, con 8 timepoints, que se encuentra asociada a un plan simplificado con tres acciones primitivas, si suponemos que se acaba de añadir una restricción provocada por un vínculo causal entre las acciones a11 (efecto at-end) y a12 (precondición at-start) hay que añadir a la cola las tripletas {(0,3,4), (1,3,4), (2,3,4), (5,3,4), (6,3,4), (7,3,4), (0,4,3), (1,4,3), (2,4,3), (5,4,3), (6,4,3), (7,4,3)}. Al testear el camino que pasa cada una de las tripletas en la cola, se pueden añadir nuevas restricciones o modificar las existentes, con lo que se pueden añadir nuevas tripletas a la cola, aumentando el procesamiento necesario.

Sin embargo se puede evitar testear muchas de estas tripletas si se tiene en cuenta que existen tareas que son causalmente independientes. En la Ilustración 36, las acciones a11 y a21 son causalmente independientes (no existe ningún vínculo entre ellas), lo mismo que a12 y a21 también son causalmente independientes. Por lo tanto si se añade o se modifica una restricción que afecta a las tareas a11 y a 12, como es el caso, las restricciones impuestas sobre a21 nunca se verán afectadas. Se puede por esto evitar introducir las tripletas {(6,3,4), (7,3,4), (6,4,3), (7,4,3)} en la cola, reduciendo el número de comprobaciones y la cola del algorimo PC-2 en un 30%.

En esta idea sencilla se apoya el algoritmo PC-2-CL (*Path Consistency 2 with Causal Links*) [74] [73] que se explicará más detalladamente a continuación.

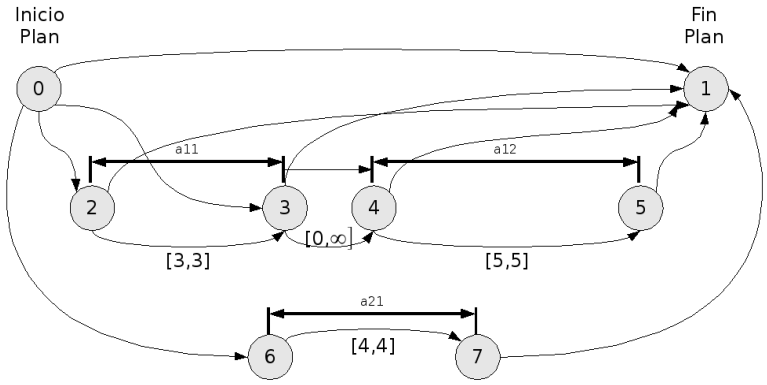


Ilustración 36: Red temporal asociada a un plan simplificado. Los arcos no etiquetados se supone que tienen una restricción [0,∞]

4.2. - Tipos de dependencia entre tareas

La clave del algoritmo PC-2-CL es su capacidad para discernir que caminos de longitud 2 es necesario comprobar y cuales no en base a las dependencias de las tareas.

Dado un *timepoint* de la red temporal pueden ocurrir los siguientes casos:

- Es el *timepoint* que marca el inicio o el final del plan (el 0 y el 1 respectivamente en el ejemplo de la Ilustración 36).
- Es un *timepoint* perteneciente a un operador primitivo, bien el que marca el inicio o el fin del mismo.
- Es un *timepoint* perteneciente a una tarea abstracta, nuevamente el que marca el inicio o el fin.
- O bien es un *timepoint* que pertenece al *timeline.*

PC-2-CL es capaz de determinar a quién pertenece el *tiempoint* y en base a ello determinar si debe o no añadir o no la tripleta a la cola. Dadas las tripletas (l,i,j) y (l,j,i):

Relaciones causales:

- Si l es un *timepoint* que pertenece a un operador primitivo σ_l y i

pertenece a otro operador primitivo σ_i, y existe un vínculo causal en λ que relaciona a ambos operadores primitivos, entonces las tripletas deben incluirse en la cola para considerar si se incumple alguna restricción.

- También se añade la tripleta a la cola si uno de los timepoints pertenece al *timeline* y el otro a un operador primitivo.

Relaciones de orden:

- Dados dos timepoints pertenecientes cada uno a una tarea, si hay una restricción de orden entre las tareas entonces también se añaden a la cola.

Landmarks:

- Los timepoints involucrados en una restricción explícita sobre el inicio y la finalización de una tarea son también considerados.

Con estas reglas, al añadir una nueva restricción dicha restricción únicamente se propaga sobre las ramas del plan que tienen una restricción causal, sobre la estructura jerárquica que soporta dicha rama, y sobre las restricciones con el timeline y las restricciones explícitas.

4.3. - El algoritmo HTNP2CL

El algoritmo HTNP2CL, es igual a HTNP2, soporta la misma expresividad temporal y tiene la misma estructura. La única diferencia se encuentra en el algoritmo de propagación de restricciones:

```
PC-2-CL(STN,λ)
1.   Q ← [(i,k,j), 1≤i<j≤n, 1≤k≤n, k≠i, k≠j]
2.  while not empty Q:
3.      (i,k,j) = pop_front(Q)
4.      REVISE-PC((i,j),k,STN)
5.      if changed(Rij) then:
6.       for l= 1 to n:
7.          if l!= i and l != j then:
8.             if l∈l_k, l_k∈λ, (i∈l_k ∨ j∈l_k) or
                  l∈constraints(l,i) ∨ l∈constraints(l,j) then:
```

9. $Q \leftarrow Q \cup \{(l,i,j)(l,j,i), 1 \leq l < n, l \neq i, l \neq j\}$

Algoritmo 14: PC-2 Causal Links

Donde *constraints* es un conjunto que mantiene las restricciones directas entre dos timepoints, añadidas por una ordenación, por el *landmarking* o por el anclaje con el *timeline.*

PC-2-CL es más eficiente en media que PC-2 puesto que evita comprobaciones innecesarias en partes de la red que son independientes. En el peor de los casos en el que existieran restricciones impuestas por vínculos causales o por cualquier otra razón, de todos, con todos los nodos de la red, PC-2-CL es menos eficiente que PC-2 por el costo de realizar comprobaciones contra la estructura causal. Sin embargo este caso es imposible, puesto que un plan en donde existieran restricciones de todas las acciones con todas las acciones impuestas por vínculos causales es un plan inválido. En general cuanto menos denso es el grafo de restricciones impuestas sobre la red STN, o lo que es lo mismo, a menor número de arcos explícitos añadidos a la red STN (no inferidos), PC-2-CL será mucho más eficiente que PC-2.

4.4. - Experimentación y evaluación de los distintos algoritmos

Usaremos distintos dominios de planificación basados en problemas reales para realizar las comparativas de los distintos algoritmos entre sí y para su comparación con otros planificadores, se estudiarán las características principales de cada uno de estos dominios.

4.4.1. - *Extinción de incendios: SIADEX*

El primer dominio que se usará como banco de pruebas, es un dominio extraído de los usados en la extinción de proyectos forestales en el proyecto SIADEX [75]. Los dominios de SIADEX hacen además uso intensivo de la expresividad presentada en el modelo temporal.

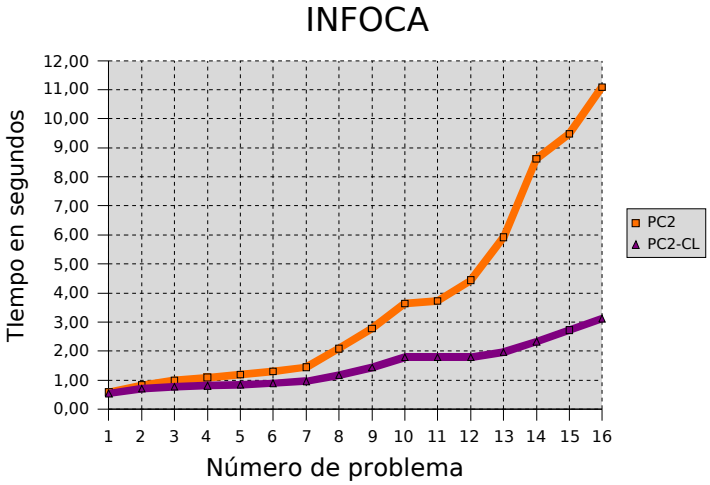
- Como es normal todas las acciones tienen que tener una temporización clara y todo tiene que estar coordinado en el tiempo.
- Se hace uso del *timeline* ya que hay gran cantidad de eventos exógenos que influyen sobre el plan. Por ejemplo eventos provenientes de

previsiones meteorológicas, tales como "previsto viento fuerte a partir de las 18:00 y hasta las 4:00 del día siguiente" (intervalar), del periodo contratado de los vehículos aéreos (intervalar), de los turnos de los grupos especialistas (cíclico), del día y la noche (cíclico), avituallamiento (ciclico) ...

- Es necesario realizar sincronizaciones complejas entre tareas, para ello se hace uso del landmarking y de las restricciones impuestas en la red de tareas asociadas a un método de expansión. Un ejemplo es el despliegue de un grupo especialista aerotransportado. El grupo especialista sale de la base de operaciones con un helicóptero que los deja en una zona de actuación. A partir de este momento el helicóptero y el grupo especialista se desvinculan y cada uno trabaja de forma independiente. Es necesario sincronizar el fin de las horas de trabajo del grupo especialista, con la preparación de su relevo, el transporte a la zona por otro vehículo y la retirada del grupo que estaba desplegado. También es necesario sincronizar los repostajes y las cargas de extintor de los vehículos aéreos.

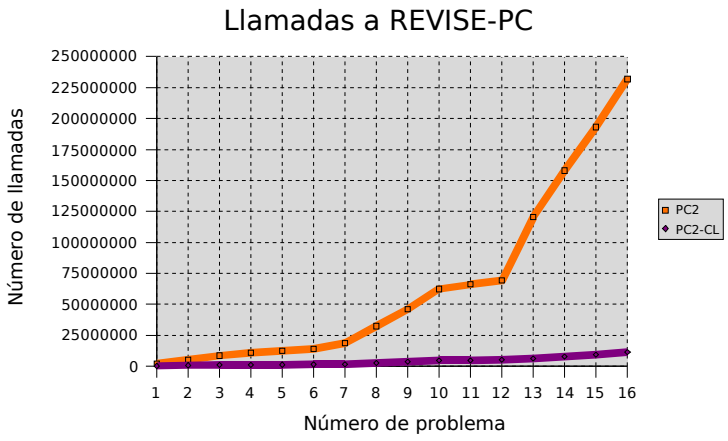
Estas características unidas al tamaño, en número de acciones, de los planes que se obtienen en SIADEX hacen que la red STN sea grande y se necesite un algoritmo de planificación eficiente que se apoye en una técnica de propagación de restricciones temporales que también sea eficiente.

El dominio utilizado para la experimentación es sencillo. Únicamente hace un despliegue al incendio de los medios situados en las bases más cercanas que actúan durante un tiempo y se repliegan. Cada experimentación se realiza tres veces de las cuales en la gráfica mostraremos su tiempo medio. En cada uno de los problemas para aumentar la complejidad lo que se hace es aumentar el número de medios que se despliegan en el incendio. Para realizar la experimentación se ha usado un Pentium IV 2.0 Ghz con un S.O GNU/Linux. Las distintas versiones de HTNP están implementadas en C++ y los dominios están escritos en HTN-PDDL.



Gráfica 3: Dominio INFOCA

La Gráfica 3 muestra los resultados de las distintas variante de HTNP sobre este dominio. Los dominios del plan INFOCA tienen como característica relevante que el plan suele tener bastantes ramas independientes, ya que los distintos



Gráfica 4: Llamadas a REVISE-PC de PC-2 y de PC-2-CL

medios en el incendio trabajan de forma bastante autónoma. Esta característica hace que PC-2 que hace uso de la estructura del plan para evitar propagar en ramas que son independientes funcione bastante más eficientemente que PC-2.

Para analizar el ahorro en eficiencia de PC-2-CL frente a PC-2 se ha añadido un contador que recoge el número de llamadas a la rutina REVISE-PC. Como se puede apreciar en la gráfica 4, el número de llamadas es sensiblemente superior en PC-2 que en PC-2-CL, en concreto para el problema 16 que es el más grande PC-2 realiza 231.836.728 llamadas mientras que PC-2-CL realiza 11.223.340 lo que supone aproximadamente un 95% más de llamadas.

Por último destacar que el dominio INFOCA es bastante complejo y aún así los tiempos obtenidos por las diferentes versiones del planificador son muy competitivos. El tamaño del plan en número de acciones primitivas que lo componen para cada uno de los problemas se muestra en la tabla 3.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
55	92	98	106	114	122	130	167
9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
200	233	241	248	268	306	339	372

Tabla 3: Número de acciones primitivas para los problemas INFOCA

4.5. - *Turismo con SAMAP*

SAMAP [85] [114] (*adapative multi-agent planning systems dependent on context*) es un sistema basado entre otras en técnicas de planificación que trata de aprovechar la creciente cantidad de servicios web orientados al turismo existentes en Internet, para generar planes de visita automáticos adaptados a las preferencias del usuario. El sistema actúa en tres etapas.

1 Se extrae un perfil de usuario. Para ello el sistema a través de una serie de formularios web, trata de extraer información relevante del usuario, tal como aficiones y gustos.

2 A partir de esta información y la existente en la base de conocimiento que contiene otros usuarios con preferencias similares se construye un modelo de usuario.

3 Se construye un plan de visita para visitar una ciudad seleccionada por el usuario.

El construir este tipo de planes constituye un reto interesante para un planificador ya que:

- Es necesario gestionar valores numéricos (fluents), por ejemplo el

dinero que tenemos disponibles y los precios de las visitas y comidas.

- Es necesario que el planificador o un módulo auxiliar construya rutas, no es suficiente con decir que el usuario se debe desplazar a un lugar, sino que es indispensable explicarle que ruta tiene que seguir para hacerlo.

- Los objetivos pueden ser especificados con distinto nivel de abstracción, desde quiero visitar a un lugar concreto hasta me gustaría visitar museos.

- Es necesario gestionar información temporal para controlar los horarios de apertura cierre de los distintos lugares de interés así como las horas para la comida y el descanso.

Hemos construido un dominio en HTN-PDDL para resolver este tipo de problemas planteados en SAMAP ya que consideramos que el modelo temporal y de planificación utilizado por HTNP2 y HTNP2CL se adapta perfectamente.

1 El manejo de literales numéricos en HTN-PDDL es lo suficientemente rico para cubrir todas las necesidades de cálculos numéricos del dominio.

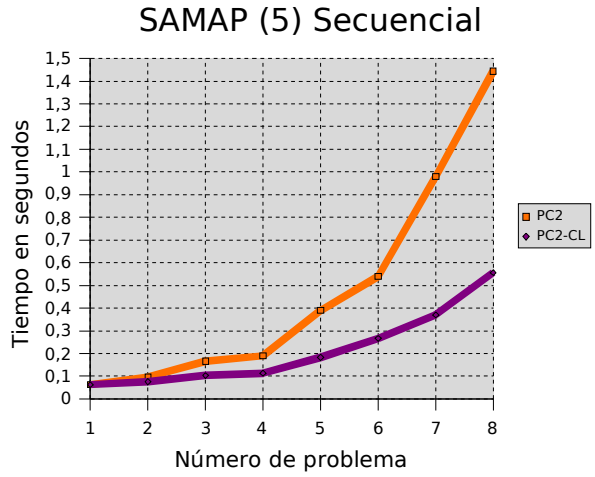
2 Utilizando scripts de Python embebidos en el propio dominio se ha implementado un algoritmo tipo A* capaz de calcular las rutas entre cualquier par de puntos.

3 Como usamos un modelo HTN, es posible especificar objetivos con distinto nivel de abstracción simplemente utilizando tareas más abstractas o más específicas.

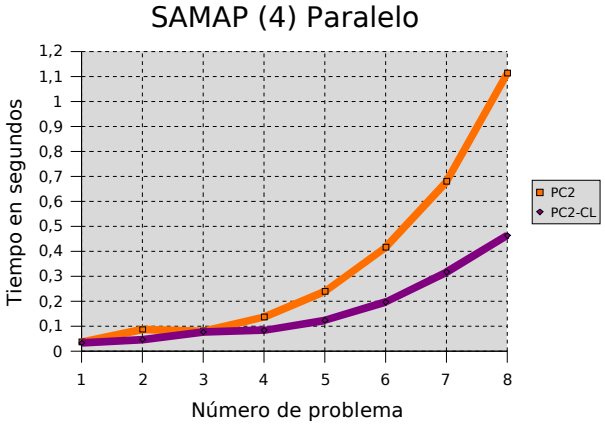
4 El modelo temporal del planificador, es capaz de gestionar landmarks, sincronizaciones entre tareas y gestión del timeline con la expresividad suficiente para modelar todas las restricciones temporales del dominio.

Se han escrito un par de versiones del dominio SAMAP para explorar los comportamientos de los algoritmos ante distintos problemas temporales. Los dominios consisten en general en un conjunto de turistas que hacen una serie de visitas programadas, se pueden incrementar tanto el número de turistas para los que obtener un plan como el número de visitas programadas. Hay una versión secuencial del algoritmo en el que las visitas de los distintos turistas son independientes y otra versión que llamamos paralela en la cual se planifica en paralelo para todos los turistas, pero estos tienen cada cierto tiempo que reunirse en un lugar en concreto antes de seguir con sus visitas. La versión paralela es por

lo tanto, desde el punto de vista temporal, más compleja ya que requiere de la sincronización entre las tareas que realizan los distintos turistas y por lo tanto es susceptible de provocar un mayor backtracking.



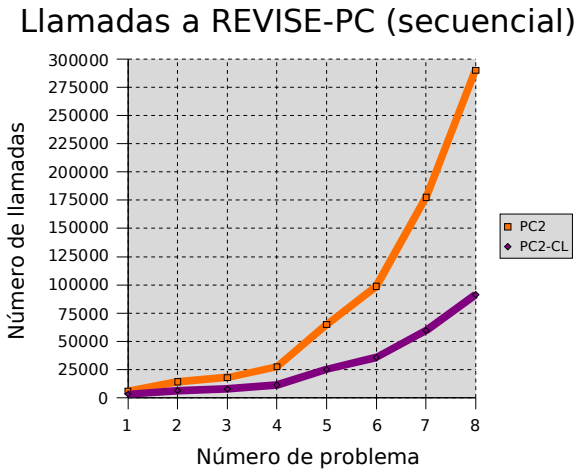
Gráfica 5: Comparativa en tiempo de los distintos algoritmos en el problema secuencial de SAMAP



Gráfica 6: Comparativa en tiempo de los distintos algoritmos en el problema paralelo de SAMAP

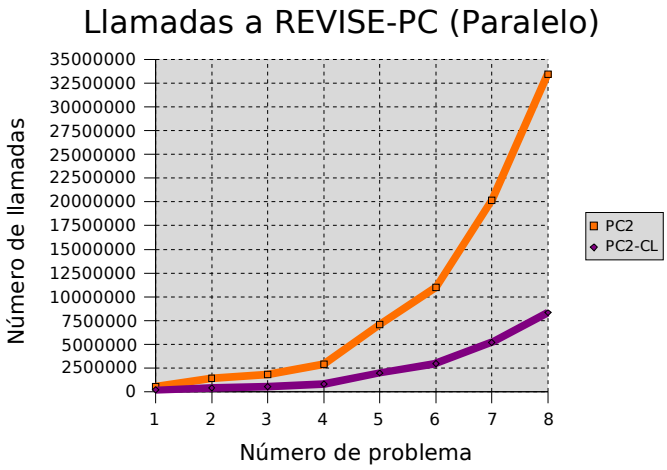
Para realizar la experimentación se ha usado un Pentium IV 2.0 Ghz con un S.O GNU/Linux. Las gráficas 5 y 6 muestran los resultados de los distintos algoritmos en los problemas secuencial y paralelo. Como era de esperar el caso paralelo cuesta más a todos los algoritmos que emplean más tiempo para resolverlo, de hecho la gráfica mostrada para el problema secuencial (Gráfica 5) calcula los planes para un turista más, cinco frente a los cuatro del caso paralelo, y aún así, los algoritmos son considerablemente más rápidos en el caso secuencial.

Se observa claramente, como es de esperar, que PC-2-CL es mucho más eficiente que PC-2, debido nuevamente a que realiza mucha menos inferencia temporal.



Gráfica 7: Llamadas a REVISE-PC en el caso secuencial de PC-2 y PC-2-CL

Las gráficas 7 y 8 muestran el número de llamadas a la función REVISE-PC de los algoritmos PC-2 y PC-2-CL que son las causantes de la diferencia en eficiencia entre ambos algoritmos. En todos los casos PC-2 hace muchas más llamadas a la función. También se aprecia claramente la diferencia en complejidad entre el problema secuencial y el paralelo (300.000 llamadas frente a 3.000.000 de llamadas).



Gráfica 8: Llamadas a REVISE-PC en el caso paralelo de PC-2 y PC-2-CL

5. - *Conclusiones y trabajo futuro*

Sólo por que algo no haya salido como lo habías planeado no significa que sea inútil.

***Thomas A. Edison**, inventor y hombre de negocios (1847-1931).*

Para terminar en este capítulo se presentará un resumen de lo expuesto en la memoria de tesis, resaltando los aspectos más destacados y las aportaciones más relevantes y se plantean algunas conclusiones al respecto. También se destacan las líneas de trabajo relacionadas con esta tesis a las que se dará prioridad durante los próximos meses y que darán continuidad al trabajo aquí expuesto.

5.1. - Resumen de las aportaciones de la tesis

Durante este trabajo se ha tratado de resaltar la dificultad que entraña la realización de una planificación automática para la resolución de problemas reales. En concreto hemos tomado tres aplicaciones distintas en diferentes proyectos de investigación y desarrollo donde la planificación automática es aplicable:

- **SIADEX (***NET033957***):** Sistema Inteligente de Ayuda a la Decisión en la Extinción de Incendios Forestales, que es un proyecto financiado por la Consejería de Medio Ambiente de la Junta de Andalucía, en donde se aplican técnicas de planificación automática para ayudar al director técnico de extinción a en la elaboración de planes de extinción.

- **ADAPTAPLAN (***TIN2005-08945-C06-02***):** Adaptación basada en aprendizaje, modelado y planificación para tareas complejas orientadas al usuario, que es un sistema orientado a desarrollar planes educativos en una serie de materias de forma telemática adaptados al perfil de usuario.

- **SAMAP (***TIC2002-04146-C05-02***):** Sistema adaptativo multiagente

para la planificación dependiente de contexto, que es un proyecto que trata de construir planes de visita personalizados a los lugares de interés en una ciudad.

Estas aplicaciones presentan una problemática común en una serie de aspectos que originan distintas líneas de investigación en el ámbito de la planificación automática:

- Integración de un planificación automático dentro de un sistema complejo en el que participan múltiples actores, ya sean humanos o otros sistemas software.
- Adaptación de las técnicas de planificación existentes a la resolución de problemas reales y a la forma de resolverlos por parte de los expertos humanos.
- Gestión del tiempo para la obtención de planes temporizados.

En el sentido de apoyar estas tres líneas de investigación se ha presentado el planificador HTNP y sus variantes temporales, así como una arquitectura completa de planificación construida a su alrededor.

En el capítulo II se presentó una arquitectura general de planificación que gira entorno a un planificador automático, en concreto el planificador HTNP. En esta arquitectura el planificador se presenta como un servicio web, que se invoca cuando es necesaria la obtención de un plan, y que se conecta con otros módulos del sistema desde los cuales el planificador extrae la información necesaria para obtener el plan, y a los cuales el planificador envía el plan resultante para su postprocesamiento, presentación al usuario y ejecución. Como se demuestra con la aplicación de esta arquitectura en aplicaciones concretas, es necesario romper el enfoque clásico de un planificador como una caja negra cerrada que apenas tiene interacción con el exterior.

Después en ese mismo capítulo se presentó el algoritmo HTNP como un planificador jerárquico orientado a la resolución de problemas de planificación en aplicaciones reales. Se ha demostrado que dicho planificador es muy eficiente en la resolución de problemas, lo que permite abordar la resolución de planes complejos con cientos de acciones y miles de objetos involucrados. Además, gracias a la expresividad del lenguaje HTN-PDDL en el que se basa dicho planificador, es posible escribir heurísticas y reglas de descomposición complejas, que facilitan el modelado de dominios reales, mejoran la eficiencia del planificador, y se acerca más a la forma de estructurar el conocimiento de planificación que utilizan los expertos humanos.

El capítulo III se ha centrado en establecer un modelo temporal para la

planificación jerárquica basada en redes STN y en la presentación de diversos algoritmos para la propagación de las restricciones temporales en dichas redes. La intención de este modelo presentado siempre fue que el modelo temporal tuviese la capacidad necesaria para expresar situaciones y restricciones temporales complejas. En este sentido primero se estableció el modelo basado en redes de restricciones temporales STN y como ajustar la planificación jerárquica a dicho modelo, mediante el uso de acciones durativas, literales temporizados, herencia de restricciones jerárquicas e imposición de restricciones temporales en las redes de tareas. Este modelo permite más expresividad temporal de la que ofrecen la mayoría de los planificadores, para aprovechar esta expresividad se modificó el lenguaje HTN-PDDL y se presentaron las herramientas expresivas de las que este lenguaje dispone para la descripción de dominios temporales. Por último se presentó el algoritmo HTNP2 un algoritmo basado en HTNP que implementa el modelo temporal presentado.

Sin embargo aunque el principal objetivo era disponer de un modelo temporal muy expresivo, en la práctica, es imprescindible que este modelo sea eficiente si se quiere que se pueda aplicar en aplicaciones reales como siempre ha sido la intención. Por eso en el capítulo IV se ha presentado el algoritmo HTNP2CL que utiliza un algoritmo de propagación de restricciones que se apoya en la estructura causal del plan para hacer la propagación de forma mucho más eficiente. En aplicaciones temporales esta mejora de la eficiencia es importantísima ya que en nuestro planificador la inferencia temporal es más lenta que el propio proceso de planificación. HTNP2CL es no solo en teoría sino en la práctica mucho más eficiente que HTNP2 como se ha demostrado de forma empírica.

El uso de diferentes fuentes de conocimiento para restringir la propagación temporal en una red como por ejemplo una STN con el fin de obtener una mejor eficiencia no es algo completamente novedoso, y en otros planificadores como por ejemplo en Ixtet [51] o en PASSAT [140] se hace algo parecido. En concreto en PASSAT se usa un enfoque totalmente ortogonal al utilizado en el modelo temporal presentado para HTNP, ya que restringe la propagación de restricciones dentro de la estructura jerárquica del plan. De hecho una posible línea de investigación futura podría ser adaptar este modelo de propagación de restricciones de PASSAT al modelo temporal explicado y comprobar si es posible aumentar aún más la eficiencia.

5.2. - Temas abiertos

Hay varias líneas de investigación ya abiertas, que debido principalmente a

su falta de madurez no han sido incluidas en este trabajo de tesis y en donde se centrarán los esfuerzos investigadores en los próximos meses.

1 Incrementar la expresividad del lenguaje HTN-PDDL e introducir capacidades de iniciativa mixta dentro del planificador.

1.1 El lenguaje HTN-PDDL es bastante expresivo pero se está estudiando la ampliación de la funcionalidad de los metatags en próximas extensiones del lenguaje, de forma que incluya preferencias de usuario y prioridades, así como la posibilidad de definir tags calculados [44] [43] [93]. La idea es que el plan no contenga únicamente la información necesaria para obtener el plan sino también información interesante para extraer información relevante en un postprocesamiento. Se está trabajando también en un lenguaje de consulta que permita realizar consultas contra las estructuras del plan, y que dicho lenguaje pueda ser utilizado en el depurador integrado, en una fase de postproceso del plan o incluso durante la propia fase de planificación para definir heurísticas.

1.2 Actualmente HTNP escoge los métodos para expandir de acuerdo con el orden en el que están descritos en el dominio y las precondiciones limitan los métodos que son aplicables. Sin embargo es interesante que los métodos se puedan escoger en uno o en otro orden dependiendo del problema que se desee resolver, de esta forma se pueden obtener planes distintos dependiendo de las métricas que utilicemos para escoger los métodos.

2 Otra línea de trabajo que se está abriendo ahora es mejorar la integración del planificador dentro de otros sistemas más complejos. Se ha explicado la arquitectura general de planificador y su diseño en forma de servicio web. Aún así es necesario "abrir" mucho más el planificador y permitir una mejor interacción con el mismo. En concreto es muy interesante que el planificador sea capaz de leer dominios y problemas y escribir planes en otros lenguajes distintos de PDDL y más extendidos, como por ejemplo BPMN[19] (*Bussines Process Management Notation*), BPEL (*Bussines Process Execution Language*) o UML (*Unified Modeling Language*).

3 Por último se necesita describir un modelo genérico de replanificación en HTNP que además tenga en cuenta el modelo temporal. Esta es una línea de investigación en la que se lleva trabajando mucho tiempo [90] sin embargo, a pesar de disponer ya de varios modelos teóricos de cómo realizar la replanificación, todavía continua abierta y se

19 http://www.oasis-open.org

necesita implementar y probar los distintos modelos.

5.3. - Publicaciones relacionadas con la tesis

Los trabajos presentados en esta tesis han dado origen a múltiples publicaciones algunas de las cuales aparecen en la bibliografía final, el objetivo de este apartado es destacar aquellas que son más importantes:

L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *Temporal enhancements of an HTN planner*, 2005, Spanish Conference on Artificial Intelligence, CAEPIA 2005 (To appear in Lecture Notes on Artificial Intelligence).

L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *Efficiently handling temporal knowledge in an HTN planner*, 2006, 16th International Conference on Automated Planning and Scheduling (ICAPS 2006).

L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao., *Bringing users and planning technology together. Experiences in SIADEX*, 2006, 16th International Conference on Automated Planning and Scheduling (ICAPS 2006).

Premiado como el mejor artículo de planificación aplicada durante la 16 ICAPS.

M. de la Asunción and L. Castillo and J. Fdez-Olivares and O. García-Pérez and A. González and F. Palao, *SIADEX: an interactive artificial intelligence planner for decision support and training in forest fire fighting*, 2005, AIComm special issue on Binding Environmental Sciences and artificial intelligence.

L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao. SIADEX. *Un entorno integral de planificación para el diseño de planes de actuación en situaciones de crisis*. VI Jornadas de Transferencia Tecnológica de Inteligencia Artificial, TTIA'2005 (AEPIA).

Premiado como la mejor aplicación de transferencia de tecnológica de inteligencia artificial durante la primera conferencia española en ciencias de la computación (CEDI 2005).

5.4. - Otras publicaciones

Hay un interés creciente en acercar las investigaciones realizadas en la comunidad universitaria al mundo empresarial y al público en general. En este sentido esta sección quiere hacerse eco del gran impacto que a tenido el proyecto SIADEX en los medios de comunicación generalistas:

Sistema Inteligente de Ayuda a la Decisión para el Diseño de Planes de Extinción. SIADEX. "Incendios Forestales: Revista Independiente de los Profesionales de la Extinción de Incendios Forestales". 12 Abril 2005, 17-25.

Perfeccionan en Granada tecnología del pentágono para el INFOCA. "Granada Hoy". 13 Junio 2005, 1,3.

Inteligencia Artificial contra los incendios forestales. "El PAÍS, edición impresa. ANDALUCÍA. Sección hecho aquí". 13 Junio 2005.

Un sistema inteligente para la extinción de incendios. "CAMPUS" 239- 2005, pg 29.

Inteligencia Forestal. "Andalucía Investiga". 19 Febrero 2005. 22-23.

Investigadores de la UGR desarrollan un sistema para ayudar en la extinción de incendios forestales. "Andalucía Investiga". 21 Enero 2005.

Investigadores de la UGR desarrollan un sistema para ayudar a la extinción de incendios forestales. "Periódico LUN@TICOS. Sección El rincón de la ciencia". 10 Febrero 2005, pg 10.

Científicos crean un sistema de ayuda para extinguir fuegos. "IDEAL de GRANADA". pg 12.

Científicos de Granada desarrollan un sistema para ayudar a extinguir fuegos. "Zona Verde. El periódico para una Andalucía sostenible.". 1 Febrero 2005.

El fuego, acorralado por la Inteligencia Artificial. "Gaceta Universitaria". 7 Febrero 2005. pg 17.

Proyecto SIADEX: Trabajan por desarrollar un sistema para ayudar en la extinción de incendios forestales. "Medan 21". 26 Enero 2005.

Investigadors de la UGR desenvolupen un sistema per ajudar en l'extinción d'incendis forestals. "Notices de la Oficina de Transferencia Tecnológica. Universitat de Girona". 28 Enero 2005.

Nuevas tecnologías e incendios forestales (II)."Medio Ambiente. Consejería de Medio Ambiente, Junta de Andalucía". 44. Otoño 2003.

Diseñado un programa que ayuda a extinguir incendios. "Tribuna complutense". 15 Febrero 2005.

Investigadores de la UGR desarrollan un sistema inteligente para ayudar en la extinción de incendios forestales. "C@mpus Digital-UGR". 1 Febrero 2005.

Científicos de Granada desarrollan un sistema para ayudar a extinguir fuegos. "Confederación Regional de Organizaciones Empresariales de Murcia". 4 Marzo 2005.

Investigadores de la UGR desarrollan un sistema para ayudar en la extinción de incendios forestales. "Europa I+D". 24 Enero 2005.

Apagar fuego. "UNIVERSIA". 13/07/2005.

La noticia sobre el proyecto no solamente ha tenido impacto en prensa, sino también en radio y televisión. Que los investigadores sepan, se ha hablado de SIADEX en la Cadena SER, en Canal Sur Radio y en Radio Nacional de España. También a aparecido en los programas de televisión, Andalucía Directo (Canal Sur Televisión Julio 2005), Canal Sur Noticias (Canal Sur Televisión Junio 2005) y se hizo un reportaje acerca de SIADEX en el programa Espacio Protegido (Canal 2 Andalucía Julio 2005).

I - *Nomenclatura*

- Γ: Red de tareas.
- Γ_0: Red de tareas objetivo para un planificador HTN.
- $\mathcal{D}$: Dominio de planificación.
- $\mathcal{P}$: Problema de planificación.
- ε_i: Estado i-ésimo de un planifiicador que busca en el espacio de estados.
- ε_0: El estado inicial del que se parte al planificar.
- α: La estructura que mantiene la agenda del planificador.
- δ: Es el conjunto de acciones descritas en el dominio.
- φ: Es un conjunto de acciones.
- ρ: Tarea genérica (operador primitivo o tarea abstracta).
- σ: Operador primitivo.
- τ: Tarea abstracta.
- θ: Orden entre las tareas de un plan.
- λ: Estructura de vínculos causales.
- ζ: Representa una precondición de un operador primitivo.
- π: Representa un plan como una secuencia de operadores primitivos.
- Π: Representa un plan como una estructura jerárquica con una composición de tareas abstractas y operadores primitivos.
- μ: Es el conjunto de métodos de una tarea abstracta.

- *l*: Es el símbolo que utilizaremos para denotar un literal.

II - *ANEXO: El lenguaje de descripción de dominios HTN-PDDL*

1. Introducción

En este documento se describe la extensión del lenguaje PDDL para la descripción de dominios jerárquicos que se denomina HTN-PDDL. HTN-PDDL es un lenguaje con soporte temporal que permite controlar y dirigir la búsqueda mediante el uso de heurísticas.

El lenguaje HTN-PDDL está basado en el estándar *de facto* PDDL [81] [128] [127] [4] (*Planing Domain Description Language)* utilizado en la *International Planning Competition (IPC),* que es una competición que se realiza de forma bianual y de forma conjunta con la Conferencia Internacional de Planificación y Scheduling *(ICAPS International Conference on Automated Planning and Scheduling)* y que trata de comparar los planificadores más modernos enfrentándolos a una batería de problemas en diversos dominios descritos en PDDL.

La versión del lenguaje a partir de la cual se extiende HTN-PDDL es la 2.2 en su nivel 3, versiones posteriores no están todavía soportadas en el lenguaje.

Las principales características del lenguaje PDDL son las siguientes:

- Las acciones o operadores primitivos tienen el estilo usado por el planificador STRIPS [37] [39].
- Se pueden definir efectos condicionales.
- Dispone de cuantificadores universales.
- Permite la descripción de axiomas sencillos.
- Se pueden describir restricciones de integridad.

- En la primera versión del lenguaje PDDL, la 1.2, existía la posibilidad de definir acciones jerárquicas. El formalismo introducido era muy complejo y eso hizo que ninguno de los planificadores más conocidos hiciera uso de esta característica. Esta capacidad del lenguaje además no fue ampliada ni modificada en versiones posteriores por lo que ha quedado abandonada.

- Se permiten utilizar distintas extensiones del lenguaje, que los planificadores pueden o no utilizar dependiendo de la potencia expresiva de estos.

El lenguaje HTN-PDDL además aporta las siguientes características:

- Aumenta la expresividad temporal de PDDL 2.2 nivel 3.

- Añade un soporte para las acciones jerárquicas, con una expresividad distinta y más clara que la usada en PDDL 2.1.

- Añade la posibilidad de invocar a lenguajes de scripting, como Python, para hacer llamadas externas o realizar ciertos cálculos, durante la fase de planificación.

Aunque HTN-PDDL es un superconjunto de PDDL se basa en un formalismo distinto al utilizado por los planificadores "planos" que son los que compiten en la IPC. Por lo tanto probablemente ningún planificador jerárquico pueda resolver los dominios y los problemas usados en la IPC sin introducir ciertos cambios sobre los mismos, a pesar de que un parser de HTN-PDDL pueda parsear los dominios escritos en PDDL. De hecho los objetivos en HTN-PDDL se describen de una manera distinta que en PDDL.

El lenguaje HTN-PDDL está en constante desarrollo y evolución, por lo tanto el lenguaje descrito en este documento puede ser ampliado en futuras versiones, aunque siempre se tratará de mantener la compatibilidad hacia atrás. Esta documento específica las características del lenguaje HTN-PDDL en su **versión 1.1**.

2. Notación

La notación que se utilizará para describir las reglas será EBNF (*Extended BNF).* Se utilizarán los siguientes formalismos para describir las reglas sintácticas del lenguaje HTN-PDDL

- Las reglas tienen la siguiente forma <elemento>: <expansion>

- Se usarán ángulos para delimitar los nombres de los elementos

sintácticos.

- Se usan los corchetes [ ] para delimitar aquellos elementos que son opcionales.
- Se usan paréntesis para delimitar bloques dentro del lenguaje HTN-PDDL o para indicar parámetros de una regla en concreto. Los paréntesis forman parte del lenguaje.
- Como en BNF el asterisco * denota 0 o más veces y el símbolo + denota 1 o más veces.

3. Comentarios

Los comentarios en HTN-PDDL empiezan con un ";" y terminan con un retorno de carro.

Ejemplo 1.
```
; Esto es un comentario escrito en HTN-PDDL
```

4. Términos y átomos

Los términos pueden ser constantes (objetos del dominio) o variables.

```
<symbol>:               <name>

<term>:                 <constant>
                        | <variable>
                        | <number>

<variable>:             ?<name>

<constant>:             <symbol>

<predicate>:            (<symbol> <term>*)

<typed-predicate>:      (<symbol> <typed-variable>*)

<typed-variable>:       <variable> - <type>
<typed-variable>:       <variable>

<type>:                 <symbol>
                        | (either <type>*)
```

Un símbolo es una cadena alfanumérica que comienza con una letra y que puede contener letras, dígitos, guiones y subrayados. Los caracteres acentuados y

la ñ del castellano también es aceptada. HTN-PDDL no es sensible a las mayúsculas, es decir, el símbolo "hola" es equivalente al símbolo "HoLa".

Una constante se identifica con un símbolo y representa un objeto del dominio. Una variable se identifica con un símbolo que comienza por un carácter de cierre de interrogación.

Al igual que en el PDDL estándar tanto símbolos, como variables, como predicados, pueden tener un tipo asociado. Se permite definir una jerarquía de tipos que efectúa las relaciones de herencia de propiedades entre los objetos del dominio.

```
Ejemplo 2.
; objetos
ferrari - automovil
; predicados (atributos de un objeto)
; instanciados y no instanciados
(velocidad_maxima ?vehiculo - automovil ?x - number)
(velocidad_maxima ferrari 220)
```

En HTN-PDDL si no se especifica expresamente un tipo se supone que hereda por defecto del tipo especial "object".

5. Dominios

Para que un planificador disponga de toda la información para operar con el lenguaje HTN-PDDL éste recibe dos ficheros de entrada. Un fichero con la especificación del dominio, que es la descripción del mundo (tareas jerárquicas, operadores primitivos, objetos, atributos y tipos) y otro fichero con el problema que queremos resolver.

La descripción de un dominio en HTN-PDDL tiene la siguiente sintaxis:

```
<domain>:         (define (domain <symbol>)
                  [<require-def>]
                  [<customization-def>]
                  [<types-def>]
                  [<constants-def>]
                  [<predicates-def>]
                  [<functions-def>]
                  [<structure-def>*])
```

En HTN-PDDL Un dominio consta de siete bloques todas excepto el bloque de personalización "customization" ya existen en PDDL 2.2. La sección de

personalización será analizada en detalle más adelante. Se empezará estudiando la sintaxis de la declaración de tipos.

6. Declaración de tipos

```
<types_def>:         (:types <type-def>*)
<type-def>:          <symbol>+ - <type>
<type-def>:          <symbol>+
```

Se usa una lista de tipos para declarar todos los tipos que heredan (son subtipo) de un tipo determinado. Los tipos aparecen precedidos de un guión (-), todo tipo que preceda al guión (subtipo) queda declarado como del tipo que aparece tras el guión (supertipo). Si no se especifica un supertipo explícitamente un tipo siempre se supone por defecto que hereda del tipo especial "object".

Se permite realizar herencia múltiple mediante el uso de clausula either.

Ejemplo 3.

```
; declaración de tipos
(:types
    persona
    persona_anorexica – persona
    persona_guapa – persona
    ; herencia múltiple
    modelo – (either persona_anorexica persona_guapa)
)
```

7. Declaración de constantes

```
<constants-def>:     (:constants <constant-def>*)
<constant-def>:      <symbol>+ - <type>
<constang-def>:      <symbol>+
```

Las constantes se representan mediante símbolos y modelan los objetos del dominio. Se permite que las constantes sean declaradas como pertenecientes a una serie de tipos determinados, como se puede apreciar en el siguiente ejemplo:

Ejemplo 4.

```
; declaración de constantes
(:types
    paz_vega – actor
    ana_belen – (either actor cantante)
)
```

8. Declaración de predicados

```
<predicates-def>:    (:predicates <typed-predicate>*)
```

Los predicados son usados para definir atributos de los objetos. También se usan para definir relaciones de cualquier aridad entre los objetos. En planificación jerárquica HTN también se pueden usar ciertos predicados que no forman parte del estado como herramientas para generar heurísticas de control del procesamiento del planificador.

```
Ejemplo 5.
; declaración de predicados
(:predicates
    (altura ?p – persona ?x -number)
    (casado ?amargado_x -persona ?amargado_y - persona)
    (jefe_de ?x -persona ?mala – persona)
    (director ?la_peor - persona)
)
```

9. Declaración de funciones

```
<functions-def>:     (:functions <function-def>*)
<function-def>:      (<typed-predicate>*) [- <type>]
                     [<python-code>]
<python-code>:       {<python_script>}
```

Las funciones utilizadas en PDDL no se corresponden totalmente con el concepto que tenemos de función, por eso en PDDL se denominan "*fluents literals*". Un "*fluent*" es un predicado que almacena un determinado valor, que en la mayoría de los casos suele ser un número, de forma que el lenguaje permite realizar operaciones aritméticas y de comparación entre estos *fluents* como veremos más adelante.

En HTN-PDDL se permite asociar un literal a una verdadera función descrita en un lenguaje de scripting. El lenguaje de script más usado en los dominios descritos en HTN-PDDL es Python (http://www.python.org).

Las convenciones a usar para el paso y retorno de parámetros al script están

todavía en una fase experimental.

Actualmente sólo se permite que el script devuelva (mediante la cláusula return) un número o un símbolo (objeto constante) ya predefinido en el dominio.

El script debe poder usar las variables PDDL declaradas como argumento de la función, en los cálculos internos. El parser de HTN-PDDL debe hacer una sustitución de estas variables por los valores que efectivamente toman en tiempo de planificación.

De esta forma si se consigue tener funciones reales, que realizan cálculos o llamadas externas en pleno proceso de planificación.

```
Ejemplo 6.
; declaración de funciones y llamadas a Python
(: functions
; un predicado python
(distance ?x1 ?y1 ?x2 ?y2)
{
 import math
 return math.sqrt ( (?x2 - ?x1) * (?x2 -?x1) + (?y2 - ?y1) * (?y2 - ?y1))
}
; un predicado pddl
(distance ?x ?y -location) - number
)
```

10. Sección principal

La sección principal recoge la definición de operadores primitivos, tareas compuestas y axiomas, que el planificador puede utilizar para resolver el problema de planificación.

```
<structure-def>:  <action-def>
                  | <durative-action-def>
                  | <derived-def>
                  | <htn-task-def>

<derived-def>:    (:derived <derived-body>)
<derived-def>:    (:derived <atomic-formula(variable)>
                  <goal-def>)
                  | (:derived <atomic-formula(variable)
                  {<python-code>}
```

Las acciones y las tareas se estudiarán con mas detalle en las siguientes secciones.

Los axiomas sirven para definir nuevos predicados a partir del uso de predicados ya existentes en una cláusula más compleja.

```
Ejemplo 7.
; declaración de axiomas
(:derived (ganga ?x – producto)
    (and (bueno ?x) (bonito ?x) (barato ?x))
)
```

11. Operadores primitivos

Los operadores primitivos (acciones) en HTN-PDDL son muy similares a los de PDDL, con algunas pequeñas ampliaciones. De hecho se pueden usar acciones descritas en PDDL directamente en HTN-PDDL.

```
<action-def>:           (:action <name>
                        :parameters (<typed-variable>*)
                        [<metags>]
                        [<preconditions-def>]
                        [<effect-def>])

<preconditions-def>:    :precondition <goal-def>

<effect-def>:           :effect <effect>

<metatags>:             :meta (<tag>*)

<tag>:                  (:tag <name> <text>)

(*) Nota: <text> es una cadena de caracteres delimitada por comillas
dobles (“”).
```

La principal diferencia la constituye la posibilidad de usar metatags, cuyo uso se explicará en las siguientes secciones.

11.1. *Parámetros*

Los parámetros son una lista de variables que el planificador instancia durante la fase de planificación, y que son los argumentos utilizados dentro del cuerpo del operador.

11.2. *Metatags*

Los *metatags* son una extensión del lenguaje que actualmente está en fase experimental, por lo que puede cambiar en futuras versiones. El concepto subyacente a los metatags es poder incluir información extra en el dominio, que aunque no sea directamente utilizable por el planificador, pueda ser utilizada por

otros módulos, o en un análisis posterior del plan resultante.

Cuando el planificador termina puede incluir esta información extra en plan resultante. En concreto, si se supone que el planificador obtiene un plan como un documento XML, los metatags podrían ser incluidos como tags XML, que podrían ser parseados y analizados con posterioridad.

```
Ejemplo 8.
; Ejemplo de acción durativa (temporizada) con
; metatags
(:durative-action move
 :parameters (?obj - object ?loc -location)
 :meta(
   (:tag prettyprint "?start ?end > mover ?obj desde ?lx hasta ?loc.
Llegada a las ?end (?duration min)")
   (:tag cost "low")
 )
 :duration (= ?duration 22)
 :condition((and
    (location ?obj ?lx)
    (not (same ?lx ?loc))
 ))
 :effect((and
    (not (location ?obj ?lx))
    (location ?obj ?loc)
 ))
)
```

La salida XML que se obtendría podría ser similar a la siguiente:

```
<primitive name="move" id="170" indx="186" start="27/01/2006 08:11:00"
end="27/01/2006 08:33:00" duration="22" start_point="64" end_point="65">
         <meta name="prettyprint">
         27/01/2006 08:11:00 &gt; mover BRI112 desde BASE01 hasta
WAITAREA. Llegada a las 27/01/2006 08:33:00 (22.000000 min)
         </meta>
         <meta name="cost">
         low
         </meta>
         <parameters>
            <parameter pos="0" name="g">
               <constant name="BRI112" id="715">
                  <type name="Brigade" id="73">
                  </type>
               </constant>
            </parameter>
.....
```

Existe un *metatag* que tiene un carácter especial que se llama *prettyprint.*

P*rettyprint* es una cadena de texto parseable, pensada para presentar las acciones resultantes del proceso de planificación, de una forma más legible al usuario final.

prettyprint, o en general cualquier metatag, debe permitir incluir dentro de la cadena de texto variables, que son sustituidas por el valor al que estas variables son ligadas. Las variables deben incluir al menos los parámetros de la acción, aunque un planificador en concreto. puede incluir variables extras a costa de la pérdida de la generalidad del dominio.

Cuando un planificador encuentra una variable que es capaz de interpretar, por defecto no debe hacer la sustitución.

Cuando termina el proceso de planificación y las acciones resultantes son presentadas al usuario (ya sea a través de la salida estándar o a través de un fichero XML o de texto), el planificador debe usar el formato especificado en *prettyprint.*

Otra alternativa también permitida es que en lugar de usar los identificadores de las variables para hacer la sustitución, se utilice el número de argumento dentro de la lista de argumentos, precedido por el carácter "$" ($1, $2, ...), al igual que en el resultado del matching de una expresión regular en Perl.

11.3. *Precondiciones*

Las precondiciones son una lista opcional de objetivos que deben ser satisfechos en el estado del mundo justo antes de la ejecución de la acción. La descripción de objetivos es bastante expresiva, ya que se permiten expresiones en lógica de primer orden. Sino se especifican precondiciones se supone que la acción puede ejecutarse en cualquier estado del universo.

La sintaxis BNF para una condición es la siguiente:

```
<goal-def>:          ( )
                     | (! <goal-def>)
                     | (:sortby <variable-ord>+  <goal-def>)
                     | (:bind <variable> <fluent-exp>)
                     | (:print <sgoal-def>)
                     | (:print <text>)
                     | (:print <term>*)
                     | <sgoal-def>
                     | <timed-goal>


<sgoal-def>:         (and <goal-def>*)
                     | (or  <goal-def>*)
                     | (not <goal-def>)
                     | (imply <goal-def> <goal-def>)
                     | (exists  (<typed-variable>* ) <goal-def>)
```

```
                    | (forall  (<typed-variable>* ) <goal-def>)
                    | <fluent-comp>
                    | <atomic-formula(term)>

<variable-ord>:     <variable> :asc
                    | <variable> :desc

<fluent-comp>:      (<binary-comp> <fluent-exp> <fluent-exp> )

<binary-comp>:      >
                    | <
                    | =
                    | >=
                    | <=
                    | !=
```

Se han añadido algunos predicados más no incluidos en PDDL estándar como son los siguientes:

- bind: Permite asignar a una variable sin instanciar el contenido de un predicado numérico o un *fluent.*

- print: Es un predicado que siempre es cierto y permite imprimir un texto libre, el contenido de una variable, o el resultado de analizar la unificación de una condición. Este predicado es útil a la hora de imprimir mensajes por pantalla durante el proceso de planificación y facilitar así la tarea de depuración.

- Cuando se produce una unificación el planificador puede escoger cualquiera de las posibles sustituciones de variables por valores constantes. A veces es interesante que el diseñador de dominios tenga el control sobre el orden en el que se quieren ir probando las unificaciones. Éste es precisamente el objetivo del predicado *sortby.*

Ejemplo 9.

```
; Un ejemplo de uso de precondiciones con sortby y bind
(:action recoger_pasajero
 :parameters (?p – person ?y - point)
 :precondition (and
                 (esperando_taxi ?p)
                 (sortby ?d :asc (and
                   (libre ?t -taxi)
                   (posicion ?t ?x - point)
                   (bind ?d (calcula_distancia ?x ?y))
                 ))
                )
 :effect (and (not (esperando_taxi ?p)) (not (libre ?t)))
)
```

Los objetivos en HTN-PDDL, no se describen en base a una condición que es necesario hacer cierta en el estado final, como ocurre en PDDL, sino como una red de tareas a descomponer. Por eso se tratarán más adelante, cuando se presenten las redes de tareas.

11.4. *Efectos*

Los efectos son una descripción completa de los cambios que la acción produce en el estado actual del universo. Los efectos pueden ser cuantificados universalmente y también se permiten efectos condicionales (dependientes del estado actual del universo), pero no todas las sentencias de la lógica de primer orden están permitidos (por ejemplo funciones de Skolem o expresiones disyuntivas). Es importante destacar que, en este sentido, el lenguaje es asimétrico, es decir, las precondiciones de las acciones son considerablemente más expresivas que sus efectos.

```
<effect>:          ( )
                   | (and <c-effect>*)
                   | <c-effect>

<c-effect>:        (forall (<typed-variable>+) <effect>)
                   | (when <goal-def> <cond-effect> )
                   | <p-effect>
                   | <timed-effect>


<p-effect>:        (<assign-op> <atomic-formula(term)>
                   <fluent-exp>)
                   | (not <af-effect>)
                   | <af-effect>

<af-effect>:       <atomic-formula(term)>
                   | (:maintain <atomic-formula(term)>)


<cond-effect>:     (and <p-effect>*)
                   | <p-effect>

<assign-op>:       assign
                   | scale-up
                   | scale-down
                   | increase
                   | decrease


<fluent-exp>:      (<binary-op> <fluent-exp> <fluent-exp>)
```

```
                | (<unary-op> <fluent-exp>)
                | <atomic-formula(term)>
                | <term>


unary_op:       -
                | :abs
                | :sqrt

binary_op:      +
                | -
                | *
                | /
                | :pow
```

A la hora de que el planificador evalúe un efecto se supone que todas las variables deben estar ligadas, bien por qué lleguen desde un parámetro, se hayan ligado al hacer una unificación en las precondiciones, o bien por que estén cuantificadas.

También se asume que como en STRIPS el valor de verdad de un predicado es persistente a lo largo del tiempo. En STRIPS por eso se define una lista de supresión para hacer falsos los predicados (retirarlos del estado, ya que cualquier predicado que no se encuentre en el estado se supone falso). En HTN-PDDL sin embargo no existe explícitamente una lista de supresión, en lugar de ello cuando se quiere hacer falso un predicado, simplemente se niega.

Ejemplo 10.
```
; hacer falso el valor de un predicado
(:action drop_item
 :parameters (?arm – robotic_arm ?obj - item)
 :precondition (holding ?arm ?obj)
 :effect ((not (holding ?arm ?obj))
))
```

Cómo es lógico ningún efecto de un operador primitivo puede alterar el valor de verdad de un predicado si no es mencionado explícitamente. Es decir si el efecto de un operador no cambia el valor de el predicado *P* el predicado *P* debe mantenerse inalterado tras la ejecución del operador.

Ejemplo 11.
```
; Algunos ejemplos autoexplicativos de efectos
; condicionales y cuantificadores universales.
(:action unload_cargo
 :parameters (?t – truck ?l - location)
 :precondition (at ?t ?l)
```

```
 :effect (forall ?x – obj
      (when (in ?obj ?t)
         (and (not (in ?obj ?t)) (in ?obj ?l))
      )
))
```

Observar que los objetivos en un efecto condicional tienen la misma sintaxis que en las precondiciones pero su función es distinta. Si un objetivo no se puede cumplir en las precondiciones el operador no puede ejecutarse, sin embargo si un objetivo no se cumple en un efecto condicional, la acción si puede ejecutarse, pero el efecto condicional no producirá cambios en el estado actual.

La clausula para proteger un predicado *maintain* pertenece a HTN-PDDL y no está soportada en el PDDL estándar. Esta cláusula sirve para evitar que otros operadores primitivos puedan alterar el predicado protegido. Si una acción trata de modificar el valor de verdad de un predicado mantenido, la aplicación de la acción fallará y el planificador hará backtracking. Para eliminar la protección simplemente hay que negarla.

```
Ejemplo 12.
; Ejemplo de cláusula maintain
(:action proteger
 :parameters (?x)
 :precondition ()
 :effect (:maintain (maintained ?x)))

(:action desproteger
 :parameters (?x)
 :precondition ()
 :effect (not (:maintain (maintained ?x))))
```

Los fluents de HTN-PDDL son iguales a los de PDDL salvo la adición de algunos operadores.

```
Ejemplo 13.
; Un ejemplo con fluents
(:action revender
 :parameters (?x -producto)
 :precondition (> (en_stock ?x) 0)
 :effect (assign (pvp ?x) (* (precio_compra ?x) 2))
)
```

12. Tareas abstractas

Una de las partes más interesantes de HTN-PDDL es la forma en como se introduce el soporte jerárquico como una capa más sobre PDDL. PDDL 1.2 tiene una sintaxis compleja para definir redes de tareas (quizá por su excesiva dependencia de la sintaxis de LISP), lo que hace que el dominio resultante sea poco legible. Uno de los objetivos principales de HTN-PDDL es que sea estructurado y fácilmente legible, sobretodo teniendo en cuenta el soporte temporal que añade un grado más de dificultad.

La notación BNF para definir redes de tareas y tareas abstractas es la siguiente:

```
<htn-task-def>:      (:task <name>
                      :parameters (<typed-variable>*)
                      [<meta-tags>]
                      <method-def>
                     )

<method-def>:        <method>*
                     | (!<method>*)

<method>:            (:method <name>
                      [<meta-tags>]
                      [<preconditions-def>]
                      :tasks <task-network>
                     )

<task-network>:      <task-structure>
                     | ( )

<task-structure>:       <task-def>
                     | ! task_def
                     | \<<dur-constraints> <task-structure>+\>
                     | \<<task-structure>+\>
                     | (<dur-constraints> <task-structure>+)
                     | (<task-structure>+)
                     | [<dur-constraints> <task-structure>]
                     | [<task-structure>]


<task-def>:          <atomic-formula(variable)>
                     | (:achieve <goal-def>)
                     | <inline-def>

<inline-def>:        (:inline [<goal-def>] [<effect>])
                     | (:!inline [<goal-def>] [<effect>])

<atomic-formula(x)>:       (<name> x*)
```

Una red de tareas en HTN-PDDL, tiene un nombre unos parámetros y una lista de métodos de descomposición asociados. Cada método lleva asociado un nombre una precondición que debe hacerse cierta para su ejecución y una red de tareas para expandir. Las redes de tareas en HTN-PDDL son de tres tipos:

- Secuenciales: Este tipo de red de tareas se representa colocando las tareas hijas a expandir entre paréntesis. Obliga a que las tareas y sus descendientes mantengan el orden.

- Paralelas o independientes: Se representan colocándolas entre corchetes. El planificador podrá escoger cualquier orden entre las tareas y entre sus subtareas, pero se considerará que si una tarea de la red falla, la red de tareas enteras es fallida.

- Permutables o combinatorias: Se representan entre ángulos. El planificador deberá comprobar todos los órdenes entre las tareas incluyendo sus subtareas.

Esta sintaxis a la hora de definir redes de tareas hace que el código HTN-PDDL sea sencillo y fácilmente legible:

Ejemplo 14.

```
; Definición de una tarea abstracta
(:task moverse
:parameters (?quien ?destino)
  (!
    (:method en-destino
     :precondition (= (posicion ?quien) ?destino)
     :tasks ()
    )
    (:method andando
     :precondition (and (bound ?donde (posicion ?quien)) (<= (distancia ?
donde ?destino) (limite_andando ?quien)))
     :tasks (ir_caminando ?quien ?destino)
    )
    (:method en_taxi
     :precondition ()
     :tasks (ir_en_taxi ?quien ?destino)
    )
  )
 )

(:task ir_en_taxi
  :parameters (?quien ?destino)
  (:method ir_en_taxi
   :precondition (= ?origen (posicion ?quien))
   :tasks (
       (esperar_taxi_en ?quien ?origen)
```

```
      (coger_taxi_a ?quien ?destino))
 )
)
```

13. Mecanismos para el control de la búsqueda

Uno de los aspectos más interesantes de la planificación HTN es la posibilidad de intervenir en el proceso de búsqueda mediante el uso de heurísticas. HTN-PDDL introduce información de control mediante el uso de las tareas *inline* y el concepto de corte.

Las tareas *inline* son, a efectos del proceso de planificación, acciones primitivas que no tienen nombre. Tienen una precondición que debe ser cierta para su ejecución y un efecto asociado que altera el estado. Sin embargo, la semántica de las acciones *inline* es distinta para el escritor de dominios que la de las acciones normales. Este tipo de acciones se utilizan para incluir información de control de la búsqueda en el estado del planificador (que es distinto de la modelización dentro del estado del planificador del estado del universo) incrustada en la descripción de la red de tareas.

Distinguimos dos tipos de acciones inline:

- Las "*:inline*": Estas acciones si sus precondiciones fallan, fallarán como una acción normal y producirán backtracking.
- Las "*:!inline*": Estas acciones no producen fallo, ni provocan backtracking aunque sus precondiciones sean falsas. Si la precondición es cierta la acción *inline* aplicará los efectos, en otro caso no hará nada.

Las acciones *inline* no deben incluirse en el plan resultante puesto que no tienen la semántica de una acción primitiva real. Los literales con los que operen las acciones *inline* no deben alterar el estado del mundo, puesto que estas acciones no están diseñadas para finalmente ejecutarse. Solamente podrán alterar los literales internos al proceso de planificación que tienen información de control.

Ejemplo 15.

```
; Uso de acciones inline

;; Esta tarea evita hacer múltiples llamadas a la función
;; calcula_distancia insertando en el estado un literal
;; con la distancia cacheada. Acelera la planificación en
;; dominios donde este cálculo es muy habitual

(:task Cachear_Distancias
```

```
   :parameters (?punto)
   (:method unico
    :precondition ()
    :tasks (
    (:inline () (forall (?x - Posicionable)
             (when (and
                              (disponible ?x)
                              (posicion ?x ?pos)
                              (assign ?d (calcula_distancia ?punto ?pos)
                            )
               (assign (distancia_cacheada ?x ?punto) ?d)
    )))
    )
)
```

El corte (!) es otro de los conceptos introducidos para mejorar la eficiencia del proceso de planificación. Este concepto está tomado del lenguaje PROLOG [20] donde se utiliza para "olvidar" las unificaciones previas si se vuelve por backtracking. Obviamente el uso del corte afecta a la completitud del proceso de refutación, por lo que debe de ser utilizado con cuidado.

En HTN-PDDL es posible usar el corte para parar el backtracking en dos lugares distintos:

- En la definición de precondiciones: Su uso es exactamente el mismo que el corte de PROLOG. Se toma de la lista de posibles unificaciones una de forma no determinista y se descarta el resto. Si se vuelve por backtracking no se vuelven a considerar el resto de las unificaciones. En dominios con muchas unificaciones usado con cuidado puede utilizarse para acelerar mucho el procesamiento y reducir el backtracking, sin que por ello se vea afectada la completitud.

- En la lista de métodos de una tarea abstracta. Sirve para que una vez que se han probado como ciertas las precondiciones de un método se descarte probar con el resto de métodos. Nuevamente usado con cuidado este corte tampoco tiene por que afectar a la completitud del algoritmo. El escritor de dominios puede conocer que los métodos son mutuamente excluyentes y que una vez que se prueba con uno, el resto ya son inválidos.

Demostremos el uso del corte con un ejemplo:

Ejemplo 15.

```
; Uso de los cortes

(:task Dormir
   :parameters (?p – persona)
   (!
    (:method en-casa
```

```
    :precondition (and
                   (en_ciudad ?p ?c)
                   (residencia ?p ?c ?h)
     )
    :tasks (
            (Ira ?p ?h)
            (Ducharse ?p)
            (Ponerse_Pijama ?p)
            (Dormir ?p)
    )
   )
   (:method en-hotel
    :precondition (and
                    (end_ciudad ?p ?c)
                    (hotel ?c ?h)
                    (camas_libres ?h)
                    (not (residencia ?p ?c ?l)
     )
     :tasks (
            (Ira ?p ?h)
            (Reservar_habitacion ?p ?h)
            (Ducharse ?p)
            (Ponerse_Pijama ?p)
            (Dormir ?p)
    )
)
```

En este ejemplo es muy claro que ambos métodos son excluyentes. La exclusión sin embargo podría no ser tan obvia y ocurrir a niveles más bajos en la jerarquía. El proceso de unificación con los hoteles disponibles en la ciudad puede ser costoso (sobre todo si hay muchos) por lo que si ya sabemos que vamos a dormir en casa, no hace falta que el planificador pruebe todas las posibles unificaciones con todos los hoteles de la ciudad.

Otra forma alternativa de construir la precondición del método en-hotel podría ser la siguiente:

Ejemplo 16.

```
; Uso de los cortes
...
    (:method en-hotel
     :precondition (!(and
                    (end_ciudad ?p ?c)
                    (hotel ?c ?h)
                    (camas_libres ?h)
                    (not (residencia ?p ?c ?l)
      ))
```

```
...
```

Esta precondición al tener un corte en las precondiciones únicamente probará con el primer hotel que encuentre libre en la ciudad. Si se vuelve por backtracking la precondición fallará. Obviamente se pueden perder soluciones válidas, pero la decisión es responsabilidad del escritor de dominios.

14. Gestión del tiempo

HTN-PDDL usa una aritmética basada en el concepto de *timepoint* o referencia temporal para definir relaciones relativas de unas acciones con respecto a otras. Toda acción dispone de dos *timepoints* referenciados con las variables especiales del lenguaje ?start y ?end que marcan el comienzo y el final de la acción.

HTN-PDDL permite almacenar referencias a estos *timepoints* para posteriormente utilizarlos usando una aritmética especial de *timepoints.*

Cuando hacemos uso del tiempo en el dominio o en el problema es necesario declarar en el bloque de requisitos que el planificador debe ser capaz de realizar la gestión del tiempo (ver subsección de requisitos).

14.1. *Acciones durativas*

Las acciones durativas fueron introducidas en PDDL 2.2, para dar soporte temporal al PDDL estándar. Dependiendo de la expresividad que tenga el planificador se dice soporta PDDL al nivel 1 (versión anterior de PDDL 1.2) , 2 (soporte de fluents), 3 (soporte de operadores primitivos durativos) o 4 (soporte de efectos continuos de las acciones en el tiempo). La expresividad de HTN-PDDL llega hasta el nivel 3, es decir hasta el soporte de acciones y literales con tiempo, por lo que los efectos numéricos de las acciones son tratados de una forma discreta.

HTN-PDDL sin embargo tiene una expresividad más rica en el uso del tiempo que PDDL 2.2 nivel 3, aunque las acciones primitivas durativas tienen prácticamente la misma sintaxis:

Las acciones durativas se definen en HTN-PDDL de la siguiente forma:

```
<durative-action-def>:  (:durative-action <name>
                        :parameters (<typed-variable>*)
                        [<meta-tags>]
                        :duration (= ?duration <fluent-exp>)
```

```
                        [:condition <timed-goal>]
                        [:effect <timed-effect>])

<timed-goal>:           (<time-specifier> <sgoal-def>)
                        | <goal-def>

<time-specifier>:       <time-point>
                        | over all
                        | between <time-point> and <time-point>

<time-point>:           at start
                        | at end
                        | <f-time-point>

<f-time-point>:         at <fluent-exp>


<timed-effect>:         (<time-point> <p-effect>)
```

La estructura de la acción durativa es muy similar a la de una acción sin tiempo, con la salvedad de que la durativa tiene asociada una duración codificada como un valor numérico fijo, o calculada a través de una expresión fluent (de esta forma se puede hacer el valor de la duración dependiente del estado). En cualquier caso es requisito indispensable que el valor numérico resultante sea mayor o igual a 0 (las acciones con duración negativa no tienen sentido).

```
Ejemplo 17.
; Poner la duración a una acción
; como se ve los efectos no son continuos
(:action correr
 :parameters (?fromx ?fromy ?tox ?toy)
 :duration (= ?duration (* 15000 (distance ?fromx ?fromy ?tox ?toy)))
 :precondition (in ?fromx ?fromy)
 :effect (and
            (at start (not (in ?fromx ?fromy)))
            (at end (in ?tox ?toy))
))
```

HTN-PDDL permite usar la cláusula *between* para manejar el concepto de *timeline* que se discutirá más adelante.

14.2. *Uso del timeline*

Los literales en el estado están temporizados y definen lo que se denomina un *timeline* o evolución del estado a lo largo del tiempo. La temporización se realiza de forma explícita en la declaración del problema en HTN-PDDL. Si en la declaración del problema un literal no está temporizado entonces se supone que es

válido desde el instante 0 (instante inicial) y hasta que un operador primitivo lo niegue.

Podemos declarar tres tipos de literales distintos:

- Literales que se hacen válidos en un instante de tiempo determinado hasta que el efecto de un operador primitivo lo niegue. Se declaran utilizando la cláusula de PDDL *at:* (at *instante_temporal* (*literal_temporizado*)).

- Literales que son ciertos durante un intervalo de tiempo determinado. Se declaran utilizando la cláusula de HTN-PDDL *between:* (between *instante_temporal_1* and *instante_temporal_2* (*literal_temporizado*)).

- Literales que son ciertos de forma periódica. Se declaran utilizando la cláusula de HTN-PDDL *between:* (between *instante_temporal_1* and *instante_temporal_2* and every *unidades_de_tiempo* (*literal_temporizado*)).

Veamos ejemplos concretos de cada uno de ellos:

```
Ejemplo 18:
;;Declaración de distintos tipos de literales temporizados
(:init
  ;; literales que son válidos desde el instante 0
  (= (precio linea_bus11 110))
  (en alhambra_palace alhambra)
  ;; literales ciertos en un instante de tiempo determinado
  ;; literal válido 120 unidades de tiempo después del inicio
  (at 600 (disponible coche_alquiler))
  ;; literal disponible en un instante temporal fijo
  (at “12:00:00 12/05/2007” (disponible habitacion_101 alhambra_palace))
  ;; literal válido solamente en un intervalo de tiempo
  (between 600 and 120 (fiesta fiesta_cerveza alhambra_palace))
  ;; literal periodico
  (between “08:30:00 12/05/2007” and “18:00:00 12/05/2007” and every 870
(abierto alhambra))
...
)
```

Hay que hacer una serie de consideraciones sobre el ejemplo mostrado. Primero se observa como en HTN-PDDL es posible mezclar referencias a un instante temporal relativas al instante de tiempo con referencias absolutas expresadas como cadenas fecha/hora. De esta forma se facilita la escritura de dominios temporales y la legibilidad de los mismos, pero también nos obligan a introducir una serie de parámetros en el planificador para que este sea capaz de interpretarlos. Estos parámetros son introducidos en la sección de personalización (*customization*) (ver la sección dedicada al bloque de personalización más adelante) del fichero del problema y marcan la unidad temporal que usaremos (en los

literales del ejemplo minutos), el formato de fecha/hora, y el instante temporal que se toma como inicio.

```
Ejemplo:
;; Ejemplo de sección de personalización, para los literales mostrados en
;; el ejemplo anterior
(:customization
   (= :time-format "%H:%M:%S %d/%m/%Y")
   (= :time-horizon-relative 1000)
   (= :time-start " 00:00:00 12/05/2007")
   (= :time-unit :minutes)
)
```

14.3. *Restricciones temporales en las redes de tareas*

Las restricciones temporales se imponen en la definición de las redes de tareas. La sintaxis que se usa en HTN-PDDL es la siguiente:

```
<dur-constraints>:      <dur-constraint>
                        | (and <dur-constraint>+ )
                        | ( )

<dur-constraint>:       (<time-operator> ?dur <fluent-exp>)
                        | (<time-operator> ?start <fluent-exp>)
                        | (<time-operator> ?end <fluent-exp>)

<time-operator>:     =
                     | <
                     | >
                     | <=
                     | >=
```

Cada acción de la red de tareas puede hacer referencia a sus *timepoints* usando las variables especiales "?start" y "?end". También puede hacer referencia a su duración mediante la variable "?dur" a la hora de definir restricciones. Los siguientes ejemplos muestran la definición de algunas restricciones en un ejemplo sencillo:

```
Ejemplo 19:
;; Tres tareas abstractas con relaciones de herencia
;; La tarea A (MA) no impone más restricciones que las de ordenación
(:task A
 :parameters ()
 (:method MA
  :precondition ()
  :tasks ((A1)(A2))
))

;; La tarea B (MB1) pone restricciones en el inicio y el fin de la
```

```
;; subtarea A2
(:task B
 :parameters ()
 (:method MB1
  :precondition ()
  :tasks ((A1)
          ((and (>= ?start 3)(<= ?end 5)) (A2)))
))

;; Definición de la tarea A2 utilizada por las tareas
;; abstractas anteriores
(:task A2
 :parameters ()
 (:method A2
  :precondition ()
  :tasks ((a21)(a22))
))
```

Observar cómo en el método MB1 de la tarea del ejemplo B, se impone la restricción de que la tarea A2 se tiene que ejecutar en el intervalo de tiempo [3,5]. Existe un mecanismo de herencia de restricciones temporales que hace que las tareas hijas en una red de tareas hereden las restricciones de sus tareas abstractas padre. De esta forma, en nuestro ejemplo si las tareas a21 y a22 durasen entre las dos (se tienen que ejecutar en secuencia) más de dos unidades temporales, se produciría una violación temporal que provocaría un backtracking en el planificador.

Las variables *?start* y *?end* de una tarea también se pueden utilizar para definir sincronizaciones complejas entre tareas o límites máximos o mínimos en la distancia temporal entre dos tareas. Estas variables son referencias al inicio y al fin de la tarea en el modelo temporal subyacente

Estas referencias (o *landmarks*) forman parte de la estructura de tipos básica gestionada en HTN-PDDL. De este modo se permite que formen parte de los argumentos de un literal y por lo tanto pueden ser insertadas en el estado de planificación para ser posteriormente ser consultadas más adelante en el proceso de planificación y servir para definir restricciones en otra red de tareas. Por comodidad los *landmarks* suelen ser definidos en la descripción de dominios como *fluents* que pueden ser utilizados directamente en los operadores de comparación al describir una restricción.

El siguiente ejemplo muestra la inserción en el estado del planificador de los landmarks:

Ejemplo 20:

```
;; inserción de los landmarks de una tarea abstracta y de un operador
```

```
primitivo en el estado
(:task A2
 :parameters ()
 (:method MA2
  :precondition ()
  :tasks ((:inline () (and (assign (start A2) ?start)
                           (assign (end A2) ?end)))
          (a21)
          (a22))
))

(:durative-action b
 :parameters()
 :duration (= ?duration 1)
 :condition()
 :effect(and (assign (start b) ?start)
         (assign (end b) ?end)
))
```

Observar cómo se hace uso de las acciones *inline* para introducir los *landmarks* de la tarea abstracta A2. Recordemos que las acciones *inline* se usan en HTN-PDDL para introducir conocimiento de control en el estado del planificador, como es la información temporal, y que estas acciones no deben ser contempladas en el plan final. Al operador primitivo b en cambio no le queda más remedio que insertar la información temporal como efectos.

Esta información temporal puede ser utilizada para definir restricciones en la red de tareas de un método distinto, como se aprecia en el siguiente ejemplo:

```
Ejemplo 21:
;; Uso de un landmark guardado con anterioridad para hacer una
;; sincronización.
(:task A3
 :parameters ()
 (:method MA3
  :precondition ()
  :tasks (((= ?start (start A2)) (b)))
))
```

En el ejemplo la tarea A3 en su método MA3 utiliza el landmark guardado en el ejemplo 20 del inicio de la tarea A2, para sincronizar el inicio de la acción b con dicha tarea.

15. Definición de un problema en HTN-PDDL

El fichero de problema en HTN-PDDL incluye el estado inicial del entorno y el objetivo a cumplir expresado como una red de tareas a resolver. En esto difiere del modelo planteado en PDDL, que está diseñado para dominios planos. También el manejo que se da a los timed initial literals es ligeramente distinto en HTN-PDDL.

La notación de la sintaxis de un fichero de problema de HTN-PDDL es la siguiente:

```
problem_definition:  (define <problem-name> <domain-name>
                     <problem-body>)

<problem-name>:      (problem <name>)

<domain-name>:       (:domain <name>)

<problem-body>:      [<require-def>]
                     [<customization-def>]
                     [<obj-declaration]
                     [<init>]
                     <goal>
```

Cada problema tiene un nombre para identificarlo y va asociado a un dominio también con un nombre determinado. Es un error sintáctico introducir en el planificador un problema y un dominio donde los nombres de dominio no coincidan.

15.1. *Declaración de constantes*

Las constantes (objetos) que se usan en la descripción del problema se declaran como una lista usando la siguiente notación:

```
<obj_declaration>:      (:objects <constant-def>* )
```

Es muy sencillo de entender mediante un ejemplo:

```
Ejemplo 22.
; Declaración de constantes
(:objects
   manuel_diaz – persona
   bugs_bunny – dibujo_animado
   A320 – vehiculo_aereo
   ....
```

```
)
```

15.2. *Declaración del estado inicial*

Los literales que son ciertos en el estado inicial se declaran en la sección de inicialización, cuya sintaxis es la siguiente:

```
<init>:        (:init <init-el>* )

<init-el>:     <atomic-formula(constant|number)>
               | (= <atomic-formula(constant|number) number)
               | (<f-time-point> <atomic-formula(constant|number))
               | (between <date> and <date>
               [and every <number>]
               <atomic-formula(constant|number)> )
```

La diferencia principal con respecto a PDDL es la posibilidad de declarar literales que se hacen ciertos de forma periódica, o en ciertos instantes de tiempo, como ya se mostró en la subsección dedicada al *timeline*:

```
Ejemplo 23.
; Declaración de literales
(:init
   ;; un literal normal
   (fabricante A320 Airbus_SAS)
   ;; un literal fluent
   (= (velocidad_maxima airbus_320) 0.82)
   ;; un literal temporizado que se hace cierto
   ;; en un instante determinado
   (at “28/03/1988 12:00:00” (disponible A320))
   ;; declaración de un evento periodico
   (between “01/01/2007 00:00:00” and  “01/01/2007 11:59:59” and every 12
(es-maniana))
   ;; si, antes del uno de enero del 2007 no había mañanas ;).
   ;; observar que estamos usando como unidad de tiempo las horas,
   ;; esto se fija en la sección de personalización (customization)
   ....
)
```

15.3. *Declaración del objetivo*

El objetivo a resolver se declara como una red de tareas. Se entiende que debe de haber al menos una tarea. En otro caso se obtendría un plan vacío. Las

tareas a incluir en la red de tareas, normalmente serán abstractas. En el caso de que todas fueran primitivas, el plan resultante sería simplemente una ordenación de las tareas primitivas (con marcas de tiempo o no dependiendo si el dominio es temporal) que se han escrito (si estas son aplicables en el estado inicial).

La gramática del lenguaje que se usa en la construcción de un objetivo es la siguiente:

```
<goal>:        (:tasks-goal
               [<comment>]
               :tasks <task-network>
               )
```

Un ejemplo sencillo de declaración de objetivo es la siguiente:

```
Ejemplo 24.
; Definición de objetivo
(:tasks-goal
 :tasks [
         (Tarea1)
         (Tarea2)
         (Tarea3)
        ]
)
```

16. Parámetros de usuario

Con la inclusión de la información temporal y otras características de HTN-PDDL y a fin de hacer más sencilla la escritura de dominios, se permite la personalización de algunos parámetros de usuario en la sección *customization.* La sección *customization* tiene la siguiente notación para su sintaxis:

```
<customization-def>:    (:customization <customization-element>*)

<customization-element>:   (= :time-unit <time-unit>)
                           | (= :time-format <text>)
                           | (= :time-horizon-limit <date>)
                           | (= :time-horizon-relative <date>)
                           | (= :time-start <date>)

<time-unit>:         :hours
                     | :minutes
                     | :seconds
                     | :days
                     | :years
                     | :months
```

En la versión 1.1 todos los parámetros tienen que ver con información temporal, pero esta previsto que esta sección incluya muchos otros parámetros en la siguiente versión.

El significado de cada uno de los parámetros es el siguiente:

- El parámetro **time-unit** define la unidad de tiempo a utilizar cuando es necesario utilizar referencias temporales relativas. Las referencias temporales relativas se codifican mediante un valor numérico. Por ejemplo si (= :time-unit hours), cuando se utilice un 3 en una expresión temporal, significará 3 horas.

- El parámetro **time-format** define el formato que se usará para las referencias temporales absolutas. Por comodidad y legibilidad del dominio se permite definir fechas y horas utilizando cadenas de texto. time-format define el formato de estas cadenas utilizando el mismo formato que el usado por la función del estándar ANSI de C strftime, que es el más extendido.

- El parámetro **time-start** se usa marcar el punto de tiempo inicial, que las acciones tomarán como referencia a la hora de anclarse. Ninguna acción debería comenzar antes del time-start.

- El parámetro **time-horizon** define el horizonte temporal hasta el que el planificador permitirá que las acciones se deslicen. Este parámetro además es utilizado para cortar la generación de intervalos de tiempo para literales intervalares. En principio estos literales podrían generar infinitos intervalos, como cada intervalo supone un punto de anclaje, el planificador podría caer en un bucle infinito hacia adelante y hacia atrás cuando vuelva por backtracking.

- El parámetro **time-horizon-relative**: es equivalente al *time-horizon* pero en lugar de tomar una referencia de tiempo absoluta, la toma relativa al inicio.

Ejemplo 25.

```
; Ejemplo de sección de personalización.
(:customization
   (= :time-format "%d/%m/%Y %H:%M:%S")
   (= :time-horizon-relative 1000)
   (= :time-start "27/1/2007 8:10:0")
   (= :time-unit :minutes)
)
```

El lenguaje permite que la sección de personalización aparezca tanto en el fichero de dominio como en el de problema, aunque lo más recomendable es que los parámetros *time-start* y *time-horizon* aparezcan en el fichero de problema. De otra forma en cierta medida se pierde la independencia del fichero de dominio del problema.

17. Requisitos

La sección de requisitos (*requirements)* define un conjunto de flags con características que el planificador debe soportar para hacer frente al dominio o al problema. Como no todos los planificadores son capaces de soportar la expresividad de HTN-PDDL, esto permite al diseñador de dominio imponer requisitos al planificador y al planificador descartar de una forma rápida a los problemas que no es capaz de abordar.

HTN-PDDL añade algunos requisitos más a los ya existentes en PDDL 2.2:

```
<require-def>:       (:requirements <require-key>*)

<require-key>:       :strips
                   | :typing
                   | :negative-preconditions
                   | :disjuntive-preconditions
                   | :equality
                   | :existential-preconditions
                   | :universal-preconditions
                   | :quantified-preconditions
                   | :conditional-effects
                   | :fluents
                   | :adl
                   | :durative-actions
                   | :derived-predicates
                   | :timed-initial-literals
                   | :metatags
                   | :htn-expansion
```

Por defecto todo planificador tiene que soportar el modelo de STRIPS, por lo tanto este *requirement* siempre se supone aunque no se establezca explícitamente. La descripción de las características que cada *requirement* imponen al planificador es la siguiente:

- **strips:** Modelo de acciones similar al de STRIPS.
- **typing:** Soporte para tipar los objetos del dominio respetando una jerarquía de tipos.

- **netative-preconditions:** Permite usar negaciones en las precondiciones.
- **disjuntive-preconditios:** Permite usar el operador "or" en las precondiciones.
- **equality:** Permite utilizar el predicado de igualdad "=" para la comparación de constantes.
- **existential-preconditions:** Uso de precondiciones cuantificadas existencialmente.
- **universal-preconditions:** Uso de precondiciones cuantificadas universalmente.
- **quantified-preconditions:** Es "azúcar sintáctica" ya que es equivalente a declarar en un dominio el soporte para precondiciones existenciales y universales.
- **conditional-efects:** Permite el uso del operador "when" para definir efectos condicionales.
- **fluents:** Permite el uso de la aritmética con números enteros y reales, así como la utilización de operadores matemáticos en las precondiciones. Permite asignar números a predicados y hacer operaciones con ellos en los efectos.
- **adl:** Incluye todas las capacidades expresivas del lenguaje ADL *(Action Description Language)* [106] . Es equivalente a declarar en un dominio que el planificador debe soportar los requisitos de *Strips, typing, negative-preconditions, disjuntive-preconditions, equality, quantified-preconditions,* y *conditional-effects".*
- **durative-actions:** Permite el uso de acciones durativas en el dominio. Cuando se impone este requisito implícitamente también se hace el de *fluents.*
- **derived-predicates:** Permite la definición de axiomas en el dominio.
- **timed-initial-literals:** Permite el uso de predicados temporizados. Destacar que la filosofía y el tratamiento de este tipo de predicados es **distinto** en HTN-PDDL que en el PDDL 2.2 nivel 3 estándar.
- **metatags:** Permite incluir tags en diferentes partes del dominio con información extra no imprescindible para el planificador, pero necesaria en procesamientos posteriores del plan por otras herramientas.

- **htn-expansion:** Requiere el manejo de redes jerárquicas y tareas abstractas.

III - ANEXO: Invocación del planificador desde la línea de órdenes

1. Obtener una copia de HTNP2CL

Para obtener una copia del planificador HTNP2CL, escriba a alguna de las direcciones de correo electrónico de los autores: o.garcia@iactive.es, l.castillo@decsai.ugr.es, faro@decsai.ugr.es, f.palao@iactive.es, o t.garzon@iactive.es.

2. Invocación del planificador

La sintaxis para invocar al planificador HTNP2CL desde la línea de órdenes es la siguiente[20]:

```
Sintaxis: ./htnp2cl [opciones] --domain_file (-d) <domain.pddl> --
problem_file (-p) <problem.pddl>
```

Donde las opciones son las siguientes:

- --help (-h): Pantalla de ayuda.
- --debug (-g): Lanza el planificador en modo de depuración del dominio.
- --verbose (-v[<level>] i.e: -v1): Lanza el planificador en modo verbosidad, para imprimir por pantalla las distintas decisiones que va tomando el planificador. Hay tres niveles de verbosidad (1,2 o 3) que aumentan progresivamente el número de mensajes que se imprimen por pantalla. Por defecto es 2.
- --output_file (-o) {filename}: Escribe el plan resultante como un fichero de

20 Versión 0.705 21 de Mayo del 2007.

texto plano en el fichero pasado como argumento.

- --xml_file (-x) {filename}: Escribe el plan resultante en formato xml para su posterior postproceso en el fichero pasado como argumento.
- --expansions_limit <number> : Número máximo de expansiones (aplicaciones de métodos) permitido. Sirve como límite del backtracking permitido durante la búsqueda.
- --depth_limit <number> : Nivel máximo de profundidad en el árbol de expansión permitido durante la búsqueda de un plan.
- --time_limit <number> : Tiempo máximo permitido para la búsqueda de un plan.

3. Invocación del planificador en modo servidor

El planificador puede ser lanzado como un servicio de planificación que puede ser invocado a través de internet usando el protocolo de comunicación XML-RPC. Generalmente es el módulo Metaserver el que se encarga de lanzar el planificador en este modo.

Cuando el planificador se lanza en este modo se le deben pasar dos argumentos más.

- --port (-p) <number>: Puerto en el cual el planificador quedará escuchando a la espera por defecto es el 8080.
- --key (-k) <string>: Es una clave o llave que el servicio que levanta el planificador debe establecer. El planificador no responderá a los requerimientos de servicio a través de su API a no ser que la clave sea correcta, de esta forma se garantiza que ningún servicio no autorizado (que no conozca la clave) pueda acceder al plan resultante.

La API XML-RPC de HTNP2CL es plenamente funcional pero aún está en fase experimental y probablemente cambie en las siguientes versiones. Existen métodos para lanzar el planificador en modo de depuración en modo remoto y controlar dicha depuración pero en esta sección y también para obtener todo tipo de información acerca del plan y estadísticas de planificación. Nos limitaremos a describir las funciones de la API fundamentales.

```
void eofPlanProcess::execute(XmlRpcValue& params, XmlRpcValue& result)

Comprueba si el planificador ha terminado su ejecución.
Lee el estado en el que se encuentra el planificador.
```

```
En el caso de que el planificador haya sido lanzado con una clave (key)
para el acceso seguro, hay que pasarla como primer argumento.
Los estados en los que se puede encontrar el planificador son los
siguientes:
   1 = Se encontró un plan
   0 = El planificador está buscando una solución
   -1 = Se detectó un error
   -2 = Error en la llave, llave incorrecta

void getXMLPlan::execute(XmlRpcValue& params, XmlRpcValue& result)

Devuelve información del plan obtenido en formato XML

void finalize::execute(XmlRpcValue& params, XmlRpcValue& result)

Manda una señal de finalización al planificador.

void readOStream::execute(XmlRpcValue& params, XmlRpcValue& result)

Lee el flujo de salida donde el planificador escribe los mensajes con
información.
El primer argumento es la key (clave).
Devuelve una cadena con el contenido del flujo.
```

IV - ANEXO: Guía de uso del depurador integrado

Como se ha comentado el planificador viene con un depurador integrado imprescindible para la depuración de dominios complejos. El depurador es muy similar al gdb de la gnu y se active poniendo el flag "-g" en la llamada al planificador. Los comandos principales son los siguientes:

```
Command: `print'. Shortcut: `p'

Description: Prints information about the planning context.
        `print state':  Prints the current state.
        `print agenda': Prints the current agenda.
        `print plan':   Prints the ongoing plan.
        `print options':        Prints the options you can peform on next
step.
        `print termtable':      Prints the internal term table.
        `print tasks':  Prints all the tasks defined in the domain.
        `print predicates':     Prints all the avaliable predicates
defined in the domain.
        `print <predicate-expression>': Prints all the predicates in the
current state, or the tasks in current plan, that match the given
expression.

Command: `display'. Shortcut `d'.

Description: Display information about the current planning context every
step until is undisplayed.
        `display':      Shows information about the current displays.
        `display <number>':     Activates the display of given number.
        `display state':        Displays the current state.
        `display agenda':       Displays the current agenda.
        `display plan': Displays the ongoing plan.
        `display termtable':    Displays the internal term table.
        `display <predicate-expression>':       Displays all the
predicates in the current state, or the tasks in current plan, that match
the given expression.
```

```
Command: `undisplay <number>'.

Description: Deactivates the display of given number.

Command: `break'. Shortcut `b'

Description: Manages the stablished breakpoints.
        `break':        Lists all defined breakpoints.
        `break <number>':       Prints breakpoint whith given id.
        `break <predicate>':    Defines a new breakpoint. <predcate> can
be a task definition or a simple predicate.
See also: `watch', `disable', `enable'.

Command: `watch <precondition>'. Shortcut `s'

Description: Defines a condition where the debugger will stop.
If ithe watch is enabled, the debugger will stop every time the condition
produce one or more valid unifications.
See also: `break', `disable', `enable'.

Command: `enable <number>'.

Description: Reactivates a previously disabled breakpoint or watch.
<number> is de id of the breakpoint or watch to enable.
See also: `break', `watch', `disable'.

Command: `disable <number>'.

Description: Deactivates a breakpoint or watch.
<number> is de id of the breakpoint or watch to disable.
See also: `break', `watch', `enable'.

Command: `describe'.

Description: Shows detailed information about a structure defined in the
domain.
        `describe <predicate-expression>':      Print the description in
the domain relative to the predicate expression.    can be a predicate or
a task.
See also: `print', `display'

Command: `next'. ShortCut: `n'

Description: Advance one more step.
See also: `continue'

Command: `continue'. Shortcut: `c'

Description: Continues the execution until a breakpoint or the end of the
program is reached.
See also: `next'
Command: `eval <precondition>'.
```

```
Description: Evaluates the given expression and prints the produced
unifications.

Command: `apply <effect>'.

Description: Applies the given effect.
Be cautious with this command is dangerous.
See also: `eval'.
```

V - ANEXO: Dominio HTN-PDDL y SHOP2 del mundo de bloques

1. Dominio HTN-PDDL

```
(define (domain bloques)
(:requirements
  :typing
  :fluents
  :derived-predicates
  :negative-preconditions
  :htn-expansion)

(:types
  bloque superficie - object)

(:constants mesa - superficie)

(:predicates
  (manovacia)
  (libre ?x - bloque)
  (cogido ?x - bloque)
  (sobremesa ?x - bloque)
  (sobre ?x ?y - bloque)
  (igual ?x ?y)
  (distinto ?x ?y))

(:derived
  (igual ?x ?x) ())

(:derived
  (distinto ?x ?y) (not (igual ?x ?y)))

(:task sobre
 :parameters (?x ?y)
 (:method poner_encima
  :precondition () ; vacío
```

```
  :tasks ((limpiar ?x)(limpiar ?y)(colocar ?x ?y))))

(:task limpiar
 :parameters (?x)
 (:method limpiar_mesa
  :precondition (igual ?x mesa)
  :tasks())
 (:method limpiar_libre
  :precondition (libre ?x)
  :tasks ())
 (:method limpiar_ocupado
  :precondition (sobre ?y ?x)
  :tasks ((limpiar ?y)(colocar ?y mesa))))

(:task colocar
 :parameters (?x ?y)
 (:method colocar
  :precondition ()
  :tasks ((primero-coge ?x)(despues-deja ?x ?y))))

(:task primero-coge
 :parameters (?x - bloque)
 (:method cogelo_de_la_mesa
  :precondition (sobremesa ?x)
  :tasks (coger ?x))
 (:method cogelo_de_la_pila
  :precondition (sobre ?x ?y)
  :tasks (desapilar ?x ?y)))

(:task despues-deja
 :parameters (?x - bloque ?y - object)
 (:method dejalo_en_la_mesa
  :precondition (igual ?y mesa)
  :tasks (dejar ?x))
 (:method dejalo_en_la_pila
  :precondition (distinto ?y mesa)
  :tasks (apilar ?x ?y)))


(:action coger
 :parameters (?x - bloque)
 :precondition (and (sobremesa ?x)(libre ?x)(manovacia))
 :effect (and (not (sobremesa ?x)) (not (libre ?x))(not (manovacia))
         (cogido ?x)))

(:action dejar
 :parameters (?x - bloque)
 :precondition (cogido ?x)
 :effect (and (sobremesa ?x) (libre ?x) (manovacia)
              (not (cogido ?x))))

(:action apilar
```

```
 :parameters (?x ?y - bloque)
 :precondition (and (cogido ?x)(libre ?y))
 :effect (and (not (cogido ?x)) (not (libre ?y)) (libre ?x) (sobre ?x ?y)
(manovacia)))

(:action desapilar
 :parameters (?x ?y - bloque)
 :precondition (and (manovacia) (libre ?x) (sobre ?x ?y))
 :effect  (and (cogido ?x) (libre ?y) (not (libre ?x)) (not (sobre ?x ?y))
(not (manovacia))))
)
```

2. Dominio SHOP2

```
(defdomain bloques (

 (:- (igual ?x ?x) nil)
 (:- (distinto ?x ?y) ((not (igual ?x ?y))))

(:method (sobre ?x ?y)
 ()
 ((:task :immediate limpiar ?x)(:task :immediate limpiar ?y)(:task
:immediate colocar ?x ?y))
)

(:method (limpiar ?x)
 (igual ?x mesa)
 ()

 (libre ?x)
 ()

 (sobre ?y ?x)
 ((:task :immediate limpiar ?y) (:task :immediate colocar ?y mesa))
)

(:method (colocar ?x ?y)
 ()
 ((:task :immediate primero-coge ?x)(:task :immediate despues-deja ?x ?y))
)

(:method (primero-coge ?x)
 (sobremesa ?x)
 ((!coger ?x))

 (sobre ?x ?y)
 ((!desapilar ?x ?y))
)

(:method (despues-deja ?x ?y)
 (igual ?y mesa)
 ((!dejar ?x))
```

```
 (distinto ?y mesa)
 ((!apilar ?x ?y))
)


(:operator (!coger ?x)
    ((sobremesa ?x) (libre ?x) (manovacia))
    ((sobremesa ?x) (libre ?x) (manovacia))
    ((cogido ?x))
)

(:operator (!dejar ?x)
      ((cogido ?x))
      ((cogido ?x))
      ((sobremesa ?x) (libre ?x) (manovacia))
)

(:operator (!apilar ?x ?y)
      ((cogido ?x) (libre ?y))
      ((cogido ?x) (libre ?y))
      ((libre ?x) (sobre ?x ?y) (manovacia))
)

(:operator (!desapilar ?x ?y)
      ((manovacia) (libre ?x) (sobre ?x ?y))
      ((libre ?x) (sobre ?x ?y) (manovacia))
      ((cogido ?x) (libre ?y))
)
)

)
```

VI - *Bibliografía*

[1]: A. Garrido, E. Onaindía, and F. Barber, *Time-optimal planning in temporal problems*, 2001. European Conference on Planning .Pg: 397-502.

[2]: A. Tacchella and C. Castellini and E. Giunchiglia, *SAT- based planning in compex domains: Concurrency constraints an nondeterminism*, 2003. Artificiall Intelligence. Vol: 147: Pg: 85-118.

[3]: A.L Blum and M.L. Furst, *Fast planning through planning graph analysis*, 1997. Artificial Intelligence. Pg: 281-300.

[4]: Alfonso Gerevini and Derek Long, *BNF Description of PDDL 3.0*, 2005. ICAPS 2006.

[5]: Allen, J. F., *An interval-based representation of temporal knowledge*, 1981.

[6]: Allen, J. F., *Towards a general theory of action and time*, 1984. Vol: 23 Pg: 123, 154.

[7]: Ambros-Ingreson, J. A. and Steel, S., *Integrating planning, execution and monitoring*, 1988. AAAI AAAI88. Pg: 83-88.

[8]: Antoniou, G. and van Harmelen, F., *Web Ontology Language*: OWL, 2004. Handbook on Ontologies Pg: 67—92.

[9]: Avesani, P. and Perini, A. and Ricci, F., *CBET: a case base exploration tool*, 1997. Lecture Notes in Computer Science. Vol: 1321: Pg: 405.

[10]: Bart Selman and Henry Kautz, *Planning as satisfiability*, 1992. ECAI .

[11]: Bonet, B. and Geffner, H., *HSP: Heuristic search planner*, 1998. AIPS-98 Planning Competition Pittsburgh, PA.

[12]: Bonet, B. and Geffner, H., *Heuristic Search Planner 2.0*, 2001. AI Magazine . Vol: 22-3: Pg: 77—80.

[13]: Booch, G., *Object-oriented analysis and design*, 1996. Addison-Wesley.

[14]: BPM Initiative, *Business Process Modeling Execution Language*, 2002. http://www.bpmi.org.

[15]: Bresina, J. and Jonsson, A. and Morris, P. and Rajan, K., *Activity Planning for the Mars Exploration Rovers*, 2005. The International Conference on Automated Planning and Scheduling (ICAPS).

[16]: C. Pereria and F. Garcia and J. Lang and R. Martin-Clouaire, *Planning with graded nondeterministic actions: a pissibilistic approach*, 1996. International Journal of Artificial Intelligence Research. Vol: 4: Pg: 287-339.

[17]: Castillo, L. and Fdez-Olivares, J. and Gonzalez, A., *On the Adequacy of Hierarchical Planning characteristics for Real-World Problem Solving*, 2001. Proceedings of the Sixth European Conference on Planning ECP. Vol: 1.

[18]: Chapman, D., *Planning for conjunctive goals*, 1987. Vol: 32 Pg: 333, 377.

[19]: Cimatti, A. and Roveri, M., *Conformant Planning via Symbolic Model Checking*, 2000. JAIR. Vol: 13: Pg: 305—338.

[20]: Clocksin, W.F. and Mellish, C.S., *Programming in Prolog*, 1984. Springer-Verlag New York, Inc. New York, NY, USA.

[21]: Cohen, PR and Greenberg, ML and Hart, DM and Howe, AE, *Trial by fire: understanding the design requirements for agents in complex environments*, 1998. AI Magazine American Association for Artificial Intelligence Menlo Park, CA, USA. Vol: 10-3: Pg: 34-48.

[22]: Cormen, T.H., *Introduction to Algorithms*, 2001. MIT Press.

[23]: Currie, K. and Tate, A., *O-Plan: The open Planning Architecture*, 1989. University of Edinburgh, Artificial Intelligence Applications Institute.

[24]: D. Draper and S. Hanks and D. Weld, *Probabilistic planning with information gathering and contingent execution*. Technical Report TR93-12-04, 1993.

[25]: D.E. Wilkins, *Domain-independent planning: Representation and plan generation*, 1984. Artificial Intelligence . Vol: 22: Pg: 269—301.

[26]: D. E. Wilkins, *Practical planning: Extending the classical AI planning paradigm*, 1988, Morgan Kaufmann.

[27]: D. E. Wilkins and R. V. Desimone, *Applying an AI plannet to military operations planning*, 1994,M. Zweben and M. S. Fox, Intelligent Scheduling, Morgan

Kaufmann.

[28]: D. Nau and T.C. Au and O. Ilghami, *SHOP2: An HTN Planning System*, 2003, Journal of Artifial Intelligence Research, 20, Pg: 379-404.

[29]: D.E. Wilkins, *Domain-independent planning: Representation and plan generation*, 1984, Artificial Intelligence,22-, Pg: 269-301.

[30]: Dave Winer et al., *XML-RPC Specification*, 1999, Technical report, UserLand Software http://www.xmlrpc.com/spec,15, Pg: 7.

[31]: Dean, M. and Schreiber, G. and others, *OWL Web Ontology Language Reference*, 2004, W3C Recommendation,10.

[32]: Dechter, R. and Meiri, I. and Pearl, J., *Temporal constraint networks*, 1991, Elsevier Science Publishers Ltd. Essex, UK, Artificial Intelligence,49 (1-3), Pg: 61—95.

[33]: Erol, K. and Hendler, J. and Nau, D., *UMCP: A sound and complete procedure for hierarchical task-network planning*, 1994, Proc. 2nd Intl. Conf. on AI Planning Systems, Pg: 249-254.

[34]: Erol, K. and Hendler, J. and Nau, D.S., *HTN planning: Complexity and expressivity*, 1994, Proceedings of the Twelfth National Conference on Artificial Intelligence, Pg: 1123-1128.

[35]: Erol, K. and Hendler, J.A. and Nau, D.S. and Tsuneto, R., A Critical Look at Critics in HTN Planning, 1995,, International Joint Conference on Artificial Intelligence,14-2, Pg: 1592-1598

[36]: F. Palao-Reinés, *Improving Planning Techniques for Web Services*, 2006, Doctoral Consortium ICAPS 06.

[37]: Fikes R. E. and Nilsson N. J., *STRIPS: A new approach to the application of theorem proving to problem solving*, 1971, Artificial Intelligence, 2, Pg: 189, 208.

[38]: Fikes, R. E., Hart, P. E., Nilsson, N. J., *Learning and executing generalized robot plans*, 1972, Artificial Intelligence, 3, Pg: 251, 288.

[39]: Fikes, R. E., Nilsson, N. J., *STRIPS, a retrospective*, 1993.

[40]: Fox M. and Long D., *PDDL2-1: an extension to PDDL for expressing temporal planning domains*, 2001, University of Durham, UK.

[41]: Fox, M. and Long, D., *PDDL+ level 5: An Extension to PDDL2. 1 for Modelling Planning Domains with Continuous Time-dependent Effects*, 2001, Technical report,

University of Durham.

[42]: G. Rabideau and R. Knight and S. Chien and A. Fukunaga and A. Govindjee, *Iterative repair planning for spacecraft operations using the ASPEN system*, 1999.

[43]: García-Pérez, O., *A Mixed Initiative Framework for Dynamic Environments*, 2005, Doctoral Consortium, ICAPS, Pg: 39-42.

[44]: García-Pérez, O., *Mixed Initiative Techniques Applied to a Plannig Problem*, 2005.

[45]: Garrido, A. and Onaindía, E., *Domain-independent temporal planning in a planning-graph-based approach*, 2006, IOS Press, AI Communications,19-4, Pg: 341-367.

[46]: Georgeff, M.P. and Lansky, A.L., *Reactive reasoning and planning*, 1987, Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), Pg: 677- 682.

[47]: Gerevini, A. and Long, D., *Plan Constraints and Preferences in PDDL3*, 2006, ICAPS Workshop on Soft Constraints and Preferences in Planning.

[48]: Gerevini, A. and Serina, I., *LPG: a planner based on local search for planning graphs with action costs*, 2002, Proc. of the Sixth Int. Conf. on AI Planning and Scheduling, Pg: 12- 22.

[49]: Ghallab, M. and Howe, A. and Knoblock, C. and McDermott, D. and Ram, A. and Veloso, M. and Weld, D. and Wilkins, D., *PDDL: the planning domain definition language*, 1998, AIPS-98 Planning Committee.

[50]: Ghallab, M. and Nau, D. and Traverso, P., *Automated planning*, 2004, Elsevier.

[51]: Ghallab, M.and Vidal, T., *Focusing on a Sub-graph for Managing Efficiently Numerical Temporal Constraints*, 1995, Proceedings of FLAIRS, vol. Melbourne Beach (FL).

[52]: Golden, K. and Etzioni, O. and Weld, D., *Omnipotence without omniscience: Efficient sensor management for planning*, 1994, AAAI.

[53]: Gómez-Pérez A. and Fernández-López M. and Corcho O., *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web*, 2004, Springer.

[54]: Green, C., *Application of theorem proving to problem solving*, 1969, Pg: 741- 747.

[55]: H. Geffner and B. Bonet, *Planning with incomplete information as heuristic search in belief space*, 2000, AAAI Press, Pg: 52-61.

[56]: H. Muñoz-Avila and D. W. Aha and L. Breslow and D. Nau, *HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations*, 1999, AAAI Press, Ninth Conference on Innovative Applications of Artificial Intelligence, Pg: 879-885.

[57]: Hammond, K.J., *Case-Based Planning: A Framework for Planning from Experience*, 1990, Lawrence Earlbaum, Cognitive Science, 14-3, Pg: 385- 443.

[58]: Hoffmann, J. and Nebel, B., *The FF planning system: Fast plan generation through heuristic search*, 2001, Journal of Artificial Intelligence Research, 14, Pg: 253-302.

[59]: Immaneni, T., & Cox, M. T., *GTrans: An Application for mixed-initiative collaborative planning during emergency response situations*, 2004, W.W. Smari & W. McQuay, Proceedings of the 2004 International Symposium on Collaborative Technologies and Systems (CTS 04), Pg: 121-126.

[60]: Intelligent scheduling, *HSTS: integrating planning and scheduling*, 1994, M. Zwebenand M. Fox, Morgan Kaufmann, Pg: 169-212.

[61]: J. McCarthy, P.J. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*, 1969, Edinburgh University Press, Machine Intelligence, 4, Pg: 463- 502.

[62]: J.S. Penberthy and D.S. Weld, *UCPOP: A sound, complete, Partial Order Planner for ADL*, 1992, 3rd. Int. Conf. on Principles of Knowledge Representation and Reasoning, Pg: 103-114.

[63]: John L. Bresina and Ari K. Jonsson and Paul H. Morris and Kanna Rajan, *Mixed-Initiative Planning in MAPGEN: Capabilities and Shortcommings*, 2005.

[64]: K.L. Myers, *CPEF: A continuous planning and execution framework*, 1999, AI Magazine, 20-4, Pg: 63-69.

[65]: K.J. Hammond, *CHEF: A model of case-based planning*, 1986, Proceedings of the Fifth National Conference on Artificial Intelligence, Pg: 267-271.

[66]: Knoblock, C.A., *Automatically Generating Abstractions for Planning*, 1994, Artificial Intelligence, 68-2, Pg: 243-302.

[67]: Koehler J., *Planning from Second Principles*, 1996, Artificial Intelligence, 87.

[68]: Koehler, J. and Nebel, B. and Hoffman, J. and Dimopoulos, Y., *Extending Planning Graphs to an ADL Subset*, 1997, Lecture Notes in Computer ScienceLecture,1348, Pg: 273.

[69]: Kvarnström, J. and Doherty, P., *TALplanner: A temporal logic based forward chaining planner*, 2000, Springer, Annals of Mathematics and Artificial Intelligence, 30-1, Pg: 119-169.

[70]: L. Castillo and A. González, *A Nonlinear Planner for Solving Sequential Control Problems in Manufacturing System*, 1998, H. Coelho, Progress in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Springer-Verlag.

[71]: L. Castillo and J. Fdez-Olivares and A. González, *Mixing expresiveness and efficiency in a manufacturing planner*, 2001, Journal of Experimental and Theoretical Artificial Intelligence, 13, Pg: 141-162.

[72]: L. Castillo, J. Fdez-Olivares, and A. González, *A temporal constraint network based temporal planner*, 2002, Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG 2002,-, Pg: 99-109.

[73]: L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *Temporal enhancements of an HTN planner*, 2005, Spanish Conference on Artificial Intelligence, CAEPIA 2005 (To appear in Lecture Notes on Artificial Intelligence).

[74]: L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *Efficiently handling temporal knowledge in an HTN planner*, 2006, 16th International Conference on Automated Planning and Scheduling (ICAPS 2006).

[75]: L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao., *Bringing users and planning technology together. Experiences in SIADEX*, 2006, 16th International Conference on Automated Planning and Scheduling (ICAPS 2006).

[76]: L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *SIADEX. Un entorno integral de planificación para el diseño de planes de actuación en situaciones de crisis*, 2005, I Jornadas de Transferencia Tecnológica de Inteligencia Artificial, TTIA'2005 (AEPIA).

[77]: L. Castillo, J. Fdez.-Olivares, O. García-Pérez, F. Palao, *Sistema Inteligente de Ayuda a la Decisión para el diseño de Planes de Extinción: SIADEX*, Revista Incendios Forestales.

[78]: L. Pryor and G. Collins, *Planning for contingencies: A decision based approach*, 1996, Journal of Artificial Intelligence Research,4, Pg: 287-339.

[79]: Laborie, P., and Ghallab, M, *Planning with sharableresource constraints*, 1995, IJCAI'95, Pg: 1643-1649.

[80]: Leiserson, C.E. and Saxe, J.B., *A mixed-integer linear programming problem which is efficiently solvable*, 1988, Academic Press, Inc. Duluth, MN, USA, Journal

of Algorithms, 9-1, Pg: 114-128.

[81]: Long D. and Fox M., *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*, 2003, Journal of Artificial Intelligence Research, 20, Pg: 61, 124.

[82]: Long, D. and Fox, M., *Efficient Implementation of the Plan Graph in STAN*, 1999, Journal of Artificial Intelligence Research, 10, Pg: 87-115.

[83]: Luis Castillo Vidal and Lluvia Morales Reynaga and Arturo González Ferrer and Juan Fernandez Olivares and Óscar García Pérez, *Obtaining IMS Learning Designs easier*, CAEPIA07.

[84]: Luis Castillo Vidal and Lluvia Morales Reynaga and Arturo González Ferrer and Juan Fernandez Olivares and Óscar García Pérez, *The integration of AI planning technologies into a Learning Management System*, TTIA07.

[85]: Luis Castillo, Eva Armengol, Eva Onaindía, Laura Sebastiá, Jesús González-Boticario, Antonio Rodríguez, Susana Fernández, Juan D. Arias, and Daniel Borrajo, *SAMAP. An user-oriented adaptive system for planning tourist visits*, Previsto 2008, Expert Systems With Applications (ISSN: 0957-4174).

[86]: M. A. Peot and D. E. Smith, *Conditional nonlinear planning*, AIPS, Pg: 189-197.

[87]: M. de la Asunción and L. Castillo and J. Fdez-Olivares and O. García-Pérez and A. González and F. Palao, *SIADEX: an interactive artificial intelligence planner for decision support and training in forest fire fighting*, 2005, AIComm special issue on Binding Environmental Sciences and artificial intelligence.

[88]: M. de la Asunción, O. García-Pérez, F. Palao, *SIADEX: A Real World Planning Approach for Forest Fire Fighting*, 2004, European Starting Artificial Intelligence Research Symposium. Collocated with the European Conference on Artificial Intelligence.

[89]: M. Do and S. Kambhampati, *SAPA: a domain-independent heuristic metrictemporal planner*, 2001, European Conference on Planning, Pg: 109-120.

[90]: Marc de la Asunción and Luis Castillo and Juan Fernández-Olivares and Oscar García-Pérez and Antonio González and Francisco Palao, *Local (Human Centered) Replanning in the SIADEX Framework*, 2003, In proceedings of Conference of Spanish Association for Artificial Intelligence. II Workshop on Planning, Scheduling and Temporal Reasoning, Pg: 79-88.

[91]: McAllester, D., Rosenblitt, D., *Systematic nonlinear planning*, 1991, AAAI-91, Pg: 634-639.

[92]: McDermott D., *A heuristic estimator for means-ends analysis in planning*, 1996,AAAI Press, Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96), Pg: 142-149.

[93]: Myers K.L., *Metatheoretic Plan Summarization and Comparison*, 2006, AAAI Press, Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06).

[94]: Myers, K.L. and Tyson, W.M. and Wolverton, M.J. and Jarvis, P.A. and Lee, T.J. and desJardins, M., *PASSAT: A User-centric Planning Framework*, 2002, Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space.

[95]: N. Kushmerick and S. Hanks and D. S. Weld, *An algorithm for probabilistic planning*, 1995, Artificial Intelligence, 76, Pg: 239-286.

[96]: N. Muscettola and P. Pandurang Nayak and B. Pell and B. C. Williams, *Remote agent: to boldly go where no AI systems has gone before*, 1998, Artificial Intelligence, 103, Pg: 5-48.

[97]: Nau, D. S. and Cao, Y. and Lotem, A. and Muñoz-Avila, H., *SHOP: Simple Hierarchical Ordered Planner*, 1999, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence table of contents, Pg: 968-975.

[98]: Nau, D.S. and Smith, S.J.J. and Erol, K., *Control strategies in HTN planning: Theory versus practice*, 1998, Proceedings of the National Conference on Artificial Intelligence, 15, Pg: 1127-1133.

[99]: Nebel, B. and Koehler, J., *Plan reuse versus plan generation: a theoretical and empirical analysis*, 1995, Artificial Intelligence, 76-1, Pg: 427-454.

[100]: Newell, A., Simon, H. A., *GPS: a program that simulates human thought*, 1963, Feigenbaum, E. A., Feldman, J., R. Oldemburg KG.

[101]: Nicholas J. Mulcahy and Josep Call, *Apes Save Tools for Future Use*, 2006, Science, 312-5776, Pg: 1038-1040.

[102]: Nilsson, N. J., *Principles of Artificial Intelligence*, 1980, Tioga Publishing.

[103]: Noy, N.F. and Crubezy, M. and Fergerson, R.W. and Knublauch, H. and Tu, S.W. and Vendetti, J. and Musen, M.A., *Protégé-2000: An Open-Source Ontology-Development and Knowledge-Acquisition Environment*, 2003, AMIA Annual Symposium Proceedings, Pg: 953.

[104]: Olawsky, D. and Gini, M., *Deferred planning and sensor use*, 1990, Morgan Kauffman, Innovative Approaches to Planning, Scheduling and Control, Pg: 166-174.

[105]: P. Avesani and A. Perini and F. Ricci, *Interactive Case-Based Planning for Forest Fire Management*, 2000, Applied Intelligence, 13-1, Pg: 41-57.

[106]: Pednault E., *ADL: Exploring the middle ground between STRIPS and the situation calculus*, 1989.

[107]: Ghallab, M. and Laborie, P., *Planning with Sharable Resource Constraints*, Proceedings International Joint Conference on Artificial Intelligence, Pg: 1643-1649.

[108]: Pollack, ME and Joslin, D. and Paolucci, M., *Flaw Selection Strategies for Partial-Order Planning*, 1996, JAIR, 6, Pg: 232-262.

[109]: R. St-Denis and F. Kabanza and M. Barbeau, *Planning control rules for reactive agents*, 1997, Artificial Intelligence, 95, Pg: 67-113.

[110]: Rina Dechter, *Constraint Processing*, Morgan Kauffman Publishers.

[111]: Rumbaugh, J. and Blaha, M. and Premerlani, W. and Eddy, F. and Lorensen, W., *Object-oriented modeling and design*, 1991.

[112]: Ruml, W. and Do, M.B. and Fromherz, M.P.J., *On-line planning and scheduling for high-speed manufacturing*, 2005, Proc. of ICAPS05, Pg: 30-39.

[113]: S. Biundo and B. Schattenberg, *From Abstract Crisis to Concrete Relief - A Preliminary Report on Combining State Abstraction and HTN Planning*, 2001, 6th European Conference on Planning (ECP-01).

[114]: S. Fernández, L. Sebastiá, and J. Fdez-Olivares, *Planning Tourist Visits Adapted to User Preferences*, 2004, Frontiers in AI and Applications: IOS Press, Planning, scheduling and Constraint Satisfaction: From theory to practice, 117, Pg: 119-128.

[115]: S. Hanks and C. Boutilier and T. Dean, *Decision-theoretic planning: Structural assumptions and computational leverage*, 1999, JAIR, 11, Pg: 1-93.

[116]: S. Kambhampati and E. Parker and E. Lambrecht, *Understanding and extending graphplan*, 1998, AAAI.

[117]: Sacerdoti E. D., *The nonlinear nature of plans*, 1975, IJCAI 1975, Pg: 206, 214.

[118]: Sacerdoti, D. E., *Planning in a hierarchy of abstraction spaces*, 1974, Artificial

Intelligence, 5, Pg: 115, 135.

[119]: Schoppers, M.J., *Universal plans for reactive robots in unpredictable environments*, 1987, IJCAI,87, Pg: 1039-1046.

[120]: Shneiderman, B., *Designing the user interface: strategies for effective human-computer interaction*, 1992, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[121]: Shostak, R., *Deciding Linear Inequalities by Computing Loop Residues*, 1981, ACM Press New York, NY, USA, Journal of the ACM (JACM), 28-4, Pg: 169-779.

[122]: SMITH, D.E. and FRANK, J. and JONSSON, A.K., *Bridging the gap between planning and scheduling*, 2000, Cambridge Univ Press, The Knowledge Engineering Review, 15-1, Pg: 47-83.

[123]: Smith, D.E. and Weld, D.S., *Temporal planning with mutual exclusion reasoning*, 1999, Proceedings of IJCAI,99, Pg: 326-333.

[124]: Smith, D.E. and Weld, D.S. Smith, D.E. and Weld, D.S., *Conformant graphplan*, 1998, Proceedings of the Fifteenth National Conference on Artificial Intelligence, Pg: 889 – 896.

[125]: Smith, S.J.J. and Nau, D.S. and Throop, T.A., *Computer Bridge - A Big Win for AI Planning*, 1998, AI Magazine, 19-2, Pg: 93-106.

[126]: Srinivasan, R. and Howe, A., *Comparison of methods for improving search efficiency in a partial-order planner*, 1995, Proc. 14th Int. Joint Conf. AI, Pg: 1620-1626.

[127]: Srivastava Biplav, *A Limited Extension of PDDL for planning with Non-Primitive Actions*, 2003, Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks. ICAPS.

[128]: Stefan Edelkamp, Jörg Hoffman, *PDDL2.2: The Language for the Classical Part of the 4th international Planning Competition*, 2004.

[129]: Subbarao Kambamphati and Romeo Sanchez Nigenda and Xuan Long Nguyen, *AltAlt: Combining the advantages of graphplan and heuristic state search*, 2000, International Conference on Knowledge-based Computer Systems.

[130]: Sussman G. J., *A computational model of skill acquisition*, 1973, Massachusetts Institute of Technology.

[131]: Sussman, G. J., *The virtuous nature of bugs*, 1974, AISB summer conference.

[132]: Tate, A., *Generating project networks*, 1977, Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Pg: 888-893.

[133]: Tate, A. and Drabble, B. and Kirby, R., *O-Plan2: An Open Architecture for Command, Planning and Control*, 1992, Morgan Kaufmann, Intelligent scheduling.

[134]: Veloso M. and Carbonell J. and Pérez A. and Borrajo D. and Fink E. and Blythe J., *Integrating planning and learning: The PRODIGY architecture*, 1995, Journal of Experimental and Theoretical Artificial Intelligence,1-7.

[135]: Warren, D., *Generating conditional plan and programs*, 1976, University of Edinburgh, Pg: 344-354.

[136]: Weld, D., *An introduction to least commitment planning*, 1994, AI Magazine,15-4, Pg: 27, 61.

[137]: Weld, D.S. and Anderson, C.R. and Smith, D.E., *Extending Graphplan to Handle Uncertainty and Sensing Actions*, 1998, Proceedings of the 15th National Conference on AI, Pg: 897-904.

[138]: Yang Q. and Tenenberg J.D. and Faculty of Mathematics and University of Waterloo and Dept. of Computer Science, *ABTWEAK: Abstracting a Nonlinear, Least Commitment Planner*, 1990, University of Waterloo, Faculty of Mathematics.

[139]: Yang, Q., *Formalizing planning knowledge for hierarchical planning*, 1990, Blackwell Publishers, Inc. Cambridge, MA, USA, Computational Intelligence, 6-1, Pg: 12-24.

[140]: Yorke Smith, N., *Exploiting the structure of hierarchicalplans in temporal constraint propagation*, 2005, Proceedings of AAAI.

[141]: Younes, H.L.S. and Simmons, R.G., *VHPOP: Versatile Heuristic Partial Order Planner*, 2003, Journal of Artificial Intelligence Research, 20, Pg: 405-430.