



UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INFORMÁTICA DE
GESTIÓN

PROYECTO FIN DE CARRERA

**REPRESENTACIÓN GRÁFICA DE DOMINIOS DE
PLANIFICACIÓN**

Autor: Sergio Morato Villar

Tutor: Ángel García Olaya

AGRADECIMIENTOS

- Tengo que dar las gracias a mi familia y especialmente a mis padres por haber estado animando siempre desde el principio de mi aventura universitaria, y especialmente en los duros momentos pasados a lo largo de la carrera que siempre han estado animando para seguir adelante.
- A mi tutor del proyecto, Ángel, porque su tiempo dedicado en las sugerencias y problemas que me han surgido y con la rapidez tratadas es más que un agradecimiento.

RESUMEN

El presente proyecto está relacionado con el área de la informática llamada Inteligencia Artificial, y dentro de la misma con la Planificación Automática. La Planificación Automática trata de encontrar la secuencia de acciones que partiendo de un estado inicial permiten llegar a un estado final. PDDL (Planning Domain Description Language, o Lenguaje de Definición de Dominios de Planificación) se ha convertido en el lenguaje estándar para describir problemas de planificación.

El objetivo de este proyecto es crear una aplicación que facilite al usuario la definición de dominios y problemas de planificación. En concreto se requiere que la aplicación sea capaz de leer un dominio de planificación y representarlo gráficamente; pueda crear de forma gráfica un problema de planificación para un dominio de planificación ya definido y pueda leer un problema de planificación de un determinado dominio de planificación y representar el propio problema gráficamente.

ABSTRACT

This project is related to an area in Computer Science called Artificial Intelligence, and inside it, Automated Planning. Automated Planning deals with the sequence of actions that allow reaching a goal state from an initial state. The Planning Domain Definition Language (PDDL) has become the standard language to describe automated planning problems.

The main goal of this project is create an application that facilitates the user the definition of planning domains and automated planning problems. Specifically, it requires that the application would be able to read a planning domain and depict it; it could create graphically an automated planning problem for a planning domain defined before and it could read an automated planning problem of a planning domain, and depict it.

ÍNDICE GENERAL

Contenido

| | |
|---|----|
| ESTRUCTURA DE LA MEMORIA..... | 9 |
| 1. ESTADO DEL ARTE..... | 10 |
| 1.1 Introducción a la Inteligencia Artificial | 10 |
| 1.2 La Planificación Automática..... | 11 |
| 1.3 Los problemas de planificación..... | 11 |
| 1.3.1 Dominio y problema..... | 14 |
| 1.3.2 Lenguajes de especificación..... | 14 |
| 1.4 Trabajos relacionados..... | 21 |
| 2. INTRODUCCIÓN AL PROYECTO | 23 |
| 2.1 Objetivos del proyecto | 23 |
| 2.2 Descripción del proyecto..... | 24 |
| 3. ARQUITECTURA..... | 25 |
| 3.1 Planteamiento del problema y solución..... | 25 |
| 3.2 Análisis de los requisitos..... | 26 |
| 3.3 Diseño | 38 |
| 3.4 Implementación..... | 39 |
| 3.4.1 Subobjetivo crear un dominio | 41 |
| 3.4.2 Subobjetivo leer dominio | 47 |
| 3.4.3 Subobjetivo crear problema | 50 |
| 3.4.4 Subobjetivo leer problema | 53 |
| 3.4.5 Cerrar programa | 54 |
| 4. PRUEBAS..... | 55 |
| 4.1 Pruebas de creación de dominio..... | 55 |
| 4.2 Pruebas de lectura de dominio | 58 |
| 4.3 Pruebas de creación de un problema | 61 |
| 4.4 Pruebas de lectura de problema..... | 65 |
| 5. CONCLUSIONES | 68 |
| 6. FUTURAS LÍNEAS/TRABAJOS | 69 |
| 7. PLAN DE TRABAJO Y PRESUPUESTO..... | 70 |
| 7.1 Descomposición en tareas y duración | 70 |

| | |
|--|-----|
| 7.2 Presupuesto | 77 |
| 7.2.1 Recursos humanos..... | 77 |
| 7.2.2 Recursos materiales..... | 78 |
| 7.2.3 Coste total..... | 80 |
| ANEXOS..... | 81 |
| A. Manual de usuario | 81 |
| A.1 Crear un dominio..... | 81 |
| A.2 Leer un dominio | 91 |
| A.3 Crear un problema de planificación | 93 |
| A.4 Leer un problema de planificación | 100 |
| B. Manual de referencia | 102 |
| B.1. Elementos necesarios..... | 102 |
| B.2. Programa utilizado..... | 102 |
| C. Unidades utilizadas..... | 104 |
| REFERENCIAS | 106 |

ÍNDICE ILUSTRACIONES

| | |
|---|----|
| Ilustración 1 Esquema general de un problema de planificación | 13 |
| Ilustración 2 Definición del estado inicial para el dominio <i>elevators</i> en itSIMPLE..... | 21 |
| Ilustración 3 Interfaz gráfica de VLEPPO | 22 |
| Ilustración 4 Modelo en cascada | 27 |
| Ilustración 5 Diagrama de casos de uso de la aplicación | 31 |
| Ilustración 6 Diagrama de “crear un dominio” | 33 |
| Ilustración 7 Diagrama de “leer un dominio” | 34 |
| Ilustración 8 Diagrama de “crear un problema” | 35 |
| Ilustración 9 Diagrama de “leer un problema” | 36 |
| Ilustración 10 Interfaz de usuario de la aplicación..... | 39 |
| Ilustración 11 Menú que aparece al usuario..... | 40 |
| Ilustración 12 Menú tras ser pulsada la tecla "alt" | 40 |
| Ilustración 13 Las opciones vuelven a ser activadas tras ser representado el dominio leído | 49 |
| Ilustración 14 Se activan todas las opciones del menú tras mostrarse el mensaje de creación satisfactoria del problema..... | 53 |
| Ilustración 15 Descomposición y duración de tareas | 71 |
| Ilustración 16 Diagrama de Gantt 1/5 | 72 |
| Ilustración 17 Diagrama de Gantt 2/5 | 73 |
| Ilustración 18 Diagrama de Gantt 3/5 | 74 |
| Ilustración 19 Diagrama de Gantt 4/5 | 75 |
| Ilustración 20 Diagrama de Gantt 5/5 | 76 |
| Ilustración 21 Introducción del dominio “barman” | 82 |
| Ilustración 22 Ejemplo de introducción de requisitos | 82 |
| Ilustración 23 Ejemplo inserción tipos..... | 83 |
| Ilustración 24 Ejemplo de asignación de tipo a subtipo..... | 83 |
| Ilustración 25 Ejemplo de selección de tipo de la constante | 84 |

| | |
|---|----|
| Ilustración 26 Ejemplo de introducción de constante | 84 |
| Ilustración 27 Introducción de nombre de predicado | 85 |
| Ilustración 28 Ejemplo inserción tipos para predicado | 85 |
| Ilustración 29 Ejemplo de introducción de parámetros para una acción..... | 86 |
| Ilustración 30 Ejemplo de selección de predicado | 87 |
| Ilustración 31 Ejemplo de introducción de tipos para predicado de una condición..... | 87 |
| Ilustración 32 Ejemplo de selección de operador de un efecto | 88 |
| Ilustración 33 Ejemplo de selección de condiciones..... | 88 |
| Ilustración 34 Pantalla que comunica al usuario qué debe hacer si quiere introducir un <i>fluent</i> .. | 89 |
| Ilustración 35 Ejemplo de selección de <i>fluent</i> | 89 |
| Ilustración 36 Ejemplo de introducción de valor para la función | 90 |
| Ilustración 37 Mensaje de creación del dominio y del fichero satisfactoriamente..... | 90 |
| Ilustración 38 Lectura de dominio: Muestra del nuevo directorio tras haber cambiado de carpeta | 91 |
| Ilustración 39 Lectura de dominio: “Edit” donde el usuario introducirá el dominio | 91 |
| Ilustración 40 Lectura de dominio: El usuario pulsando sobre el dominio también lo puede añadir..... | 92 |
| Ilustración 41 Lectura de dominio: La leyenda se muestra en la izquierda y el dominio en la parte central derecha..... | 92 |
| Ilustración 42 Creación de problema: Representación de los tipos seleccionados | 93 |
| Ilustración 43 Creación de problema: Muestra de acciones..... | 94 |
| Ilustración 44 Creación de problema: CheckBox de los objetos seleccionados | 95 |
| Ilustración 45 Creación de problema: Resultado de pulsar el botón para borrar | 95 |
| Ilustración 46 Creación de problema: Muestra de predicados y funciones del dominio..... | 96 |
| Ilustración 47 Creación de problema: Situación pulsar sobre el predicado "passenger-at" | 97 |
| Ilustración 48 Creación de problema: Situación al pulsar el botón de borrar un predicado o una función..... | 98 |
| Ilustración 49 Creación de problema: Situación tras haber pulsado el botón del estado meta y haber seleccionado el predicado "lift-at"..... | 98 |

| | |
|---|-----|
| Ilustración 50 Creación de problema: Botón con la etiqueta "Click to finish" tras haber insertado el predicado "lift-at" | 99 |
| Ilustración 51 Creación de problema: Mensaje de creación satisfactoria del problema | 100 |
| Ilustración 52 Lectura de problema: En la parte izquierda se muestra la leyenda y en la parte central-derecha el problema | 101 |

ÍNDICE TABLAS

| | |
|---|----|
| Tabla 1 Requisitos del lenguaje PDDL 2.1 y funciones de los mismos | 18 |
| Tabla 2 Requisitos funcionales | 29 |
| Tabla 3 Requisitos no funcionales | 30 |
| Tabla 4 Caso de uso “crear un dominio” extendido | 33 |
| Tabla 5 Caso de uso “leer un dominio” extendido | 34 |
| Tabla 6 Caso de uso “crear un problema” extendido | 36 |
| Tabla 7 Caso de uso “leer un problema” extendido | 37 |
| Tabla 8 Campos de la lista de tipos o constantes y finalidad de los mismos | 43 |
| Tabla 9 Campos de la lista de predicados y funciones y finalidad de los mismos | 44 |
| Tabla 10 Campos de la lista de acciones y finalidad de los mismos | 44 |
| Tabla 11 Campos del registro <i>TPrec</i> y funcionalidad de los mismos | 45 |
| Tabla 12 Campos de la lista <i>TLista_Operador</i> y funcionalidad de los mismos..... | 46 |
| Tabla 13 Campos del registro <i>tipo_fluent</i> y funcionalidad de los mismos..... | 46 |
| Tabla 14 Campos de <i>TLista_andor</i> y funcionalidad de los mismos | 47 |
| Tabla 15 Pruebas de creación de dominio..... | 58 |
| Tabla 16 Pruebas de lectura de dominio | 61 |
| Tabla 17 Pruebas de creación de problema | 64 |
| Tabla 18 Pruebas de lectura de problema..... | 67 |
| Tabla 19 Horas empleadas por tarea | 70 |
| Tabla 20 Bases de cotización 2010, 2011 y 2012 | 78 |
| Tabla 21 Coste de recursos humanos | 78 |
| Tabla 22 Coste de los equipos..... | 79 |
| Tabla 23 Coste del material fungible | 79 |
| Tabla 24 Suma del coste de equipos con el material fungible | 80 |
| Tabla 25 Coste total del proyecto..... | 80 |

ESTRUCTURA DE LA MEMORIA

En este apartado se describe la estructura de la memoria:

- “*Estado del arte*”: En este capítulo se procede a la explicación de los términos y ámbitos para llevar a cabo la comprensión del proyecto, profundizando en los más importantes. Para facilitar su comprensión se han introducido ejemplos sencillos de la vida real que permiten entenderlos con mayor claridad.
- “*Introducción al proyecto*”: En este capítulo se realiza una introducción sobre el proyecto y se presentan los objetivos del mismo.
- “*Arquitectura*”: En este capítulo se describen las partes en las que ha sido dividido el proyecto.
- “*Conclusiones*”: En este capítulo se presentan las conclusiones obtenidas en el proyecto.
- “*Futuras líneas de trabajo*”: Este capítulo muestra las posibles evoluciones del proyecto para conseguir una mejora del mismo.
- “*Plan de trabajo y presupuesto*”: En este capítulo se muestran las tareas que componen el proyecto así como su duración, además de mostrar el presupuesto del mismo.
- “*Anexos*”: En este capítulo se detallan tanto el manual de usuario como el de referencia.
- “*Referencias*”: Este capítulo detalla las fuentes de información utilizadas para la realización del proyecto.

1. ESTADO DEL ARTE

En este capítulo se explicarán los términos y ámbitos necesarios para poder llevar a cabo la comprensión del proyecto.

1.1 Introducción a la Inteligencia Artificial

La Inteligencia Artificial se puede definir como la rama de la informática que se encarga de investigar el conjunto de técnicas, métodos, herramientas y metodologías que ayudan a construir sistemas que se comportan igual que un humano en la resolución de problemas concretos. Además, ha sido una de las áreas dentro del campo de la informática que más ha hecho evolucionar los problemas que pueden ser resueltos a través de las computadoras [1].

El objetivo de la Inteligencia Artificial consiste en la construcción de sistemas, tanto hardware como software, que sean capaces de replicar aspectos de lo que se suele considerar “inteligencia”. Partiendo de la definición anterior, la Inteligencia Artificial involucra a muchos campos de investigación y desarrollo diferentes, tales como la Robótica, los Sistemas Expertos, la Resolución de Problemas, etc.

Ya que el conocimiento que poseen las personas sobre un determinado campo no suele estar estructurado según los mecanismos de representación más comúnmente utilizados en la programación convencional (como pilas, listas, variables, etc.), se necesitará un conjunto de formalismos de representación que permitan representar las estructuras más complejas que emplean los humanos. En consecuencia, uno de los campos de estudio fundamentales dentro de la Inteligencia Artificial es la representación del conocimiento de manera que pueda ser fácilmente manipulado para obtener una solución.

Una vez que se haya representado ese conocimiento gracias a estos formalismos, se pasa a resolverlos. Para ello, en la Inteligencia Artificial clásica será necesario iniciar un proceso de búsqueda para lograr esa resolución, que será encontrar un objetivo determinado. Normalmente, supone alcanzar un estado final a partir de un estado inicial a través de un espacio de estados, que puede ser representado como un árbol en el que cada uno de los nodos equivale a uno de esos estados.

1.2 La Planificación Automática

La planificación puede definirse como el proceso de búsqueda y articulación de una secuencia de acciones que van a permitir alcanzar un determinado objetivo partiendo de un estado inicial. Por tanto, un plan es la secuencia de acciones que ha permitido conseguir dicho objetivo.

Se puede decir que los estudios sobre este campo todavía se encuentran en una etapa inicial, ya que el estudio de esta área no supera las cuatro décadas, aunque en los últimos años cada vez son más los sistemas que se desarrollan. Precisamente este interés por esta área, y concretamente por los problemas y soluciones que se abordan, es debido a las aplicaciones directas en campos tan importantes y llamativos como [2]:

- Robótica: Es posible planificar el movimiento de robots, ya se trate del desplazamiento del propio robot, o del movimiento de sus extremidades articuladas.
- Logística: Analizar el transporte de objetos y personas entre diferentes localizaciones.
- Simulación: Para el diseño de agentes inteligentes, en juegos y otras aplicaciones destinadas al entretenimiento.
- Espacio: En misiones de exploración espaciales mediante el control de aeronaves y vehículos no tripulados.
- Redes de comunicaciones: Tanto en la realización de consultas y servicios web, como en la organización de redes de ordenadores.
- Militar: En tareas militares de logística y estrategia.

1.3 Los problemas de planificación

Se debe entender el concepto de espacio de estados para comprender el esquema de un problema de planificación. El espacio de estados es una técnica de representación muy utilizada en la Inteligencia Artificial que permite formalizar problemas. Un espacio de estados se define en función de dos elementos:

- El conjunto de estados. Representa el mundo por el que la técnica de búsqueda va a moverse para intentar resolver los problemas.
- El conjunto de operadores. Representan la forma de moverse por ese conjunto de estados. En el caso de la Planificación Automática los operadores son las acciones que nos permiten pasar de un estado a otro y que forman el plan.

Para formalizar un problema de planificación históricamente se han usado dos notaciones principales (en ambas se utiliza la lógica de predicados para representar los estados, acciones y objetivos): ADL y STRIPS. En ambas, para aplicar una acción se deben cumplir unas precondiciones y, tras haber ejecutado la acción, habrán provocado unos efectos. En el apartado “Lenguaje PDDL” de la memoria se mostrará un ejemplo de la estructura de un fichero que contenga un problema de planificación. Además, ambas notaciones representan por un lado el dominio (predicados que pueden conformar un estado y acciones) y por otro el problema (estado inicial y metas), en forma de predicados instanciados.

Para obtener la secuencia de acciones (el resultado del problema de planificación o plan), existen varios enfoques, donde el más utilizado son las búsquedas en el espacio de estado. Dentro de este tipo de búsqueda se pueden diferenciar:

- No informadas: Se realizan en los espacios de estado donde no se dispone de información de por dónde se debe realizar la búsqueda. Como posibles técnicas se encuentran las búsquedas de estado que operan hacia adelante (la búsqueda en amplitud, la búsqueda en profundidad, etc.) y las que operan hacia atrás, partiendo del objetivo (búsqueda hacia atrás).
- Heurísticas: En la mayoría de espacios de estado existe información que permite guiar los procesos de búsqueda. Esta información no es perfecta y no siempre acierta con el mejor camino, y se denomina heurística. A destacar, se encuentran técnicas como la de escalada o la de mejor primero.

Con todos estos conceptos, una posible definición para un problema de planificación es que es un problema de búsqueda que requiere encontrar una secuencia eficiente de acciones que conducen a un sistema, partiendo de un estado inicial, hacia un estado o estados en los que se cumplen unas metas. De una manera más formal, un problema de planificación (P) se puede definir como $P = (L, A, I, F)$ donde:

- L: Conjunto de literales que representan los hechos que tienen relevancia en el problema.
- A: Conjunto de acciones que convierten un estado en otro.
- I: El estado inicial del problema.
- F: Las metas del problema.

En lógica de predicados quedaría representado de la siguiente forma:

- L: Conjunto de todos los posibles predicados y objetos.
- A: Conjunto de todas las posibles acciones que se pueden aplicar.
- I: Conjunto de todos los predicados instanciados que describen el estado inicial.
- F: Conjunto de todos los predicados que describen las metas.

Por tanto se puede describir ahora un plan como la secuencia de acciones instanciadas que partiendo del estado inicial logran las metas.

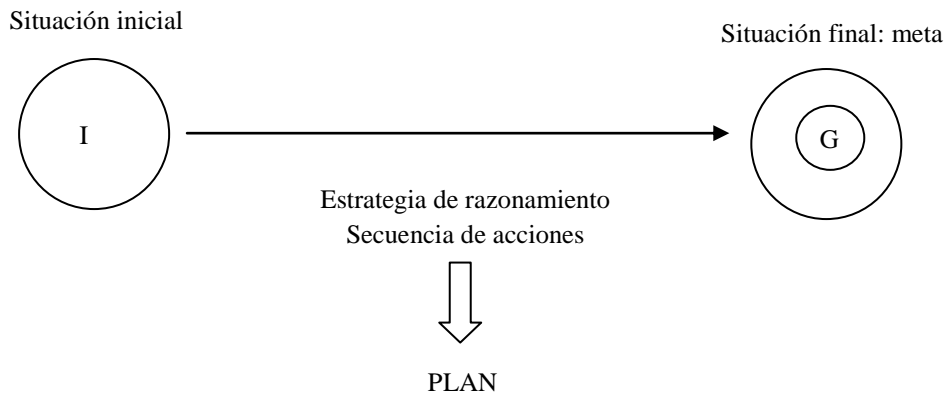


Ilustración 1 Esquema general de un problema de planificación [3]

Para comprenderlo mejor, un ejemplo de un problema de planificación puede ser este caso:

- Una persona está en Madrid, y quiere un “plan” que la permita visitar el Museo del Prado y el estadio de fútbol Santiago Bernabéu.

En el mismo, se puede identificar como estado inicial: “dónde se encuentra la persona”, “cuánto dinero posee”, “precio de los tickets”, “duración de la visita”, etc.

La meta sería la realización de las dos metas, es decir, visitar el Museo del Prado y el estadio de fútbol Santiago Bernabéu.

Como posibles acciones del mismo: “viajar por la ciudad en tren”, “comprar un ticket”, etc.

1.3.1 Dominio y problema

Para poder comprender este concepto, se puede imaginar un doble ejemplo de problema de planificación:

- Problema 1: Una persona está en Madrid, en la calle Serrano y quiere ir al Museo del Prado.
- Problema 2: Una persona está en Madrid, en la Plaza Mayor, y quiere ir al estadio de fútbol Santiago Bernabéu.

El conjunto de predicados posibles (no los instanciados) y el conjunto de objetos genéricos (que se llamarán tipos) serán los mismos en ambos problemas. El conjunto de las acciones posibles (no las instanciadas) serán igualmente las mismas en ambos problemas. Sin embargo, las metas y el estado inicial son distintos en ambos; y el mismo caso se encuentra en algunos objetos, que serán distintos, pero no los tipos (los objetos genéricos que se han mencionado anteriormente).

Por tanto se puede separar la información común (la que no depende el problema) de la información específica del problema. En consecuencia:

- Dominio: Sería la información común, es decir, la lista de los posibles predicados, tipos y acciones.
- Problema: Sería la información específica, es decir, los objetos, el estado inicial y el estado final.

1.3.2 Lenguajes de especificación

Para lograr una planificación eficiente, es tan importante tanto tener buenos algoritmos como buenos lenguajes. El lenguaje STRIPS (el cual se ha mencionado anteriormente) ha condicionado la gran mayoría de trabajos sobre planificación desde comienzos de los años 70.

Este lenguaje es muy simple y tiene algunas limitaciones, ya que resulta muy restringido para la mayoría de los dominios complejos, aunque admite un elevado grado de ampliación. Una ampliación del mismo es el lenguaje ADL, más expresivo que STRIPS y que está basado en un modelo algebraico para caracterizar los estados del mundo. Algunas de sus principales extensiones respecto a STRIPS son las precondiciones y objetivos negados, precondiciones cuantificadas, comparaciones, efectos condicionales, etc.

Lenguaje PDDL

ADL no es la única ampliación de STRIPS utilizada por los planificadores. Desde 1998 se ha venido organizando una competición de planificadores, la IPC (“International Planning Competition”). Para ella, era necesario un lenguaje para poder especificar dominios y problemas de forma común para todos los planificadores, y se creó el denominado “lenguaje de definición de dominios de planificación” (Planning Domain Definition Language: PDDL), que es otra ampliación de STRIPS, que incorpora muchos de los elementos de ADL.

PDDL se ha convertido en el lenguaje estándar para describir problemas de planificación. Entre sus principales características [3]:

- Modelo de acciones basado en STRIPS.
- Efectos condicionales y cuantificación universal, tal y como se propone en ADL.
- Especificación en las acciones jerárquicas. Las acciones se descomponen en subacciones y subobjetivos que permiten abordar problemas más complejos.
- Definición de axiomas del dominio. Los axiomas son fórmulas lógicas que establecen relaciones entre los hechos que se satisfacen en un estado (al contrario que las acciones, que definen relaciones entre sucesivos estados).
- Especificación de restricciones de seguridad. Estas restricciones permiten definir un conjunto de objetivos que deben cumplirse durante todo el proceso de planificación.

Dado el gran número de características que PDDL [4] puede expresar, prácticamente no existe ningún planificador capaz de manejarlas todas. PDDL agrupa estas características en un conjunto de requerimientos para que los planificadores puedan comprobar rápidamente si son capaces de manejar un determinado dominio.

PDDL tiene varias versiones, entre las que se puede destacar la versión PDDL 2.1, por ser la más usada. Esta versión tiene como principales características la especificación de costes de ejecución de operadores, tipos para variables de operadores, modelos no conservativos de acciones, etc.

La versión actual es PDDL 3.1. Sin embargo no existen prácticamente planificadores que soporten esta versión, por lo que este PFC se ha centrado en la versión PDDL 2.1, aunque en los casos que no sea así, se hará constar en la memoria.

La estructura de un fichero de dominio de acuerdo a este lenguaje de planificación sería la siguiente.*

```
(define (domain <name>)
  (:requirements <:req 1> ... <:req n>)
  (:types <subtype1> ... <subtype n> - <type1>
         <typen>)
  (:constants <cons1> ... <cons n>)
  (:predicates <p1> <p2> ... <p n>)
  (:functions  <f1> - number
              <f2> - number
              ...
              <f n> - number)
  (:action 1)
  ...
  (:action n)
)
```

De acuerdo a esta estructura, un posible ejemplo de fichero dominio sería:

```
(define (domain Rover)
  (:requirements :typing)
  (:types rover waypoint store camera mode lander objective)
  (:predicates (at ?x - rover ?y - waypoint)
              (at_lander ?x - lander ?y - waypoint)
              (can_traverse ?r - rover ?x - waypoint ?y - waypoint)
              )
  (:action navigate
   :parameters (?x - rover ?y - waypoint ?z - waypoint)
   :precondition (and (can_traverse ?x ?y ?z) (available ?x) (at ?x ?y)
                     (visible ?y ?z)
                    )
   :effect (and (not (at ?x ?y)) (at ?x ?z)
               )
  )
)
```

* - *number* es utilizado únicamente para la versión PDDL 3.1. En la versión 2.1 no es contemplado.

Como se puede ver, las acciones tienen precondiciones (literales que deben ser validados para que la acción se pueda aplicar) y efectos (literales que la acción añade, modifica, o borra del estado cuando se aplica). Estos literales se encuentran relacionados mediante una serie de operadores.

Hay que tener en cuenta que no se pueden utilizar los mismos operadores para la precondición de una acción que para el efecto de una acción. Los posibles operadores que se pueden utilizar en una precondición son los siguientes:

- and / or / not
- imply <cond> <effect>
 - o (imply (and (at ?r1 ?w1) (at ?r2 ?w2)) (= ?w1 ?w2))
- exists <variable> <literal>
 - o (exists (?r – rover) (at ?r ?w2))
- forall <variables> <literal>
 - o (forall (? – rover) (at ?r ?w2))

Los posibles operadores que se pueden utilizar en los efectos de una acción son los siguientes:

- and / not
- forall <variables> <effect>
- when <expression> <effect>

Los requisitos de este lenguaje de especificación junto con la función de cada uno se muestran a continuación en la siguiente tabla [5]:

| REQUISITO | FUNCIÓN |
|----------------------------|--|
| :strips | Sentencia básica de STRIPS. |
| :typing | En la declaración de variables, permite introducir tipos. |
| :negative-preconditions | Permite el operador <i>not</i> en las precondiciones. |
| :disjunctive-preconditions | Permite el operador <i>or</i> en las precondiciones. |
| :equality | Permite incorporar el operador = a los predicados. |
| :existential-preconditions | Permite el operador <i>exists</i> en las precondiciones de las acciones. |
| :universal-preconditions | Permite el operador <i>forall</i> en las precondiciones y efectos de las acciones. |

| | |
|---------------------------|--|
| :quantified-preconditions | Equivale a los requisitos <i>:existential-preconditions</i> + <i>:universal-preconditions</i> |
| :conditional-effects | Permite el operador <i>when</i> en los efectos de las acciones. |
| :adl | Equivale a los requisitos <i>:strips</i> + <i>:typing</i> + <i>:negative-preconditions</i> + <i>:disjunctive-preconditions</i> + <i>:equality</i> + <i>:quantified-preconditions</i> + <i>:conditional-effects</i> |
| :action-costs * | Permite introducir funciones (costes en las acciones). |

Tabla 1 Requisitos del lenguaje PDDL 2.1 y funciones de los mismos

Versiones posteriores de PDDL incluyen otros requisitos, que se han omitido en este proyecto. Para una lista completa y una descripción de PDDL 3.1 se puede consultar la referencia [6] de la memoria.

Como se ha mencionado en la tabla, mediante el requisito *:action-costs* es posible la introducción de funciones, también denominadas *fluents*. Estas funciones pueden ser usadas tanto en las precondiciones de las acciones como en los efectos de las mismas.

Un posible ejemplo de la definición de funciones en el fichero dominio es:

```
(:functions (fuel-level ?r - rover) - number
            (fuel-used ?r - rover) - number
            (fuel-required ?w1 ?w2 - waypoint) - number
            (total-fuel-used) - number)
```

Las funciones se usan:

- En las precondiciones: Van acompañadas de los operadores relacionales (=, >, <, <=, >=).

Un ejemplo:

```
(< (total-fuel-used) 10)
```

- En los efectos: Los valores son modificados mediante *increase*, *decrease*, *assign*, *scale-up*, *scale-down*.

Un ejemplo:

```
(increase (fuel-used ?r) 8) **
```

* Aunque en la versión original de PDDL 2.1 el nombre de esta etiqueta es *:fluents*, se ha puesto la de la versión 3.1 por ser la utilizada por los planificadores modernos.

** Hay pocos planificadores que manejen precondiciones numéricas.

Un ejemplo de acción utilizando funciones sería:

```
(:action drive
:parameters (?t - truck ?from ?to - place)
:precondition (and (at ?t ?from) (< (fuel-needed ?to ?from) (fuel-level ?t))
:effect (and (not (at ?t ?from)) (at ?t ?to)
          (increase (total-cost) (drive-cost ?from ?to))
          (decrease (fuel-level ?t) (fuel-needed ?to ?from))))
```

La estructura de un fichero problema sería la siguiente:

```
(define (problem <name>)
  (:domain <name>)
  (:objects
    <obj1> - <tipo>
    <obj2> - <tipo>
    ...
    <obj n> - <tipo>)
  (:init
    (<pred1>)
    (<pred2>)
    ...
    (<pred n>)
    (<func1>)
    ...
    (<func n>))
  (:goal (and
    (<pred1>)
    (<pred2>)
    ...
    (<pred n>)
    (<func1>)
    ...
    (<func n>)))
  (:metric <met>))
```

Así, un ejemplo de fichero que contenga un problema de planificación sería:

```
(define (problem roverprob1234)
  (:domain Rover)
  (:objects
    general - Lander
    colour high_res low_res - Mode
    rover0 - Rover
    rover0store - Store
    waypoint0 waypoint1 waypoint2 waypoint3 - Waypoint
    ... )
  (:init
    (= (total-cost) 0)
    (visible waypoint1 waypoint0)
    (visible waypoint0 waypoint1)
    (visible waypoint2 waypoint0)
    (visible waypoint0 waypoint2)
    (visible waypoint2 waypoint1)
    (visible waypoint1 waypoint2)
    ...)
  (:goal (and
    (communicated_soil_data waypoint2)
    (communicated_rock_data waypoint3)
    (communicated_image_data objective1 high_res)))
  (:metric minimize (total-cost))
)
```

1.4 Trabajos relacionados

No existen muchas herramientas para la representación gráfica de dominios y problemas de planificación como la que se ha realizado en este proyecto fin de carrera. La más conocida es itSIMPLE [13]. itSIMPLE es una herramienta muy potente para el diseño de dominios de planificación y presenta un entorno integrado de Ingeniería del Conocimiento, que permite adquisición del conocimiento, modelado del dominio, análisis y verificación del modelo, mantenimiento del conocimiento y análisis de los planes. Todo ello con una interfaz gráfica amigable y muy elaborada. En itSIMPLE la descripción del dominio se hace por medio de diagramas UML que se convierten automáticamente a código PDDL. itSIMPLE es una herramienta muy potente, pero no permite la carga directa de dominios o problemas en PDDL y tampoco la creación de estos de manera gráfica sin pasar por UML. Esto implica que para diseñar un dominio el usuario debe conocer la sintaxis de este lenguaje, lo que puede complicar su uso en la práctica. En contraste la aplicación desarrollada en este proyecto fin de carrera solamente requiere del conocimiento básico de PDDL, y en el caso de que se quiera crear un problema para un dominio ya existente, ni siquiera es necesario conocer su sintaxis. La imagen muestra un ejemplo de la definición del estado inicial para el dominio *elevators* [10].

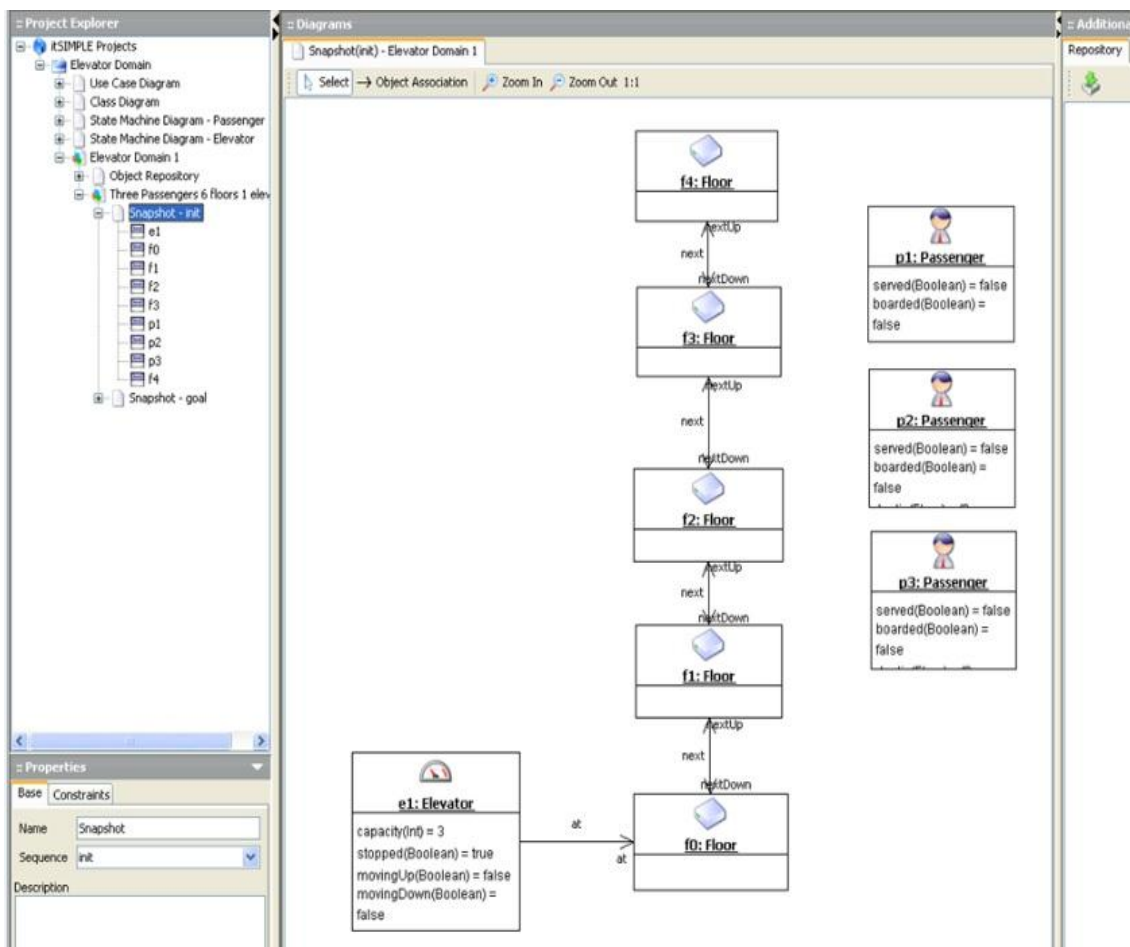


Ilustración 2 Definición del estado inicial para el dominio *elevators* en itSIMPLE

Otra herramienta que también es utilizada para la representación gráfica de dominios y problemas de planificación es VLEPPO [14]. Esta herramienta representa gráficamente tanto dominios como problemas de planificación, e incluso, se centra en servicios web para poder tener una relación con otros planificadores. Sin embargo, no permite crear un problema de planificación a partir de un dominio de planificación mediante la representación gráfica y a su vez la interfaz gráfica utilizada por esta herramienta se divide en tres partes, cada una de las cuales contiene una compleja y abundante simbología en comparación con la simplicidad realizada para este proyecto fin de carrera. A continuación se muestra una figura donde se puede observar la interfaz gráfica de VLEPPO.

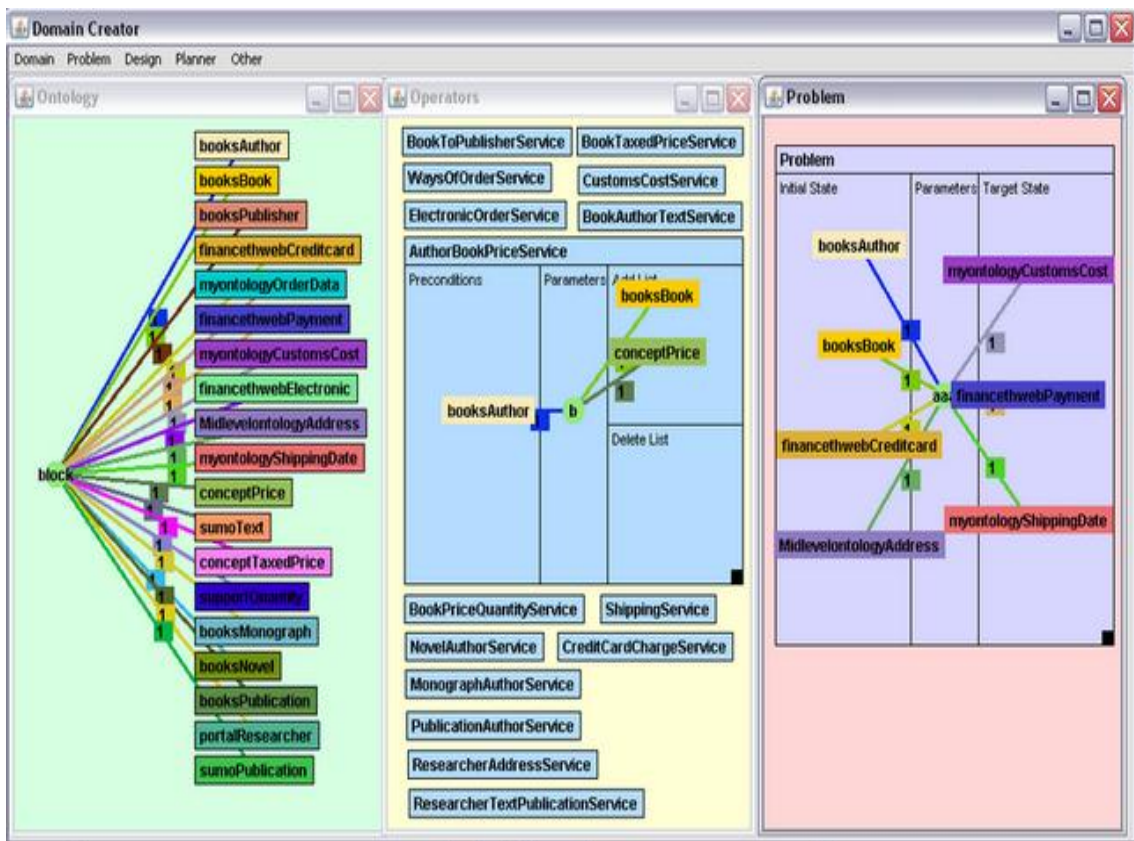


Ilustración 3 Interfaz gráfica de VLEPPO

2. INTRODUCCIÓN AL PROYECTO

Este capítulo contiene una introducción al proyecto. En primer lugar se introducirán los objetivos del proyecto, y posteriormente se explicará de forma general cómo ha sido dividido el proyecto en etapas para llevar a cabo correctamente su realización.

2.1 Objetivos del proyecto

Como se ha visto anteriormente, los ficheros formados por dominios de planificación contienen objetos, predicados y acciones que se pueden realizar sobre dichos objetos. Estos ficheros contienen esta información mediante la sintaxis empleada por un lenguaje de especificación, en este caso concreto el lenguaje PDDL. Estos ficheros son de texto y por consiguiente no muestran una representación gráfica para poder facilitar al usuario la comprensión de los mismos.

Por tanto, el objetivo principal del proyecto es crear una aplicación que permita representar gráficamente dominios y problemas de planificación. Este objetivo general está formado por cuatro subobjetivos más específicos, los cuales se detallan a continuación:

- *Crear un fichero que contenga un dominio de planificación:* Partiendo desde cero, la aplicación deberá ser capaz de crear un fichero que contenga el dominio de planificación con toda la información sobre el mismo que desee el usuario.
- *Leer un fichero que contenga un dominio de planificación y representarlo gráficamente:* A partir de un fichero que contenga un dominio de planificación, la aplicación deberá ser capaz de almacenar toda la información del mismo, y a su vez, representarla gráficamente.
- *Crear un fichero que contenga un problema de planificación:* La aplicación deberá ser capaz de crear un fichero que contenga un problema de planificación creado a partir de un dominio de planificación. Durante la creación del mismo, se deberá representar gráficamente tanto los objetos que formarán parte del problema, como el estado inicial y el estado final del mismo.
- *Leer un fichero que contenga un problema de planificación y representarlo gráficamente:* Partiendo de un fichero que contenga un problema de planificación para un determinado dominio, la aplicación deberá ser capaz de almacenar toda la información del problema, así como hacer una representación gráfica de la misma.

2.2 Descripción del proyecto

El desarrollo de este proyecto se ha producido en las siguientes fases o etapas:

1. Planteamiento del problema y solución: Es la primera fase del proyecto. En ella se ha procedido al estudio del problema, es decir, se ha procedido a entender los conceptos con los cuáles se va a trabajar para desarrollar el mismo, tales como planificación, lenguaje PDDL, etc., y además se ha elegido el lenguaje de implementación de la aplicación, así como la plataforma de desarrollo, que se explicará en el capítulo B.2 de esta memoria. Se ha estudiado la carencia de representación gráfica por parte de un dominio de planificación, y se ha considerado el implantar una representación gráfica para representar a los mismos, además de mejorar la creación y representación de un problema de planificación a partir de un dominio determinado mediante la ayuda gráfica.
2. Análisis de requisitos de usuario y diseño: Mediante reuniones con el tutor del proyecto se han recogido los requisitos de usuario que debe satisfacer la aplicación. Una vez que se han obtenido los requisitos de usuario, se ha pasado a un análisis y ordenación de los mismos para facilitar el desarrollo de las posteriores fases.
3. Implementación: En esta fase se ha generado el código fuente y los elementos necesarios para el correcto funcionamiento de la aplicación. Esta fase fue la más duradera debido a que se tuvo que cambiar de plataforma a la hora de realizar el código, es decir, de estar trabajando en el entorno Delphi se pasó al entorno *Lazarus* (en el apartado 5 de la memoria se explicará el motivo).
4. Pruebas: En esta fase se han realizado pruebas de acuerdo con los requisitos del sistema para comprobar el cumplimiento de los mismos y el correcto funcionamiento de la aplicación. Han sido llevadas a cabo por el creador del proyecto y el tutor, además de los padres del propio creador del proyecto los cuales se han considerado como usuarios sin experiencia en tal campo de la informática y por tanto de la aplicación.

3. ARQUITECTURA

En este capítulo se describen las fases seguidas en la realización de la aplicación. Tal y como se mencionó en el apartado “2.2 descripción del proyecto”, la primera fase sería el planteamiento del problema y solución al mismo, seguida del análisis de requisitos. La tercera etapa sería la de diseño, y la cuarta de implementación.

3.1 Planteamiento del problema y solución

Este apartado corresponde a la primera fase del proyecto. En ella se plantea el problema del que trata el proyecto y las soluciones que se han propuesto para resolver el mismo.

A través de las reuniones mantenidas con el tutor y el estudio de la documentación, se ha llevado a cabo la determinación del problema que se presenta con la realización de este proyecto.

El problema consiste en la dificultad de encontrar los dominios de los problemas de planificación representados gráficamente, ya que sólo se encuentran almacenados en ficheros mediante un lenguaje de especificación, en este caso el lenguaje mencionado previamente denominado “PDDL”. Si estuvieran representados gráficamente sería una importante ayuda al usuario para trabajar con los mismos, ya que para grandes dominios almacenados en ficheros, el usuario no se puede sentir tan cómodo como si los tuviera representados gráficamente.

A su vez, un problema de planificación se encuentra con la misma carencia que un dominio de planificación; sí que está almacenado correctamente en un fichero, pero sin embargo no hay representación gráfica del mismo para poder permitir al usuario una mayor comodidad a la hora de trabajar.

Pero no son los únicos problemas encontrados, ya que está la dificultad de encontrar cómodo el poder crear un problema de planificación. Es necesario partir de un dominio para poder crearlo, y sin embargo es muy difícil encontrar una aplicación gráfica que permita construir un problema de planificación partiendo de un dominio de planificación determinado y que ambos estén representados gráficamente durante su creación.

Partiendo de estos problemas, se puede añadir que no es posible encontrar una misma aplicación que englobe el ser capaz de realizar ciertas tareas como crear o leer un dominio de planificación, crear un fichero que contenga un problema de planificación a partir de un dominio de planificación o leer un fichero que contenga un problema de planificación, todo ello, mediante representaciones gráficas.

Se ha de tener en cuenta también que el usuario final de la aplicación será un usuario genérico, es decir, esta aplicación podrá ser utilizada por cualquier usuario con un mínimo conocimiento sobre dominios y problemas de planificación, no tiene por qué ser un experto.

En consecuencia es necesaria la creación de una aplicación que resuelva estos problemas surgidos con la mayor simplicidad, sencillez, fiabilidad y eficacia posible para conseguir la máxima satisfacción por parte del usuario final.

3.2 Análisis de los requisitos

La recogida de requisitos del usuario se ha producido mediante distintas reuniones con el tutor. Hay que recordar que el objetivo de este proyecto se ha subdividido en cuatro subobjetivos:

- *Crear un fichero que contenga un dominio de planificación.*
- *Leer un fichero que contenga un dominio de planificación y representarlo gráficamente.*
- *Crear un fichero que contenga un problema de planificación, a partir de un determinado dominio de planificación, mediante representaciones gráficas.*
- *Leer un fichero que contenga un problema de planificación y representarlo gráficamente.*

Se ha considerado que lo adecuado sería que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior, por tanto se ha elegido el modelo en cascada para definir el ciclo de vida del proyecto.

El modelo en cascada [7] es un modelo que admite la posibilidad de hacer iteraciones, es decir, si se tiene que volver a una de las etapas anteriores al mantenimiento, hay que recorrer de nuevo el resto de etapas. Este modelo presenta algunas ventajas tales como:

- La planificación es sencilla.
- La calidad del producto resultante es alta.
- Permite trabajar con personal poco cualificado.

En la siguiente ilustración queda reflejado el modelo cascada:

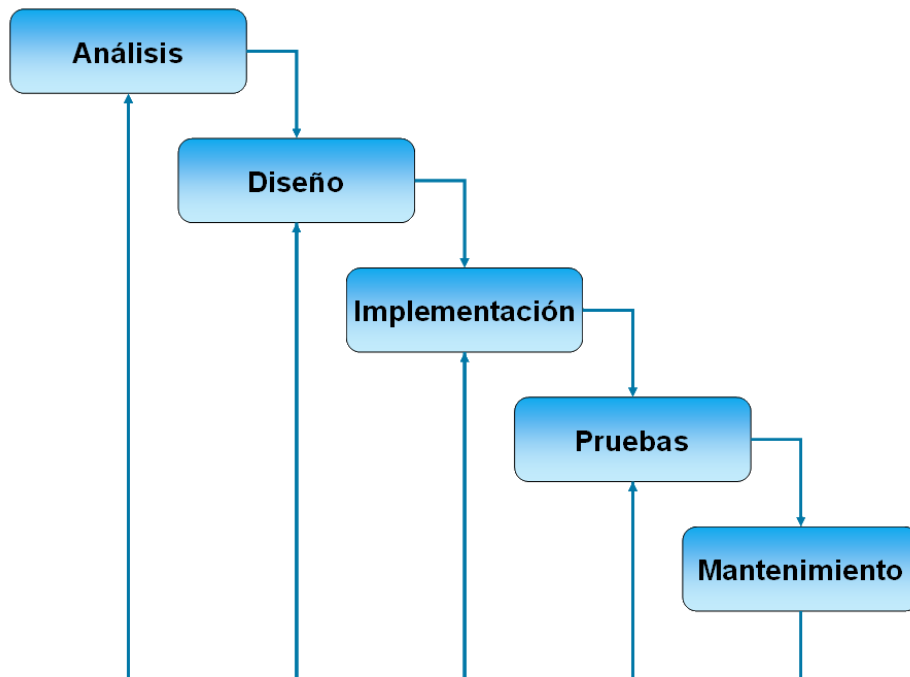


Ilustración 4 Modelo en cascada

El objetivo del análisis de requisitos es analizar y documentar las necesidades funcionales que deberán ser soportadas por la aplicación a desarrollar y obtener un conjunto de requisitos que debe cumplir el software de la aplicación. Estos se dividen en dos grupos:

- Requisitos funcionales: aquellos que definen el comportamiento interno del software.
- Requisitos no funcionales: aquellos que especifican criterios que pueden usarse para juzgar la operación de un sistema.

A continuación, se detallan los requisitos que debe cumplir la aplicación, obtenidos como se ha dicho anteriormente de las reuniones mantenidas con el tutor del proyecto.

Los requisitos funcionales de este proyecto son los siguientes:

| ID | Tipo | Descripción | Comentarios |
|-----------|----------------------|--|---|
| 1 | Entrada al sistema | El acceso de los usuarios a la aplicación se realizará sin necesidad de autenticación previa. | Cualquier usuario podrá acceder a la aplicación. |
| 2 | Entrada al sistema | La aplicación tendrá un único perfil de usuario. | La aplicación sólo tendrá un usuario genérico. |
| 3 | Creación del dominio | La aplicación deberá permitir insertar al usuario el nombre del dominio que desee crear y comprobar que el nombre es correcto. | En caso de la no introducción correcta del dominio, se le comunicará al usuario y se le permitirá introducir otro dominio. |
| 4 | Creación del dominio | La aplicación deberá permitir insertar al usuario los requisitos que desee entre los disponibles en PDDL 2.1 y comprobar que el usuario los introduce de forma correcta. | En caso de la no introducción correcta de los requisitos, se le comunicará al usuario y se le permitirá introducir de nuevo los requisitos. |
| 5 | Creación del dominio | La aplicación deberá permitir insertar al usuario los tipos que desee y comprobar que el usuario los introduce de forma correcta. | En caso de la no introducción correcta de los tipos, se le comunicará al usuario y se le permitirá introducir de nuevo los tipos. |
| 6 | Creación del dominio | La aplicación deberá permitir insertar al usuario las constantes que desee y comprobar que el usuario las introduce de forma correcta. | En caso de la no introducción correcta de las constantes, se le comunicará al usuario y se le permitirá introducir de nuevo las constantes. |
| 7 | Creación del dominio | La aplicación deberá permitir insertar al usuario los predicados que desee y comprobar que el usuario los introduce de forma correcta. | En caso de la no introducción correcta de los predicados, se le comunicará al usuario y se le permitirá introducir de nuevo los predicados. |
| 8 | Creación del dominio | La aplicación deberá permitir insertar al usuario las funciones que desee y comprobar que el usuario las introduce de forma correcta. | En caso de la no introducción correcta de las funciones, se le comunicará al usuario y se le permitirá introducir de nuevo las funciones. |
| 9 | Creación del dominio | La aplicación deberá permitir insertar al usuario las acciones que desee y comprobar que el usuario las introduce de forma correcta. | En caso de la no introducción correcta de las acciones, se le comunicará al usuario y se le permitirá introducir de nuevo las acciones. |
| 10 | Creación del dominio | Si se ha creado el fichero dominio correctamente, la aplicación deberá avisar al usuario de forma clara que el fichero dominio ha sido creado correctamente. | - |

| | | | |
|----|-----------------------|--|--|
| 11 | Lectura del dominio | La aplicación deberá permitir insertar al usuario el dominio y comprobar que ese dominio existe y se encuentra en un fichero en el mismo directorio. | En caso de la no introducción correcta del dominio o que el dominio no exista, se le comunicará al usuario y se le permitirá introducir de nuevo el dominio. |
| 12 | Creación del problema | La aplicación deberá permitir insertar al usuario la métrica que desee. | En caso de la no introducción correcta de la métrica, se le comunicará al usuario y se le permitirá introducir otra métrica. |
| 13 | Creación del problema | La aplicación deberá permitir insertar al usuario el nombre de fichero que desee y comprobar que ese fichero existe. | En caso de la no introducción correcta del nombre de fichero, se le comunicará al usuario y se le permitirá introducir de nuevo otro nombre de fichero. |
| 14 | Creación del problema | Si se ha creado el fichero problema correctamente, la aplicación deberá avisar al usuario de forma clara que el fichero problema ha sido creado correctamente. | - |
| 15 | Interfaz de usuario | Todos los botones de la aplicación tendrán un texto claro e intuitivo que indique su significado para el usuario. | - |
| 16 | Interfaz de usuario | La aplicación deberá tener un botón que permita volver a inicializar la misma cuando el usuario lo desee siempre que no esté abierta sólo la consola. | - |
| 17 | Interfaz de usuario | La aplicación deberá contener una opción que permita poder cerrarse en cualquier momento que el usuario lo desee. | - |
| 18 | Interfaz de usuario | La aplicación a la hora de la representación gráfica deberá tener una leyenda para que el usuario pueda identificar de forma clara sin ningún inconveniente los elementos representados. | - |

Tabla 2 Requisitos funcionales

Los requisitos no funcionales del proyecto son los siguientes:

| ID | Tipo | Descripción |
|-----------|------------------------|--|
| 1 | Funcionamiento | La aplicación deberá funcionar en cualquier PC con Windows XP/Vista/7. |
| 2 | Idioma | El inglés será el idioma de la aplicación por defecto. |
| 3 | Entrada/salida | Todos los ficheros tratados a lo largo de la aplicación tendrán la extensión “.pddl”. |
| 4 | Entrada/salida | “- object” no se podrá excluir en la sintaxis empleada a la hora de introducir los tipos. |
| 5 | Entrada/salida | “- number” no se podrá excluir en la sintaxis empleada a la hora de introducir las funciones. |
| 6 | Entrada/salida | El nombre del dominio, tipos, subtipos, constantes, predicados, funciones y acciones no puede empezar por un número. |
| 7 | Salida | El dominio que inserte el usuario siempre será almacenado por defecto en el fichero “ <i>domain.pddl</i> ”. |
| 8 | Representación gráfica | Un objeto se representará en un rectángulo de color rojo cuyo interior estará formado por el nombre del objeto en color verde. |
| 9 | Representación gráfica | Una constante se representará en un rectángulo de color rojo cuyo interior estará formado por el nombre de la constante en color oliva. |
| 10 | Representación gráfica | Un predicado se representará en un rectángulo cuyo interior estará formado por los tipos y constantes que contenga además del nombre del predicado. Tanto el nombre como las líneas del rectángulo serán de color azul. |
| 11 | Representación gráfica | Una función se representará en un rectángulo cuyo interior estará formado por los tipos, constantes o valor que contenga además del nombre del predicado. Tanto el nombre como las líneas del rectángulo serán de color verde azulado. |
| 12 | Representación gráfica | Los predicados y funciones que engloben el estado inicial del problema serán de color azul. |
| 13 | Representación gráfica | Los predicados y funciones que engloben el estado final del problema serán de color marrón. |

Tabla 3 Requisitos no funcionales

Una vez identificados y analizados los requisitos tal y como se han mencionado anteriormente es hora de presentar los casos de uso de la aplicación para describirla de una forma más clara. A su vez, con el objetivo de facilitar su comprensión, para cada caso de uso se muestra su diagrama y su caso de uso extendido.

A continuación se muestran los casos de usos de la aplicación:

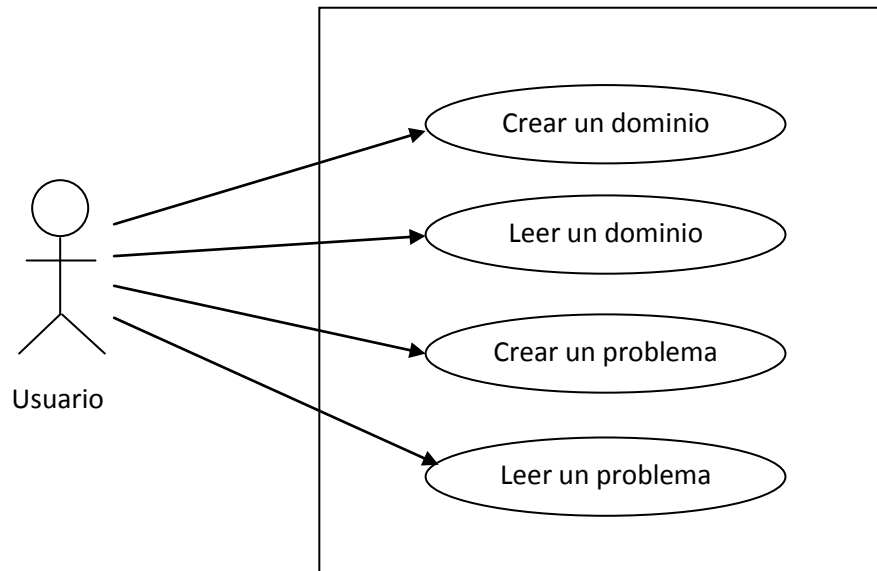
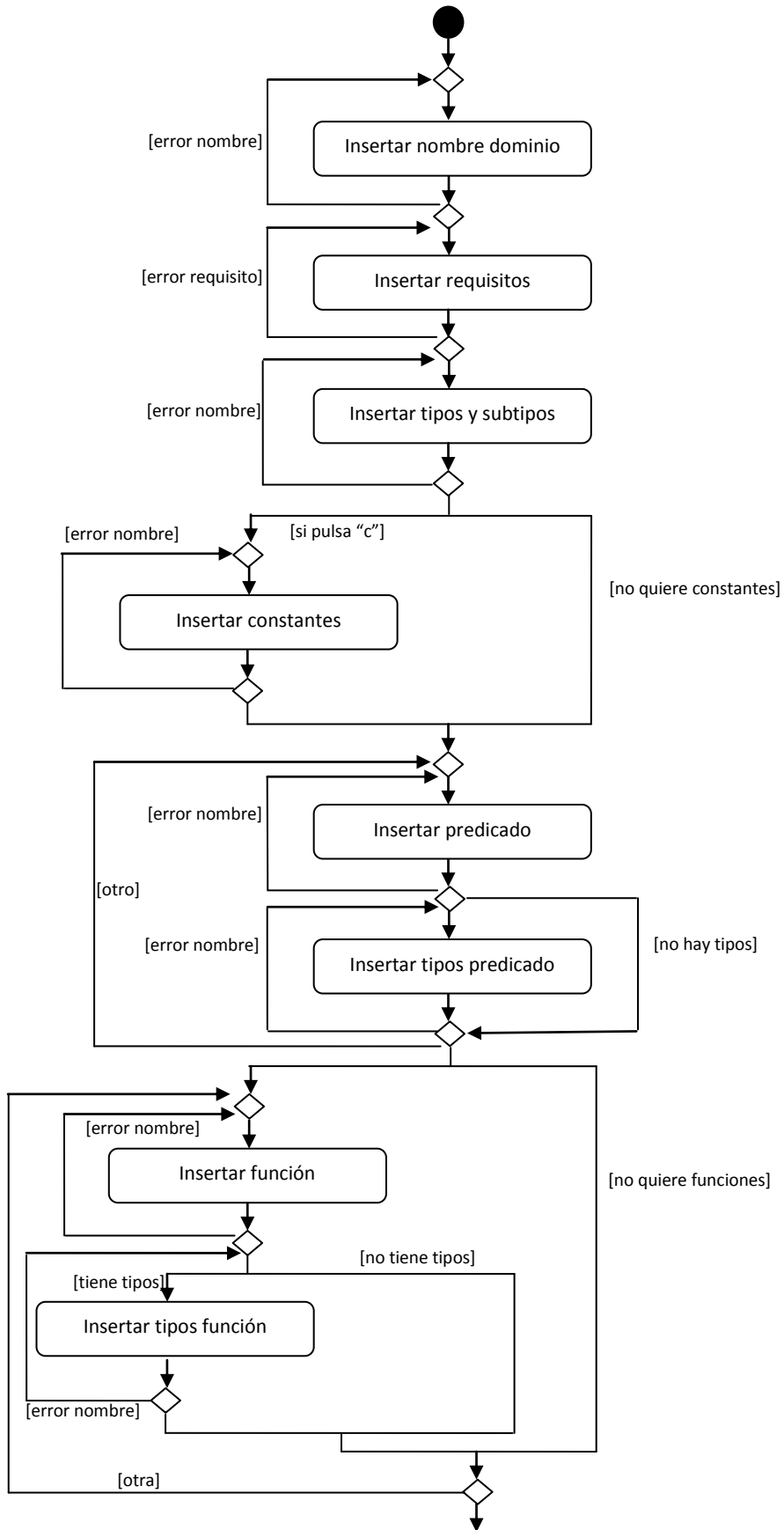


Ilustración 5 Diagrama de casos de uso de la aplicación

- Caso de uso “**crear un dominio**”: Se muestra a continuación el diagrama correspondiente y su caso de uso extendido:



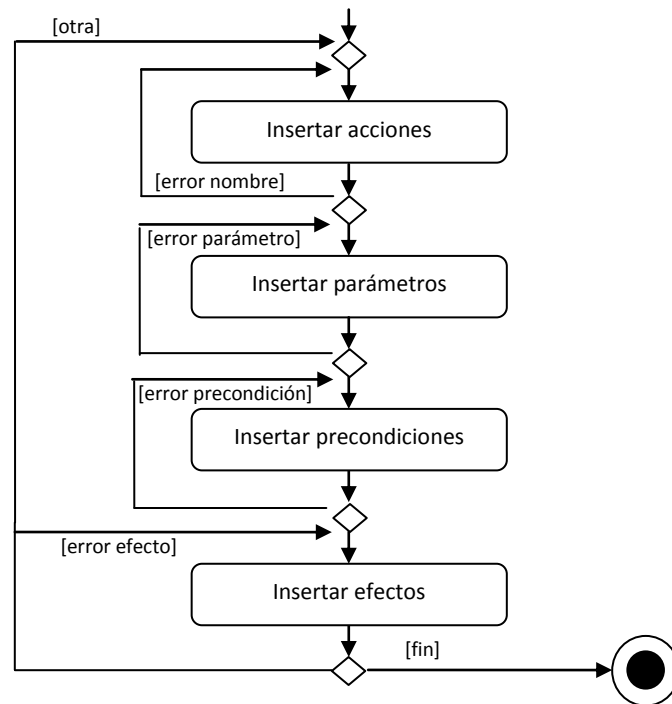


Ilustración 6 Diagrama de “crear un dominio”

| | |
|-------------------------|--|
| Caso de uso | Crear un dominio |
| Actores | Usuario |
| Objetivo | Crear un dominio determinado y almacenarlo en un fichero |
| Precondiciones | - |
| Poscondiciones | El dominio debe estar almacenado en un fichero |
| Escenario básico | Insertar nombre del dominio a crear. Insertar requisitos del dominio. Insertar tipos y subtipos. Insertar predicados y funciones Insertar tipos para predicados y funciones. Insertar acciones. Insertar parámetros de acciones. Insertar precondiciones de acciones. Insertar efectos de acciones. Escribir dominio en el fichero. |

Tabla 4 Caso de uso “crear un dominio” extendido

- Caso de uso “**leer un dominio**”: Se muestra a continuación el diagrama correspondiente y su caso de uso extendido:

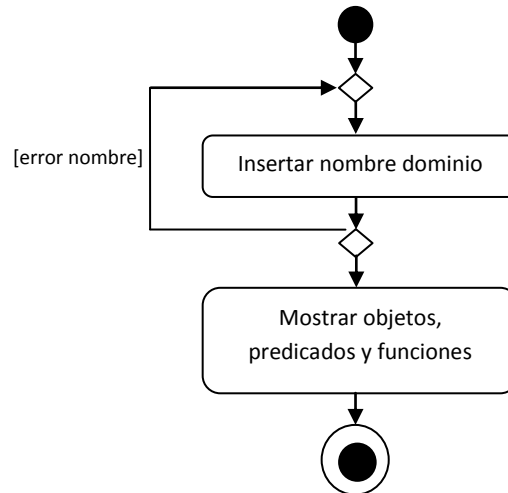


Ilustración 7 Diagrama de “leer un dominio”

| | |
|-------------------------|--|
| Caso de uso | Leer un dominio |
| Actores | Usuario |
| Objetivo | Mostrar por pantalla los objetos, predicados y funciones del dominio |
| Precondiciones | Tener un fichero donde se encuentre almacenado el dominio determinado a leer |
| Poscondiciones | Haber mostrado tanto los objetos, predicados y funciones que formaban parte de ese dominio |
| Escenario básico | Insertar el nombre del dominio a leer. Mostrar objetos, predicados y funciones. |

Tabla 5 Caso de uso “leer un dominio” extendido

- Caso de uso “**crear un problema**”: Se muestra a continuación el diagrama correspondiente y su caso de uso extendido:

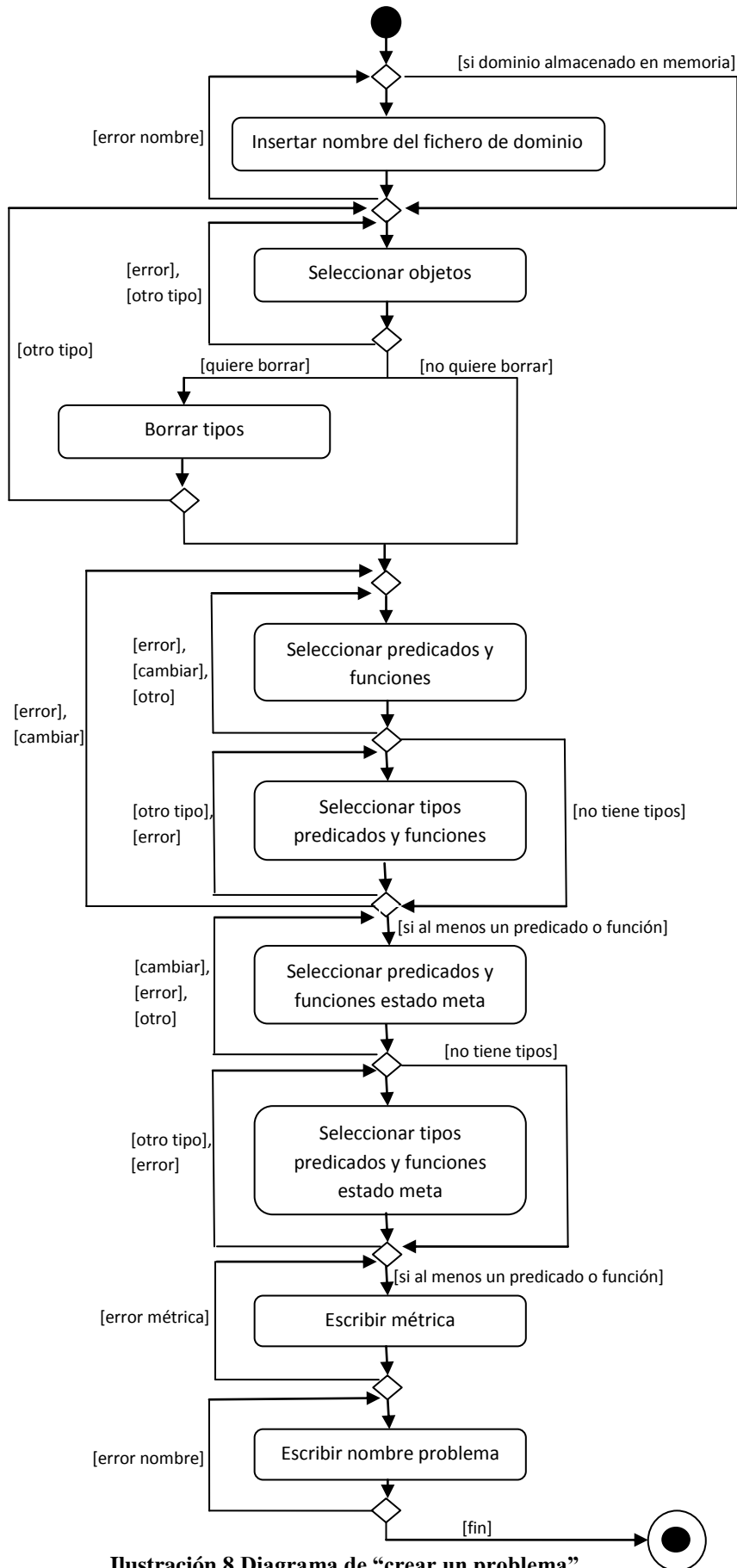


Ilustración 8 Diagrama de “crear un problema”

| | |
|-------------------------|--|
| Caso de uso | Crear un problema |
| Actores | Usuario |
| Objetivo | Crear un problema determinado y almacenarlo en un fichero |
| Precondiciones | Existencia de un fichero que contenga el dominio para el problema o tener almacenado en memoria un dominio |
| Poscondiciones | El problema debe estar almacenado en un fichero |
| Escenario básico | <p>Insertar nombre dominio (en caso de no haber un dominio almacenado en memoria).</p> <p>Seleccionar objetos para el problema.</p> <p>Seleccionar predicados y funciones para el estado inicial del problema.</p> <p>Seleccionar tipos para predicados y funciones del estado inicial.</p> <p>Seleccionar predicados y funciones para el problema en su estado meta.</p> <p>Seleccionar tipos para predicados y funciones del estado meta.</p> <p>Escribir métrica.</p> <p>Escribir nombre del fichero problema.</p> <p>Escribir el problema en el fichero.</p> |

Tabla 6 Caso de uso “crear un problema” extendido

- Caso de uso “**leer un problema**”: Se muestra a continuación el diagrama correspondiente y su caso de uso extendido:

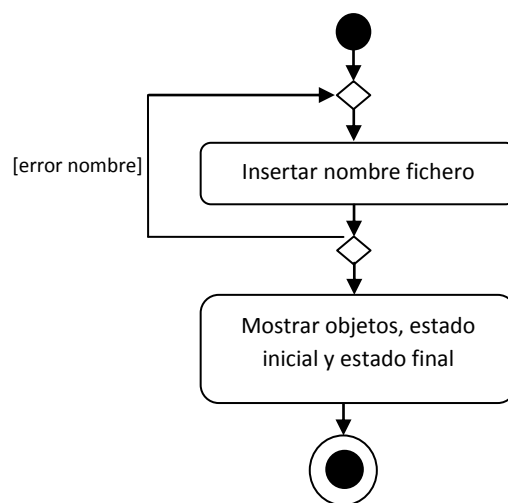


Ilustración 9 Diagrama de “leer un problema”

| | |
|-------------------------|--|
| Caso de uso | Leer un problema |
| Actores | Usuario |
| Objetivo | Mostrar por pantalla tanto los objetos, como el estado inicial y el estado final del problema. |
| Precondiciones | Tener un fichero donde se encuentre almacenado el problema determinado a leer. Tener el dominio de ese problema almacenado en un fichero. |
| Poscondiciones | Mostrar tanto los objetos, como el estado inicial como el estado final del problema. |
| Escenario básico | Insertar nombre del fichero a leer. Mostrar objetos, estado inicial y estado final. |

Tabla 7 Caso de uso “leer un problema” extendido

3.3 Diseño

Durante esta fase del trabajo se realizó el diseño de la aplicación partiendo del análisis detallado realizado en la fase anterior.

La aplicación será identificada de manera rápida y sencilla en la carpeta denominada *PFC* mediante el ejecutable de nombre *proyecto*.

Los ficheros pertenecientes a la aplicación, como se mencionó anteriormente en los requisitos, contendrán la extensión “.pddl”, y se localizarán en la carpeta *PFC*. Si el usuario desea leer ficheros de otras carpetas, se podrá cambiar el directorio por defecto de la carpeta *PFC* a lo largo de la ejecución de la aplicación.

Los objetivos de diseño son conseguir una interfaz gráfica agradable, comprensible y accesible para todo tipo de usuarios con un mínimo conocimiento en dominios y problemas de planificación.

Para el diseño de la aplicación se ha tenido en cuenta que anteriormente se dividió el objetivo general del proyecto en cuatro subobjetivos, por tanto, para facilitar al usuario su interacción con la aplicación, se ha decidido crear un menú que contiene los cuatro subobjetivos del proyecto, además de otra opción que permita al usuario salir de la aplicación siempre que lo desee. A su vez, también se decidió incluir un botón para poder volver a inicializar la aplicación siempre que lo desee el usuario mientras se encuentre trabajando con la misma, pero sólo es visible una vez que el usuario se encuentra realizando uno de los cuatro subobjetivos.

En la parte superior de la pantalla se mostrará el nombre de la aplicación, y debajo del mismo, el menú comentado anteriormente. Este menú será visible siempre, al igual que el nombre de la aplicación, y el usuario podrá acceder a cualquier opción de menú pulsando el botón correspondiente del mismo. A su vez, al lanzar la aplicación, aparte de la pantalla mencionada, también aparecerá una consola.

A continuación se muestra un ejemplo de lanzamiento de la aplicación, donde en el rectángulo blanco se mostraría el directorio desde el que se ejecuta la aplicación de consola:

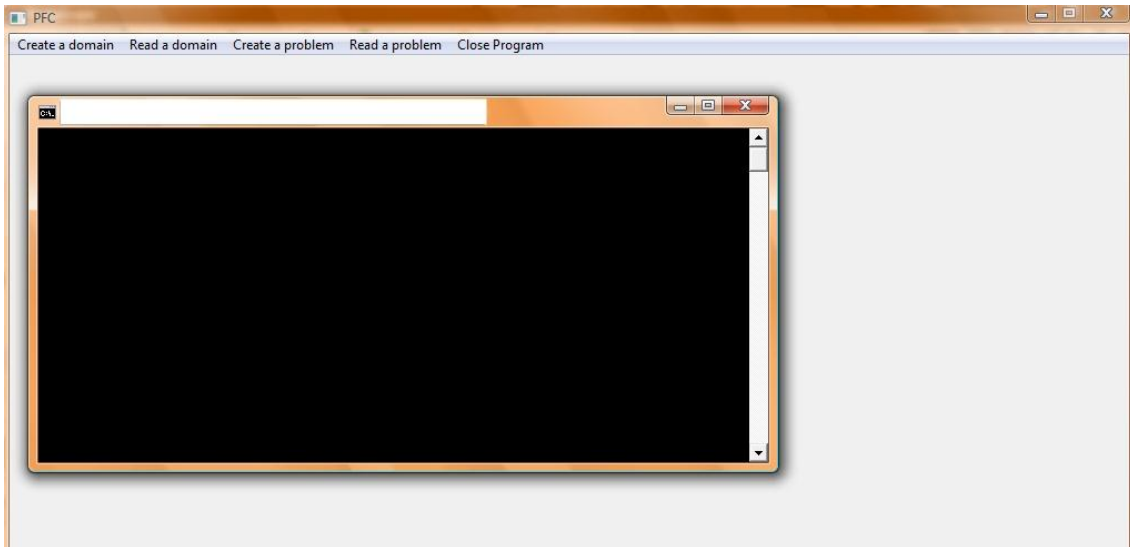


Ilustración 10 Interfaz de usuario de la aplicación

3.4 Implementación

En este apartado se explicará la fase de implementación que corresponde con el último paso en la realización de la aplicación, partiendo de todos los pasos realizados anteriormente.

Como se mencionó anteriormente, la aplicación se ha desarrollado mediante el lenguaje *Object-Pascal* en el entorno de desarrollo *Lazarus* [8], a través del cual se ha creado el ejecutable para poder lanzar la aplicación. En el mismo, se ha trabajado con una ventana de formulario (en la cual se insertan los botones y demás componentes necesarios) y con las unidades necesarias, donde se encuentra el código fuente.

El formulario utilizado se llamará *PFC*, aunque el ejecutable se denominará *proyecto*.

La metodología que ha marcado el desarrollo de la aplicación contiene los siguientes pasos:

1. Creación de un nuevo proyecto en el entorno de desarrollo *Lazarus*.
2. Creación de los elementos que formarán parte del formulario.
3. Creación de las unidades necesarias para llevar a cabo la realización del código fuente de la forma más cómoda y eficiente.
4. Escritura del código fuente.
5. Creación del ejecutable.
6. Ejecución de la aplicación.

Como se ha explicado anteriormente, el usuario podrá realizar cuatro posibles tareas en la aplicación. Además, es necesario (tal y como se indicó en los requisitos) que pueda salir de la aplicación siempre que lo desee, por tanto se ha creado un menú que permita al usuario poder elegir la opción que desee. Este menú ha sido creado en la ventana del formulario (seleccionando un “*TMainMenu*” de la paleta de componentes de *Lazarus*), en la cual se han añadido cinco campos para el mismo, mostrando el nombre que se desee mediante la propiedad “*caption*”. El usuario lo vería de la siguiente forma:

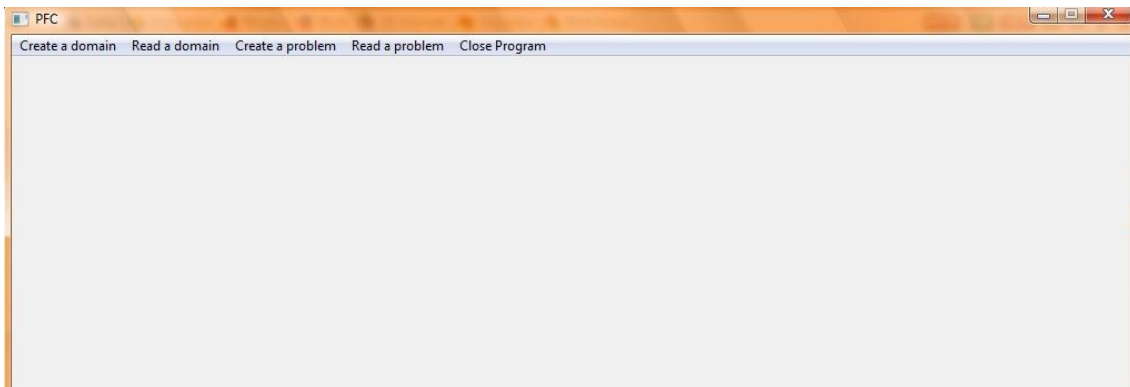


Ilustración 11 Menú que aparece al usuario

Para hacer más cómoda al usuario su posible selección, si el usuario pulsa la tecla “*alt*” se subrayará una letra de los nombres de cada opción, de tal forma que si pulsa a la vez la tecla “*alt*” + la letra correspondiente subrayada automáticamente se elige esa opción sin tener que pulsar con el ratón. Esto ha sido posible al igual que antes con la propiedad “*caption*”, de tal forma que se ha añadido “&” justo antes de la letra correspondiente. Un ejemplo sería: *Create a &domain*

En la siguiente ilustración se muestra cómo quedaría el menú tras pulsar la tecla “*alt*”:

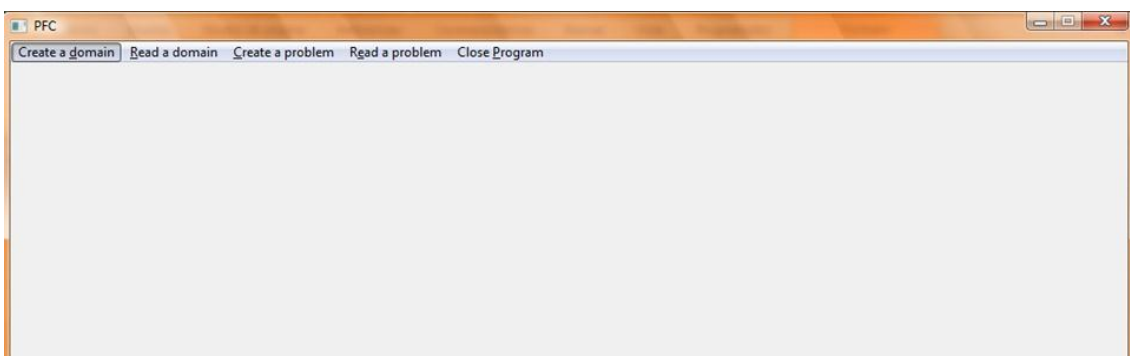


Ilustración 12 Menú tras ser pulsada la tecla "alt"

Para saber cómo es posible saber cuál opción ha seleccionado el usuario del menú, se utiliza el evento “*OnClick*”, es decir, la acción a realizar cuando se acciona el pulsador izquierdo del ratón (aunque en este caso también se accionaría pulsando la tecla “*alt*” + la letra subrayada correspondiente). Con este evento, que permite escribir

código dentro del mismo, se ha permitido diferenciar cada opción precisamente escribiendo diferentes llamadas a los procedimientos y funciones existentes que tratarán por separado las posibles acciones.

Antes de centrarnos en cada opción del menú o subobjetivo del proyecto, hay que mencionar que al inicio de cada unidad se ha tenido que añadir:

- `{$APPTYPE CONSOLE}`: No sería posible lanzar la consola sin esta sentencia.
- `{$LongStrings ON}`: Necesaria para leer cadenas con gran cantidad de caracteres incluidos.
- `{$MODE Delphi}` ó `{$mode objfpc}` y `{$H+}`: Estas sentencias son necesarias para pasar el código fuente del proyecto Delphi a Lazarus.

Una vez mencionados estos detalles, se irá describiendo de manera detallada la implementación de cada opción del menú.

3.4.1 Subobjetivo crear un dominio

En los requisitos previamente analizados se acordó que uno de los subobjetivos del proyecto era permitir la creación por parte de un usuario de un dominio de planificación determinado y almacenarlo en el fichero “*domain.pddl*”. Este subobjetivo ha sido incluido como una opción del menú y se ha denominado “*Create a domain*”. En él, es necesaria la utilización de la consola para la creación del mismo, y no el formulario general. Esto es debido a que el código para su implementación fue realizado en modo consola en “*Delphi*” antes de cambiar al entorno de desarrollo “*Lazarus*”, tal y como se explicará en el apartado 5 de la memoria, y se decidió dejar el modo consola sin ser necesaria la utilización del formulario. Por tanto se ha escrito “*PFC.visible:=false;*” que es una propiedad mediante la cual el formulario no se muestra al usuario y sólo se muestra en este caso la consola. Esta sentencia viene dentro del procedimiento *CreandoFichero* que es el que va a permitir crear el fichero donde será almacenado el dominio que inserte el usuario.

En este mismo procedimiento, y antes de empezar a pedir al usuario los datos del dominio, se abre el fichero “*domain.pddl*” mediante las siguientes sentencias:

```
assignFile(f, “domain.pddl”);  
rewrite(f);
```

La función “*pedir_dominio*” será la encargada de pedir el nombre del dominio a insertar. Para poder mostrar un mensaje por pantalla en la consola es necesario poner la sentencia “*write(‘Mensaje a mostrar’);*” o “*writeln(‘Mensaje a mostrar’);*”. La primera permite escribir a continuación, mientras que la segunda hace un salto de línea y lo que escriba el usuario será en la línea de abajo.

Para mayor comodidad del usuario, se ha cambiado el color del mensaje mediante la sentencia *textcolor()* poniendo entre los paréntesis el color a elegir. Además

se puede añadir un color de fondo mediante la sentencia *textbackground()*, y entre los paréntesis poniendo el número correspondiente a un color de los permitidos en el lenguaje *Object-Pascal*. También se ha utilizado *gotoxy(numero1, numero2)*; que permite colocar el texto en la consola en las posiciones que indiquen los números introducidos. También hay que destacar la sentencia *clrscr()*; que limpia la pantalla de la consola. A lo largo del programa se han ido añadiendo estas sentencias cada vez que se han tenido que utilizar.

Una vez introducido el nombre del dominio, la sentencia *readln(dominio)*; se encargará de leer los datos introducidos por el usuario, y se llamará al procedimiento “*escribirDominio*” para poder escribirlo en el fichero “*domain.pddl*”. Escribir en un fichero es similar a escribir por consola, pero hay que añadir el fichero, por tanto sería “*write(f, dominio);*”.

A continuación se llamará a la función “*pedir_requisitos*” que es la encargada de pedir los requisitos del dominio (mostrará una lista con los requisitos a elegir por parte del usuario, cada cual con un número). El usuario introducirá el número correspondiente al requisito que desee, y si quiere introducir más de un requisito, deberá separar esos números por comas. Si el usuario introduce la cadena vacía, entonces automáticamente se considera que sólo ha elegido el requisito “*:strips*”.

La cadena de números introducida será tratada mediante el procedimiento *leer_cadena*, que obtiene los números introducidos y los inserta en una lista. Mediante el procedimiento *insertarRequisitos* se comprobará que los números correspondan a requisitos, y si es así se almacenarán los requisitos definitivos en la lista de requisitos definitiva. Estos requisitos serán escritos en el fichero mediante el procedimiento “*escribirRequisitos*”.

Como se ha dicho, la estructura para tener almacenados los requisitos será una lista, en este caso llamada *TLista_req*.

La estructura para poder almacenar los tipos (también se utilizará para el almacenamiento de constantes, que se verá posteriormente) será una lista (*TLista_Types*) con los siguientes campos en cada nodo de la misma:

| CAMPO | FINALIDAD |
|--------------|--|
| <i>Info</i> | Almacenar el nombre del tipo de la constante, será de tipo <i>string</i> . |
| <i>Tipo</i> | Almacenar si es un tipo o una constante, será de tipo <i>string</i> . |
| <i>Iden</i> | Almacenar el identificador del tipo o constante. |
| <i>Ts</i> | Almacenar el tipo al que pertenece, será de tipo <i>string</i> . |

| | |
|---------------|--|
| <i>numero</i> | Almacenar el número a través del cual ha sido insertado. |
| <i>Sgte</i> | Puntero que permite enlazar con el siguiente nodo de la lista. |

Tabla 8 Campos de la lista de tipos o constantes y finalidad de los mismos

La función *pedir_tipos* será la encargada de obtener la cadena de tipos que introduzca el usuario. A diferencia de los requisitos, esta vez el usuario sí que tiene que introducir el nombre de los mismos, pero si quiere introducir más de un nombre el procedimiento será el mismo, separarlo por comas. La novedad, es que el usuario puede introducir subtipos. Para introducir un subtipo debe introducir el nombre del mismo, un espacio, y posteriormente un guión. El procedimiento para su inserción en la lista de tipos del sistema será *insertarTypes*, que se encargará de diferenciar los tipos de los subtipos mediante el campo de los mismos (“*ts*”), y en caso de ser un subtipo se llamará a la función *pedirSubtipo* para obtener el tipo al que pertenece. Mientras que *escribirTypes* será el encargado de escribir los tipos en el fichero, teniendo en cuenta que si es un tipo hay que escribir “- object” (es un requisito del sistema) y si es un subtipo “- tipo al que pertenezca”.

Mediante la función booleana *puedoContinuar* se comprueba si el usuario puede introducir constantes de acuerdo con los requisitos que ha introducido previamente. En caso afirmativo, la función *quiereConstantes* se encarga mediante la sentencia “*readKey()*;” si la tecla introducida corresponde a la letra “*c*”, en cuyo caso el usuario sí querría introducir constantes. En primer lugar se introducirá el tipo al que pertenezcan la o las constantes que se quieran introducir, para posteriormente introducirla/s. De todo ello se encargará el procedimiento *insertarConstantes*, mientras que *escribirConstants* será el encargado de escribirlas en el fichero “*domain.pddl*”. La estructura para almacenar las mismas será la de tabla 8 de la memoria.

La estructura para poder almacenar los predicados y funciones (*TPuntero_PredFunct*) es una lista donde cada nodo tendrá los siguientes campos:

| CAMPO | FINALIDAD |
|---------------|---|
| <i>nombre</i> | Almacenar el nombre del predicado o de la función, será de tipo <i>string</i> . |
| <i>Tipo</i> | Almacenar si es un predicado o función, será de tipo <i>string</i> . |
| <i>Valor</i> | Almacenar el valor de un predicado o de una función, será de tipo entero. |
| <i>Types</i> | Lista que se encarga de almacenar los tipos de un predicado o función. |

| | |
|-------------|--|
| <i>Sgte</i> | Puntero que permite enlazar con el siguiente nodo de la lista de predicados o funciones. |
|-------------|--|

Tabla 9 Campos de la lista de predicados y funciones y finalidad de los mismos

Para que el usuario pueda introducir los predicados, se ha creado un bucle. De él no se podrá salir hasta que el usuario introduce un nombre vacío, o ya ha introducido correctamente al menos un predicado. Por otra parte, una vez que se introduce un nombre, siempre se comprueba que no haya sido introducido anteriormente, además de que el mismo no empiece por un número, lo cual tampoco se permite. De este proceso se encarga la función *PredicadoPermitido*. Una vez comprobado que es correcto, se pasa a la introducción de los tipos para el mismo, tal y como se explicó anteriormente en la introducción de tipos, con la diferencia de que ahora se tienen que insertar tipos ya existentes. Cuando se hayan insertado los tipos en la lista mediante el procedimiento *insertarObjetos* (hay que recordar que puede haber predicados sin tipos), el procedimiento *insertarPredicado* se encargará de insertar el predicado y sus tipos (en caso de haberlos) en la lista de predicados destinada al almacenamiento de los mismos en la aplicación. Para la escritura de los mismos en el fichero se llamará al procedimiento *escribirPF*.

El procedimiento para la inserción de funciones, en caso de ser posible debido a los requisitos, es igual al de los predicados y la estructura de almacenamiento será la misma, mostrada en la tabla 9 de la memoria.

La estructura para poder almacenar las acciones ha sido una lista (*TPuntero_Acciones*), donde los campos de los nodos serán los siguientes:

| CAMPO | FINALIDAD |
|----------------------|--|
| <i>Nombre</i> | Almacenar el nombre de una acción, será de tipo <i>string</i> . |
| <i>Parameters</i> | Lista para almacenar los parámetros introducidos en la acción. Esta lista tendrá la misma estructura que la lista de tipos mencionada previamente. |
| <i>Preconditions</i> | Estructura de tipo registro para almacenar la precondición de una acción. |
| <i>Effects</i> | Estructura de tipo registro para almacenar el efecto de una acción. |
| <i>sgte</i> | Puntero para poder enlazar con el nodo siguiente de la presente lista. |

Tabla 10 Campos de la lista de acciones y finalidad de los mismos

El proceso para pedir las acciones será el mismo que para los predicados y funciones, es decir, un bucle hasta que se introduzca vacío y ya haya al menos una acción correcta introducida. Sin embargo, hay que pedir los parámetros. Para ello se realiza el mismo proceso que para la introducción de tipos para un predicado. Pero se

puede encontrar la situación de que el usuario no introduzca ningún parámetro, debido a que haya un predicado o función sin ellos y va a ser utilizado en la precondition o en el efecto de la acción, o simplemente, por error. Se comprobará mediante la función *hayPredFunctST* pasándose como parámetro o la lista de predicados o la lista de funciones. Pero en caso negativo, todavía se comprueba mediante *nodejaVacioParConst* si es posible la no introducción de parámetros mediante las constantes. Si todas estas funciones devuelven el valor falso, entonces hay que introducir algún parámetro, en caso contrario se puede seguir sin la introducción de los mismos. *pedirPrecondicion* será el procedimiento encargado de pedir una precondition y almacenarla. El registro para almacenar las precondiciones (*TPrec*) tiene los siguientes campos:

| CAMPO | FINALIDAD |
|---------------------|---|
| <i>operador</i> | Almacenar el operador introducido, será de tipo <i>string</i> . |
| <i>tipo</i> | Almacenar si es una precondition o un efecto, será de tipo <i>string</i> . |
| <i>caso_imply</i> | Almacenar si el operador introducido en primer lugar es un <i>imply</i> , será de tipo <i>string</i> . |
| <i>t_fluent</i> | Almacenará el registro denominado <i>t_fluent</i> que será descrito posteriormente. |
| <i>pf</i> | Contener el nodo del predicado o función correspondiente si el operador introducido ha sido "" o <i>not</i> . |
| <i>lista_o</i> | Almacenar la estructura de la lista <i>TLista_Operador</i> creada para los operadores, que se describirá posteriormente. |
| <i>op_imply</i> | Almacenar el operador correspondiente en caso de tratarse el primer operador de un <i>imply</i> . |
| <i>pf_imply</i> | Almacenar el nodo correspondiente al predicado o función de un <i>imply</i> si el siguiente operador introducido es "" o <i>not</i> . |
| <i>lista_eimply</i> | Almacenar los efectos para el caso de que se haya introducido un <i>when</i> como primer operador. |

Tabla 11 Campos del registro *TPrec* y funcionalidad de los mismos

A continuación se describen los campos de un nodo de la lista *TLista_Operador*:

| CAMPO | FINALIDAD |
|------------------|--|
| <i>ope</i> | Almacenar el operador, de tipo <i>string</i> . |
| <i>variable</i> | Almacenar el identificador para el caso de haber introducido el operador <i>exists</i> , de tipo <i>string</i> . |
| <i>variables</i> | Almacenar la variable para el caso de haber introducido el operador <i>forall</i> , de tipo <i>string</i> . |
| <i>literales</i> | Almacenar en un registro donde cada campo almacenará tanto el nuevo operador (<i>op</i> , campo de tipo <i>string</i>), <i>fluent</i> que es el campo que se encarga de almacenar <i>tipo_fluent</i> que será explicado posteriormente, el campo <i>predfunct</i> que almacena el predicado o función en caso de ser el operador “” o <i>not</i> , y por último el campo <i>lista_literal</i> que es una lista <i>TPLista_andor</i> que será explicada posteriormente. |
| <i>sgte</i> | Enlazar con el siguiente nodo de la lista. |

Tabla 12 Campos de la lista *TLista_Operador* y funcionalidad de los mismos

Los campos del registro *tipo_fluent* se muestran a continuación en la siguiente tabla:

| CAMPO | FINALIDAD |
|---------------|--|
| <i>fluent</i> | Almacenar el operador relacional, de tipo <i>string</i> . |
| <i>signo</i> | Almacenar el signo, de tipo <i>string</i> . |
| <i>funct1</i> | Almacenar el nodo de la lista de funciones. |
| <i>funct2</i> | Almacenar en un campo <i>f</i> el nodo de la función en caso de haber sido introducida y en <i>n</i> el valor en caso de no haberse introducido función. |
| <i>funct3</i> | Almacenar en un campo <i>f</i> el nodo de la función en caso de haber sido introducida y en <i>n</i> el valor en caso de no haberse introducido función. |

Tabla 13 Campos del registro *tipo_fluent* y funcionalidad de los mismos

A continuación se muestra la estructura de *TPLista_andor*:

| CAMPO | FUNCIONALIDAD |
|----------------|--|
| <i>oper</i> | Almacenar el operador, es de tipo <i>string</i> . |
| <i>literal</i> | Almacenar el nodo determinado de la lista de predicados o funciones. |
| <i>sgte</i> | Poder enlazar con el siguiente nodo de la lista. |

Tabla 14 Campos de *TPLista_andor* y funcionalidad de los mismos

Partiendo de que la estructura para almacenar predicados y funciones fue detallada anteriormente, en primer lugar se pedirá el operador correspondiente a la precondition (mediante la función *pedir_operator*), que el usuario deberá seleccionar de la lista que se le muestra. La función *operadorPermitido* será la encargada de comprobar que el operador introducido es correcto. Según el operador introducido se llamará a distintos procedimientos.

Hay que destacar, que en caso de “and” u “or” es necesario pedir el número máximo de condiciones que forman la precondition. Si el usuario ha introducido un número, pero quiere finalizar antes, bastará con introducir “INTRO” cuando se le muestre el mensaje indicado.

Para los efectos se realizará el mismo proceso que para las preconditiones, con la diferencia de que tendrán operadores distintos, por lo que al realizar las comprobaciones se hará de forma apropiada.

Una vez insertada al menos una acción, se escribirá o escribirán en el fichero mediante *escribirAccion(f, lista_acc)*; y posteriormente se mostrará un mensaje al usuario comunicándole que se ha creado el dominio satisfactoriamente y se ha guardado en el fichero “*domain.pddl*”. Se vuelve a mostrar la ventana de formulario, que se había ocultado para trabajar más cómodamente con la consola, y se vuelven a activar todas las opciones del menú principal de la ventana del formulario.

3.4.2 Subobjetivo leer dominio

Denominado en las opciones de menú “*Read a Domain*”, es uno de los subobjetivos del proyecto, concretamente el de leer un dominio de un fichero.

Lo primero será pedir al usuario el nombre del dominio determinado que quiere leer, para ello se le mostrará un “*edit*” donde podrá insertar el nombre del mismo, y un botón con la etiqueta *Read domain*.

Si pulsa el botón con esa etiqueta, se procede a leer lo escrito en el “*edit*”. Para ello se utiliza el evento “*onClick*” del propio botón. La función *FindFileofDomain* permitirá saber si ese dominio corresponde a un dominio existente en uno de los

ficheros del directorio actual. Dentro del mismo, mediante este código se obtienen en una lista los ficheros del directorio que tienen la extensión “.pddl”:

```
FindResult := FindFirst(Path + Mask, faAnyFile - faDirectory, SearchRec);
while FindResult = 0 do
begin
  insertarFin(SearchRec.Name, ", ", ", ", res, list_ficheros);
  res := res + 1;
  FindResult := FindNext(SearchRec);
end;
{ free memory }
FindClose(SearchRec);
```

Hay que añadir que también se encuentra otro botón con la etiqueta “*Click to change folder*” que si se pulsa permitirá al usuario cambiar de directorio. Se tiene que recordar que el directorio por defecto es la carpeta donde se encuentra el ejecutable, pero es posible cambiar de directorio mediante la pulsación de este botón, y se trabajará con el mismo hasta que el usuario finalice la opción o pulse el botón “*Reset*”.

Una vez se obtiene la lista, se pasa a saber si el fichero corresponde a un dominio mediante la función *dPrimParteCadCoincide*. En caso afirmativo, se consigue el dominio y se comprueba con el nombre insertado en el “*edit*” previamente. Si coinciden la función devolverá el valor *true*, mientras que si no coinciden devolverá el valor *false*. En caso negativo, *lista_do:=ListaDominiosFicheros('.', '*pddl');* va a permitir obtener una lista de todos los ficheros del directorio actual con extensión “.pddl”, almacenando en ella los nombres de los dominios posibles. Esta lista será insertada en una “*memo*” mediante “*memo1.lines.add();*”. Para facilitar al usuario la selección del dominio que desee, el usuario podrá pulsar con el ratón sobre el nombre de la “*memo*” y se escribirá en el “*edit*”.

Una vez pulsado el botón, se pasa a la lectura del dominio, en *leerFichero*. En él, se comprueban los paréntesis de inicio con los de final, y si coinciden, hay que tratar esa cadena. Para ello se llama a la función *decideLeer* y dentro de la misma a *dPrimParte* (devuelve la primera parte de la cadena hasta que se encuentra un espacio) para obtener si se trata de los requisitos (:requirements), los tipos (:types), las constantes (:constants), los predicados (:predicates), las funciones (:functions) o las acciones (:actions). Si se trata de acciones, se añaden en la “*memo*” para tenerlas almacenadas. Según cada primera parte devuelta, se llamará a distintos procedimientos para proceder a la lectura de los mismos. Hay que destacar que la función *devolverPalabra* devuelve lo escrito desde la posición indicada hasta encontrar un espacio, es decir una palabra, e *insertarFin* se encarga de insertar, en una lista en este caso, la palabra obtenida anteriormente, es decir, el requisito.

Una vez almacenado todo el dominio, es hora de representarlo. Para que se muestre continuamente por la pantalla del formulario, se ha elegido representarlo dentro de una “*paintBox*”, y es necesario escribir el código dentro del evento *onPaint* de la misma. Para ello se han creado diversas variables booleanas que controlarán las

distintas representaciones. En este caso, es *pb1:=true;* y *paintBox1.invalidate;* para llamar al evento *onPaint*. Dentro del mismo se llama al procedimiento *MostrarTipandConst*. En él, se procederá a la representación de tipos y constantes del mismo. De ello se encargará el procedimiento *representarObjOptm* al cual es necesario pasar la lista de objetos del dominio así como la altura inicial de la “*paintBox*”.

Para la representación se ha utilizado el componente Canvas. Canvas tiene muchas propiedades, muy valiosas a la hora de la representación de gráficos por pantalla, por la cual ha sido elegido [9].

Pero no solamente ha sido necesaria una “*paintBox*”. También ha sido necesaria otra para poder mostrar la leyenda tal y como indican los requisitos de usuario descritos anteriormente, y a su vez es necesaria su representación mediante el evento *onPaint*.

Habiendo mencionado ya la leyenda, se vuelve al punto anterior, que era que una vez representados los tipos y constantes, se pasa a representar los predicados y funciones mediante *RepresentarFigurasParaDominio*. En él, en primer lugar se representarán los tipos de un determinado predicado o función, para posteriormente representar el propio predicado y función en base a la altura y ancho que han ocupado estos tipos. Así se irá realizando hasta que finalmente no haya más predicados o funciones que representar. Al finalizar el proceso, se volverán a activar todas las opciones del menú principal de la pantalla formulario tal y como se mostró en el apartado 3.4.1.

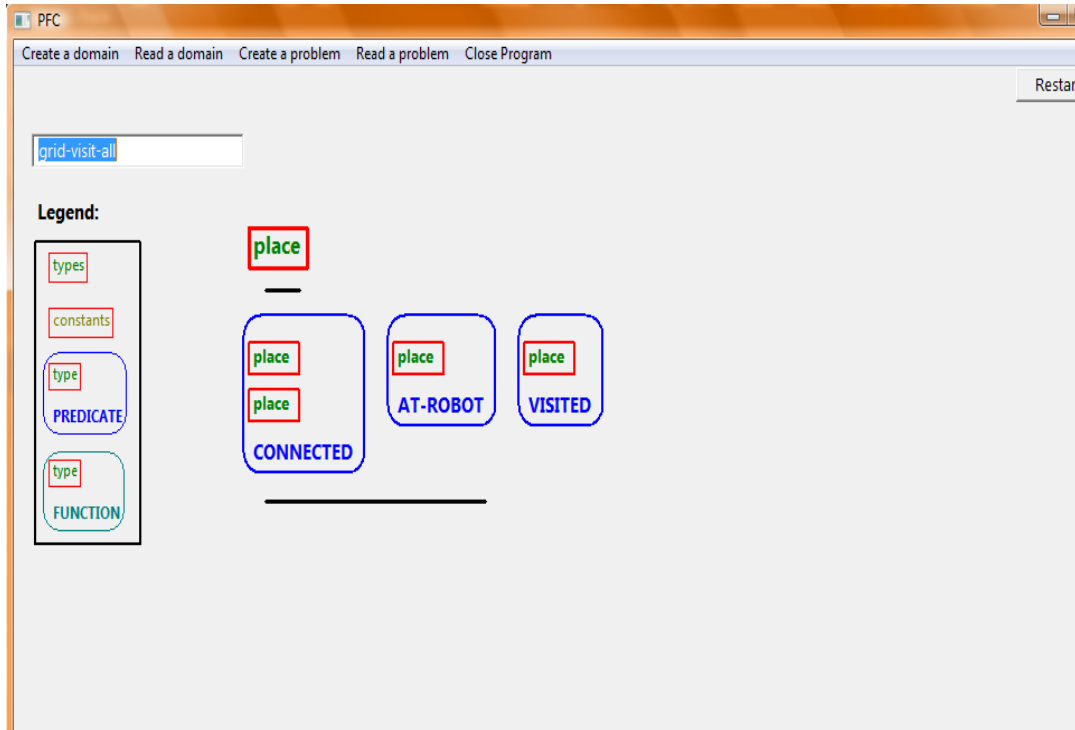


Ilustración 13 Las opciones vuelven a ser activadas tras ser representado el dominio leído

3.4.3 Subobjetivo crear problema

El subobjetivo crear problema se encuentra identificado como “*Create a Problem*” en las opciones del menú de la pantalla del formulario. Este subobjetivo consiste en que partiendo de un dominio de planificación se pueda crear un problema de planificación mediante la ayuda de la representación gráfica.

Una vez que el usuario haya pulsado esta opción, se pueden encontrar dos casos, el primero que el usuario haya leído previamente un dominio y no haya pulsado ni el botón con la etiqueta “*reset*” ni haya cerrado la aplicación. El segundo caso es que quiera introducir el nombre del fichero donde se encuentra el dominio.

Para saber en qué situación se encuentra, se ha definido el booleano “*act_rd*” que tendrá valor *true* cuando el usuario pulse la opción del menú de leer un dominio, pero tendrá valor *false* cuando se pulse el botón con la etiqueta “*reset*” y se inicialice la aplicación. Por tanto si el dominio se encuentra almacenado en memoria no hace falta leer otra vez el fichero, pero si *act_rd* tiene valor *false* habrá que pedir al usuario el nombre del fichero donde se encuentra el dominio que quiere ser leído. Este procedimiento será el mismo que para la opción de leer un problema de planificación y muy parecido al descrito en el apartado 3.4.2, ya que se realizará de la misma manera con la diferencia de que esta vez se pide el nombre del fichero en lugar del dominio, al igual que la lectura del fichero. De la misma manera, también se podrá cambiar el directorio actual de trabajo.

Una vez que se tiene el dominio almacenado en memoria, ya se puede empezar a crear el problema. A partir de ahora, siempre se va a mostrar en la pantalla el botón con la etiqueta “*reset*” y el botón con la etiqueta “*Click to show actions*”. Pulsando sobre él, se mostrarán las acciones del dominio en una “*memo*” tal y como fueron insertadas durante la lectura del fichero, y en caso de volver a ser pulsado, se ocultarán.

A su vez, en el evento *onPaint* de la “*paintBox*” se activa el tipo para que se pueda mostrar la leyenda correspondiente a la representación del estado inicial, tal y como se especificó en los requisitos.

En la parte izquierda, debajo de la leyenda, se muestran los nombres de los tipos y de las constantes (mediante *labels*) que se pueden seleccionar cada vez que se pulse con el ratón sobre uno de ellos para añadir al problema. Para que todo esto sea posible, se ha definido un *array* dinámico de *labels* que contiene tanto los tipos como las constantes. En primer lugar se han contado los tipos y las constantes del dominio, y se ha establecido la longitud del *array* en base a ese número.

Para comprobar si un tipo o constante es seleccionado, se utiliza nuevamente la propiedad “*onClick*” y se llama al procedimiento *OCseleccionados*.

En ese procedimiento, se va a crear la lista de objetos del problema, obteniendo el nombre del tipo o constante seleccionado, y creando la lista mediante el procedimiento *crearListaTCP* para posteriormente representarlo mediante *representarFiguras*.

Nada más pulsar el primer *label*, es decir, al seleccionar el primer tipo u objeto que pasará a formar parte de los objetos del problema, aparecerá el botón con la etiqueta “*Click if you want to delete an object*”. Si se pulsa sobre él, aparecerán los nombres de los objetos que han sido seleccionados para el problema, además de otro botón con la etiqueta “*Click to delete*”. Para la creación del *checkBox* donde aparecerán todos los nombres mencionados anteriormente, el proceso ha sido el mismo que para los *labels*. Posteriormente se produce la llamada al procedimiento encargado de borrar los tipos y constantes que han sido seleccionados del *checkBox*. Y en *BorrarElementoIden* se borraría o de la lista de tipos o de constantes buscando el identificador determinado.

Este proceso de selección de tipos y constantes (además del posible borrado de los mismos) se hará repetidamente hasta que el usuario pulse el botón con la etiqueta “*Click to continue adding predicates and functions*”, situado justo en la parte inferior del último tipo o constante mostrada para elegir. Este botón al ser pulsado llamará al procedimiento *mostrarPredyFunct*. Este procedimiento se va a encargar de borrar los tipos y constantes hasta ahora mostrados para mostrar ahora los *labels* correspondientes a los nombres de predicados y funciones, y, a su vez, al lado de los mismos, se mostrarán los tipos por los que están compuestos entre paréntesis. Pero también se encarga de comprobar que un predicado o función puede ser seleccionado por el usuario viendo si sus tipos se corresponden a objetos del problema. En caso de que no se pudieran seleccionar todos los tipos del mismo mediante los objetos del problema, el predicado será desactivado cambiando la propiedad “*enabled*” del predicado determinado a *false*, de tal forma que ya no se podrá pulsar con el ratón sobre él.

El usuario entonces sólo puede pulsar con el ratón sobre los activados, y una vez seleccionado uno se llama al procedimiento *PFSeleccionados*. Este procedimiento se encarga de comprobar todos los objetos que pueden ser seleccionados para los tipos del predicado o función seleccionada, y se muestra otro botón con la etiqueta “*Click to add to the predicate*” o “*Click to add to the function*” según corresponda. Con los objetos obtenidos, que pueden valer para ser seleccionados como tipos, se crea como en el proceso anterior una *checkBox* donde se mostrarán los nombres que puede seleccionar el usuario. Hay que tener en cuenta que si por ejemplo una función no tiene tipos, hay que pedir el número de inicialización de la misma. Para ello se mostrará un “*edit*” en donde el usuario insertará el número.

Una vez que se ha introducido al menos un predicado o una función de forma correcta para formar parte del estado inicial del problema, se mostrará el botón con la etiqueta “*Click to delete a predicate or a function*”, por si el usuario se ha equivocado al introducir los predicados o las funciones. El procedimiento para poder borrarlo será muy parecido al mencionado anteriormente para poder borrar tipos y constantes, con la diferencia de que ahora se llamará al predicado *borrarPredFunc*.

Al igual que el botón anterior, el botón “*Click if you want to start with the goal*” aparecerá una vez que se haya seleccionado al menos un predicado o función y se hayan introducido de forma correcta sus tipos. Si es pulsado, se pasará a elegir los predicados y funciones posibles para el estado meta del problema, cambiando la representación de

los mismos, ya que ahora los predicados y funciones que formen parte del estado inicial serán de color azul y los del estado final de color marrón.

El proceso para seleccionar los predicados y funciones para el estado final será el mismo que el explicado anteriormente para la selección del estado inicial, a diferencia de que ahora aparecerá el botón “*Click to finish*” que permitirá saber si el usuario ya ha terminado de crear el estado final del problema.

Una vez que el usuario pulsa ese botón, en la pantalla sólo se mostrará un “edit” para introducir la métrica del problema final y el botón para su correspondiente inserción, que tendrá la etiqueta *Insert* (además de los mencionados para mostrar las acciones y para inicializar de nuevo la aplicación, que se muestran siempre). Para poder borrar por ejemplo los *labels* creados, se utilizaría la propiedad *.free*; mientras que para evitar la representación por pantalla, todos los valores de los booleanos que actuaban en el evento *onPaint* de ambas *paintBox* han sido puestos con valor *false*.

La propiedad *.clear* permite limpiar el “edit” para facilitar al usuario una nueva inserción. En este caso, será la del nombre del problema final, que a su vez será el nombre del fichero que contendrá el problema.

El código para la obtención del nombre del problema será el mismo de la métrica, pero como ya se ha terminado con esta opción del menú, se añaden líneas de código que permitan volver la aplicación a su estado inicial:

```
memo1.clear;  
memo1.visible:=false;  
//Se crea el fichero del problema.  
crearFicheroProblema;  
CreateDomain.enabled:=true;  
ReadDomain.enabled:=true;  
CreateProblem.enabled:=true;  
ReadProblem.enabled:=true;  
//Se le comunica al usuario que ha realizado los pasos satisfactoriamente.  
showmessage('You have created the problem successfully!');
```

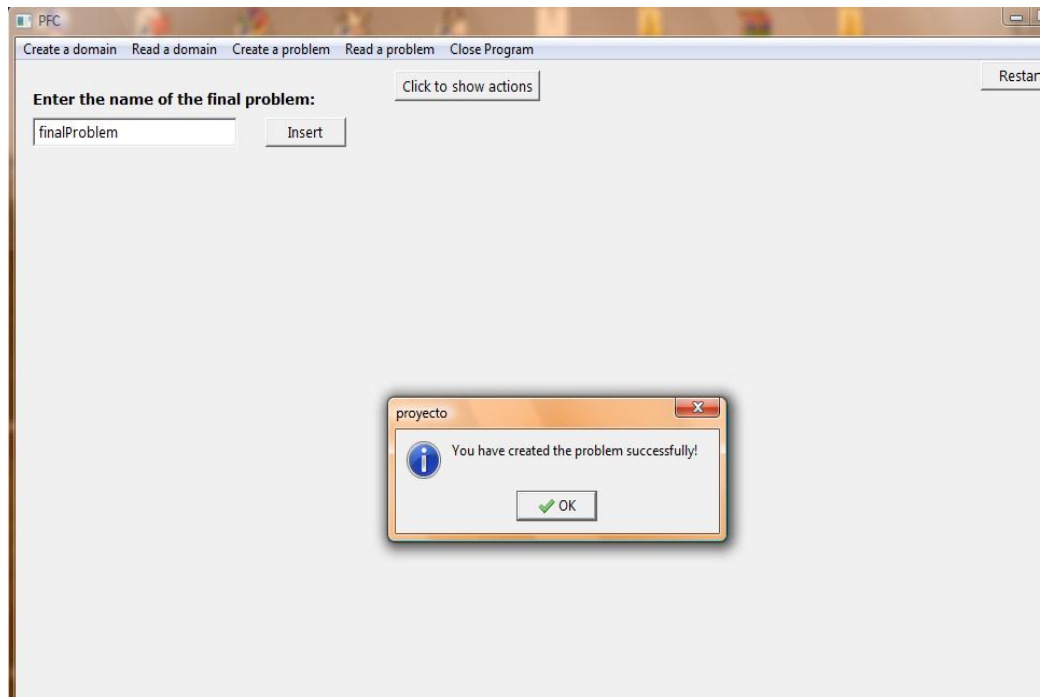


Ilustración 14 Se activan todas las opciones del menú tras mostrarse el mensaje de creación satisfactoria del problema

En el anterior fragmento de código se puede apreciar la llamada al procedimiento *crearFicheroProblema*, que es el encargado de escribir el problema almacenado en memoria en el fichero destinado a su almacenamiento. Dentro del mismo, lo primero será crear el nombre del fichero mediante *nombre:=n_fp+'.pddl'*; donde *n_fp* como fue mencionado anteriormente era la variable donde se encontraba el nombre del problema, a la cual es necesaria la extensión *".pddl"* tal y como se acordó en los requisitos para poder nombrar correctamente al fichero.

3.4.4 Subobjetivo leer problema

Identificado como *"Read Problem"* en las opciones del menú de la pantalla del formulario. Este subobjetivo consiste en leer un problema de planificación y a partir de él hacer una representación gráfica del mismo. Para ello se realizará el mismo proceso que en el apartado 3.4.2, a la hora de pedir el problema, con la única diferencia de que ahora se pedirá el nombre del fichero donde se encuentra almacenado el problema. En este caso, también se podrá cambiar el directorio actual de trabajo. Para su comprobación se llamará a la función *FindProblemToRead*. Una vez comprobada la existencia del fichero, se llama al procedimiento *leerFicheroProblema*, donde a su vez se va a comprobar que existe el dominio del problema. En caso de que ese dominio no exista en otro fichero, no se podrá realizar este subobjetivo y se volverá al menú inicial automáticamente. Para la comprobación de la existencia de ese dominio, se llamará a

FindFiles, que en el caso de devolver vacío indica que el dominio no existe y se volvería a la situación inicial.

Si por el contrario el dominio existe, se procederá a la lectura y almacenamiento tanto de los predicados y funciones de ese dominio, para poder identificar los predicados y funciones del problema (mediante *almacenar_pred_func*), como a la lectura del propio fichero donde se encuentra el problema, mediante el mismo procedimiento mencionado anteriormente en el apartado 3.4.2, es decir, viendo la primera parte de la cadena según los paréntesis correspondientes, y a diferencia del apartado anterior, identificando si es:

- *:objects*: se trata de los objetos del problema.
- *:init*: se trata de los predicados y funciones del estado inicial del problema.
- *:goal*: se trata de los predicados y funciones del estado final del problema.
- *:metric*: se trata de la métrica del problema.

Para la representación del problema, una vez leídos e identificados los predicados y funciones del mismo, se llamará en primer lugar al procedimiento *MostrarTipandConst* descrito anteriormente, y posteriormente para los predicados y funciones a *RepresentarFigurasFP*, el cual se encargará tanto del estado inicial como del estado final. Este procedimiento se basa en la forma de representación del apartado 3.4.2, con la diferencia de los colores, ya que se estableció en los requisitos que en el estado inicial tanto los predicados como las funciones deberían ser de color azul y en el estado final deberían ser de color marrón. Otra diferencia respecto a ese apartado es la representación de la leyenda, a la cual habrá que pasar como parámetro un tipo distinto, en este caso concreto el “3”.

3.4.5 Cerrar programa

En las opciones de menú se identifica mediante “*Close Program*”. No es un subobjetivo del programa pero sí es un requisito de usuario, por lo tanto se ha decidido incluirlo en las opciones de menú.

Dentro del evento “*onClick*” se ha llamado al procedimiento *cerrarPrograma* que mediante la sentencia *close* termina la aplicación de manera inmediata.

4. PRUEBAS

En esta fase se van a detallar las pruebas que han sido necesarias para comprobar el correcto funcionamiento de la aplicación. Con la realización de estas pruebas se pretende comprobar que se cumple el objetivo para el que ha sido realizada la aplicación. Como ya se mencionó en el capítulo 2.1 el objetivo principal de la aplicación se subdividió en cuatro subobjetivos que se recuerdan a continuación:

- Creación de un dominio de planificación y su posterior almacenamiento en un fichero.
- Lectura de un fichero que contenga un dominio de planificación y representación gráfica del mismo.
- Creación de un problema de planificación y su posterior almacenamiento en un fichero. Todo ello mediante representación gráfica.
- Lectura de un fichero que contenga problema de planificación para un dominio y representación gráfica del mismo.

Por tanto, para verificar el cumplimiento de los mismos, es necesario realizar una evaluación completa de la aplicación para verificar que la aplicación funciona correctamente y cumple con los requisitos establecidos.

Para ello, partiendo de los cuatro subobjetivos y el cumplimiento de los mismos, se han realizado una serie de pruebas para verificar cada uno de ellos. En la mayoría de los casos se ha trabajado con los dominios de la International Planning Competition 2011 [10].

4.1 Pruebas de creación de dominio

En este apartado se detallarán, de todas las pruebas realizadas, las más destacadas para comprobar el cumplimiento del primer subobjetivo de la aplicación, es decir, la creación de un dominio por parte del usuario y su almacenamiento tanto en la aplicación como en el fichero “*domain.pddl*”.

A continuación se muestra una tabla donde además de las pruebas, se muestran los resultados obtenidos y los resultados esperados.

| Nº | PRUEBA | RESULTADO ESPERADO | RESULTADO OBTENIDO |
|----|--|--|----------------------------------|
| 1 | Pulsar la tecla “ <i>alt</i> ”, y posteriormente pulsar la tecla “ <i>alt</i> ” + la tecla “ <i>d</i> ” sobre opción “Create a domain” | En primer lugar se debe subrayar la letra “ <i>d</i> ” de <i>domain</i> y posteriormente debe desaparecer la pantalla de formulario donde se encuentra el menú, quedando únicamente la consola. | Se cumple el resultado esperado. |
| 2 | Introducción de nombre vacío al pedir el dominio. | Se debe mostrar un mensaje diciendo que el usuario no ha introducido nada y se vuelve a pedir el dominio. | Se cumple el resultado esperado. |
| 3 | Introducción de la cadena “caso” a la hora de pedir los requisitos. | Se debe borrar la cadena “caso” al no pertenecer a los requisitos del sistema y se vuelven a pedir los mismos. | Se cumple el resultado esperado. |
| 4 | Introducción de la cadena “tipo, tipo1, 1tipo, tipo2 -” a la hora de pedir los tipos. | Al introducir la cadena “tipo, tipo1, 1tipo, tipo2 -”, la aplicación debe detectar que <i>1tipo</i> no es un nombre correcto al empezar por un número, y por tanto debe insertar sólo <i>tipo</i> y <i>tipo1</i> , mientras que con el guión debe identificar que <i>tipo2</i> es un subtipo, por tanto debe pedir el subtipo para el mismo. | Se cumple el resultado esperado. |
| 5 | Introducción del nombre <i>tipo3</i> como elección del subtipo <i>tipo2</i> en la prueba 4. | La aplicación debe mostrar un mensaje error comunicando que se ha introducido algo incorrecto al comprobarse que no es un tipo insertado y debe volverse a pedir la inserción de un tipo. | Se cumple el resultado esperado. |
| 6 | Pulsar la letra “f” cuando se le pide al usuario si quiere introducir constantes. | La aplicación debe pasar a pedir los predicados o funciones según corresponda. | Se cumple el resultado esperado. |

| | | | |
|----|---|---|----------------------------------|
| 7 | Introducir como tipo para una constante que se va a introducir desde los tipos insertados en la prueba 5, el número 6. | Se le debe comunicar al usuario que ha introducido algo incorrecto, al comprobar que este número no corresponde con ningún tipo ni subtipo insertado previamente, y se debe volver a pedir el tipo o subtipo correspondiente. | Se cumple el resultado esperado. |
| 8 | Introducir un predicado de nombre "at" cuando ya había insertado uno previamente. | Al introducir un nombre del predicado se debe comprobar si empieza por número o está repetido, y en este caso se cumple la segunda opción y por tanto se debe mostrar el mensaje del error al usuario y en consecuencia, proceder a pedir una nueva introducción. | Se cumple el resultado esperado. |
| 9 | Intentar introducir funciones sin introducir el requisito ":action-costs". | Al no introducir este requisito, no está permitido insertar funciones en el dominio, por tanto no debe dejar introducir las mismas. | Se cumple el resultado esperado. |
| 10 | Intentar introducir constantes sin introducir ni el requisito ":adl", ni ":typing". | Al no introducir ninguno de estos requisitos, no está permitido insertar constantes en el dominio, por tanto no debe dejar introducir las. | Se cumple el resultado esperado. |
| 11 | Intento de creación del dominio <i>pegsolitaire-sequential</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | El dominio debe ser almacenado en memoria y en el fichero " <i>domain.pddl</i> ". | Se cumple el resultado esperado. |
| 12 | Intento de creación del dominio <i>transport</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | El dominio debe ser almacenado en memoria y en el fichero " <i>domain.pddl</i> ". | Se cumple el resultado esperado. |

| | | | |
|----|--|---|----------------------------------|
| 13 | Intento de creación del dominio <i>scanalyzer3d</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | El dominio debe ser almacenado en memoria y en el fichero “ <i>domain.pddl</i> ”. | Se cumple el resultado esperado. |
| 14 | Intento de creación del dominio <i>dominio_prueba</i> , cuya única finalidad es mostrar el correcto funcionamiento de los posibles operadores en las acciones. | El dominio debe ser almacenado en memoria y en el fichero “ <i>domain.pddl</i> ”. | Se cumple el resultado esperado. |

Tabla 15 Pruebas de creación de dominio

4.2 Pruebas de lectura de dominio

En este apartado se detallarán, de todas las pruebas realizadas, las más destacadas para comprobar el cumplimiento del segundo subobjetivo de la aplicación, es decir, leer un dominio de planificación de un fichero y representarlo gráficamente por la pantalla del formulario.

En la tabla que se muestra a continuación, además de las pruebas, se muestran los resultados obtenidos y los resultados esperados. Tal y como se especificó en los requisitos no funcionales, no se podrá excluir “- object” en la sintaxis empleada a la hora de declarar los tipos, al igual que en la declaración de *fluents* no se podía excluir “- number”. Por tanto, han tenido que ser modificados los ficheros que no cumplían estos requisitos no funcionales.

| Nº | PRUEBA | RESULTADO ESPERADO | RESULTADO OBTENIDO |
|----|--|---|----------------------------------|
| 1 | Nada más inicializar la aplicación, pulsar la tecla “ <i>alt</i> ”, y posteriormente pulsar la tecla “ <i>alt</i> ” + la tecla “ <i>r</i> ” sobre opción “Read a domain” | En primer lugar se debe subrayar la letra “ <i>r</i> ” de <i>Read</i> y posteriormente se debe pedir el nombre del dominio. | Se cumple el resultado esperado. |
| 2 | Introducir como nombre de fichero vacío. | Debe aparecer la lista de dominios posibles. | Se cumple el resultado esperado. |

| | | | |
|---|--|---|----------------------------------|
| 3 | Introducir un nombre de dominio inexistente en el directorio actual. | Debe mostrarse un mensaje de error en la introducción y posteriormente aparecer la lista de dominios posibles. | Se cumple el resultado esperado. |
| 4 | Introducción del dominio <i>barman</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 5 | Introducción del dominio <i>elevators-sequencedstrips</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 6 | Introducción del dominio <i>Parcprinter</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 7 | Introducción del dominio <i>floor-tile</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 8 | Introducción del dominio <i>grid-visit-all</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 9 | Introducción del dominio <i>openstacks-sequencedstrips-nonADL-</i> | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, | Se cumple el resultado esperado. |

| | | | |
|----|--|---|----------------------------------|
| | <i>nonNegatedp01</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | |
| 10 | Introducción del dominio <i>openstacks-sequencedstrips-nonADL-nonNegatedp20</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 11 | Introducción del dominio <i>parking</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 12 | Introducción del dominio <i>pegsolitaire-sequential</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 13 | Introducción del dominio <i>scanalyzer3d</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 14 | Introducción del dominio <i>sokoban-sequential</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |

| | | | |
|----|---|---|----------------------------------|
| 15 | Introducción del dominio <i>tidybot</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 16 | Introducción del dominio <i>transport</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 17 | Introducción del dominio <i>transport-strips</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 18 | Introducción del dominio <i>woodworking</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese dominio y una vez leído representar los tipos, constantes, predicados y funciones del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |

Tabla 16 Pruebas de lectura de dominio

4.3 Pruebas de creación de un problema

En este apartado se detallarán, de todas las pruebas realizadas, las más destacadas para comprobar el cumplimiento del tercer subobjetivo de la aplicación, es decir, la lectura de un dominio de planificación y a partir del mismo la creación de un problema de planificación mediante la ayuda de la representación gráfica y su escritura en un fichero de extensión “.pddl”.

En la siguiente tabla además, se muestra el resultado obtenido y el resultado esperado.

| Nº | PRUEBA | RESULTADO ESPERADO | RESULTADO OBTENIDO |
|----|---|---|----------------------------------|
| 1 | Nada más inicializar la aplicación, pulsar la tecla “alt”, y posteriormente pulsar la tecla “alt” + la tecla “c” sobre opción “Create a problem” | En primer lugar se debe subrayar la letra “c” de <i>Create</i> y posteriormente se debe pedir el nombre del fichero donde se encuentra el problema. | Se cumple el resultado esperado. |
| 2 | Nada más inicializar la aplicación, y seleccionando la opción “Create a problem” introducir un nombre de fichero vacío. | Debe aparecer una lista con los nombres de fichero con un dominio disponibles. | Se cumple el resultado esperado. |
| 3 | Nada más inicializar la aplicación, y seleccionando la opción “Create a problem” introducir como nombre de fichero de dominio uno no existente. | Debe mostrarse un mensaje error y además aparecer una lista con los nombres de fichero con un dominio disponibles. | Se cumple el resultado esperado. |
| 4 | Tras haber seleccionado unos tipos y constantes para el problema, pulsar el botón de borrado y seleccionar un objeto que aparece en la “checkBox” y borrarlo pulsando el botón correspondiente. | La aplicación al pulsar el botón de borrado debe mostrar un “checkBox” donde se muestren todos los objetos que han sido seleccionados para el problema, y comprobar al pulsar el consiguiente botón de borrado cuál o cuáles han sido seleccionados y lo debe borrar de la lista destinada al problema. Además, lógicamente, tampoco debe aparecer su representación. | Se cumple el resultado esperado. |
| 5 | Pulsar el botón de muestra de las acciones. | La aplicación al pulsar el botón debe hacer visible la “memo” donde se encuentran las acciones y por tanto mostrarlas y a su vez cambiar la etiqueta del botón a “ <i>Click to hide actions</i> ” para que el usuario entienda qué es lo que ocurrirá al pulsarlo de nuevo. | Se cumple el resultado esperado. |

| | | | |
|----|--|--|----------------------------------|
| 6 | Pulsar el botón con etiqueta “Restart” a la mitad del proceso de creación. | La aplicación al pulsar el botón debe inicializar la aplicación, y por tanto olvidar todo lo que se encontraba en memoria hasta ese momento y activar las opciones del menú. | Se cumple el resultado esperado. |
| 7 | Pulsar el botón con la etiqueta “Click to add to the predicate” sin haber seleccionado todos los tipos por los que está compuesto el mismo. | La aplicación al pulsar el botón comprueba si se han seleccionado todos los tipos para el predicado, y al comprobar que no, muestra un mensaje al usuario comunicándole que no ha introducido todos los tipos, mostrándole cuáles son los faltantes. | Se cumple el resultado esperado. |
| 8 | Pulsar el botón con la etiqueta “Click to add to the function” para una función sin tipos sin introducir nada en el “edit” destinado a ello. | La aplicación al pulsar el botón comprueba si se ha introducido un número de inicialización de la función, y en este caso al comprobar que no, debe mostrar un mensaje al usuario comunicándole sobre su error. | Se cumple el resultado esperado. |
| 9 | Pulsar el botón con la etiqueta “Click to add to the function” para una función sin tipos habiendo introducido “cadena” en el “edit” destinado a ello. | La aplicación al pulsar el botón comprueba si se ha introducido un número de inicialización de la función, y en este caso al comprobar que se ha introducido una cadena de caracteres muestra un mensaje al usuario comunicándole su error. | Se cumple el resultado esperado. |
| 10 | Pulsar el botón con la etiqueta “Click to delete a predicate or a function” mientras se está trabajando con el estado inicial. | La aplicación al pulsar el botón muestra de forma ordenada en una “checkbox” según su representación, todos los predicados y funciones seleccionados para formar parte del estado inicial del problema. | Se cumple el resultado esperado. |

| | | | |
|----|--|--|----------------------------------|
| 11 | Pulsar el botón con la etiqueta “ <i>Click to delete</i> ” tras haber seleccionado un predicado que aparece en la “ <i>checkBox</i> ”. | La aplicación al pulsar el botón comprueba el único campo de la “ <i>checkBox</i> ” que ha sido seleccionado, y borra el predicado determinado del estado inicial del problema que se está creando, por lo que en consecuencia, una vez borrado, no muestra su representación. | Se cumple el resultado esperado. |
| 12 | Pulsar a la hora de insertar una métrica el botón con la etiqueta “ <i>Insert</i> ” sin haber introducido ninguna métrica. | La aplicación al pulsar el botón comprueba si se ha introducido una métrica, y en este caso no ha sido así, por lo que se muestra un mensaje comunicándoselo al usuario. | Se cumple el resultado esperado. |
| 13 | Pulsar a la hora de insertar el nombre del problema el botón con la etiqueta “ <i>Insert</i> ” sin haber introducido ningún nombre. | La aplicación al pulsar el botón comprueba si se ha introducido un nombre y como este caso no ha sido así, se muestra un mensaje comunicándoselo al usuario. | Se cumple el resultado esperado. |
| 14 | Intentar crear el problema <i>prueba_problema</i> a partir del dominio de planificación <i>elevators</i> . | La aplicación deberá guiar al usuario hasta que el mismo haya creado el problema. A su vez, deberá representar gráficamente el problema y escribirlo en el nuevo fichero creado con el nombre del problema (“ <i>prueba_problema.pddl</i> ”), y avisar sobre ello al usuario. | Se cumple el resultado esperado. |
| 15 | Intentar crear el problema <i>parking-parking</i> a partir del dominio de planificación <i>parking</i> . | La aplicación deberá guiar al usuario hasta que el mismo haya creado el problema. A su vez, deberá representar gráficamente el problema y escribirlo en el nuevo fichero creado con el nombre del problema (“ <i>parking-parking.pddl</i> ”), y avisar sobre ello al usuario. | Se cumple el resultado esperado. |

Tabla 17 Pruebas de creación de problema

4.4 Pruebas de lectura de problema

En este apartado se detallarán, de todas las pruebas realizadas, las más destacadas para comprobar el cumplimiento del cuarto subobjetivo de la aplicación, es decir, la lectura de un problema de un fichero y su representación gráfica por la pantalla del formulario.

En la tabla siguiente además de las pruebas, se muestra el resultado obtenido y el resultado esperado.

| Nº | PRUEBA | RESULTADO ESPERADO | RESULTADO OBTENIDO |
|----|--|---|----------------------------------|
| 1 | Nada más inicializar la aplicación, pulsar la tecla “alt”, y posteriormente pulsar la tecla “alt” + la tecla “e” sobre opción “Read a problem” | En primer lugar se debe subrayar la letra “e” de <i>Read</i> y posteriormente pedir el nombre del fichero donde se encuentra el problema. | Se cumple el resultado esperado. |
| 2 | Introducir un nombre de fichero vacío. | Debe aparecer una lista con los nombres de fichero con un problema disponibles. | Se cumple el resultado esperado. |
| 3 | Pulsar sobre un nombre de problema de la lista aparecida en la prueba 2. | El nombre debe aparecer en el recuadro. | Se cumple el resultado esperado. |
| 4 | Pulsar el botón “ <i>Click to change folder</i> ” para cambiar el directorio actual. | Debe aparecer una ventana donde aparezcan todos los directorios posibles, y la aplicación debe de reconocer el directorio que ha sido elegido por el usuario y trabajará sobre él. Además, debe mostrar en la parte superior de la pantalla cuál es el directorio en el que se encuentra actualmente. | Se cumple el resultado esperado. |
| 5 | Introducción del fichero que contiene el problema “ <i>prob</i> ” del dominio <i>barman</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |

| | | | |
|----|--|---|----------------------------------|
| 6 | Introducción del fichero que contiene el problema <i>elevators-sequencedstrips-p40_60_1</i> del dominio <i>elevators-sequencedstrips</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 7 | Introducción del fichero que contiene el problema <i>prob019</i> del dominio <i>floor-tile</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 8 | Introducción del fichero que contiene el problema <i>os-sequencedstrips-p50_3</i> del dominio <i>openstacks-sequencedstrips-nonADL-nonNegated</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 9 | Introducción del fichero que contiene el problema <i>PrintJob</i> del dominio <i>eTipp</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 10 | Introducción del fichero que contiene el problema <i>pegsolitaire-sequential-074</i> del dominio <i>pegsolitaire-sequential</i> , | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una | Se cumple el resultado esperado. |

| | | | |
|----|--|---|----------------------------------|
| | correspondiente a la categoría secuencial de la International Planning Competition 2011. | leyenda que permita identificar al usuario qué es cada dibujo. | |
| 11 | Introducción del fichero que contiene el problema <i>scanalyzer3d-41</i> del dominio <i>scanalyzer3d</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 12 | Introducción del fichero que contiene el problema <i>parking</i> del dominio <i>parking</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 13 | Introducción del fichero que contiene el problema <i>p117-microban-sequential</i> del dominio <i>sokoban-sequential</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |
| 14 | Introducción del fichero que contiene el problema <i>wood-prob</i> del dominio <i>woodworking</i> , correspondiente a la categoría secuencial de la International Planning Competition 2011. | La aplicación debe encargarse de leer ese problema y una vez leído representar los objetos, el estado inicial y el estado final del mismo; así como de mostrar una leyenda que permita identificar al usuario qué es cada dibujo. | Se cumple el resultado esperado. |

Tabla 18 Pruebas de lectura de problema

5. CONCLUSIONES

En este apartado se recogen las conclusiones obtenidas tras la realización del proyecto.

Recordando que el objetivo principal del proyecto se había dividido en cuatro subobjetivos, como se puede observar en el apartado “*estructura de la memoria*” de esta memoria, viendo el trabajo realizado se puede comprobar que se han cumplido satisfactoriamente todos los subobjetivos y por tanto el objetivo del proyecto.

A su vez, todos los requisitos mencionados en el apartado 3.2 han sido cumplidos satisfactoriamente como se ha podido comprobar a lo largo de esta memoria.

En lo que respecta a las conclusiones personales, para la realización del proyecto un proceso clave ha sido el de la formación. Durante los inicios del proyecto ha sido necesario el estudio detallado de la documentación sobre el lenguaje *PDDL*, dominios y problemas de planificación, debido al gran desconocimiento que poseía sobre los mismos. Sin embargo, a medida que he ido avanzando con el proyecto y con las ayudas del tutor, he conseguido poco a poco familiarizarme con estos conceptos hasta adquirir un gran conocimiento.

Lo mismo puedo decir de la realización del código fuente, ha sido muy compleja porque hasta este momento nunca había trabajado a este nivel de programación tratando con formularios, ya que hasta ahora sólo había trabajado con consolas.

Y de especial dificultad fue el momento cuando a mitad del proyecto me di cuenta que no iba por el buen camino con el código fuente y el programa elegido y tuve que traspasar el código de Delphi a *Lazarus* y empezar casi desde cero de nuevo con el código del proyecto, además de familiarizarme con este nuevo programa una vez que estaba acostumbrado al anterior. En consecuencia, tengo que reconocer que me ha hecho alargar la duración del proyecto e incrementado las horas de trabajo, y que una buena planificación inicial me hubiera podido impedir este retraso.

Por tanto ha sido todo un desafío la realización del proyecto, de la cual me siento muy orgulloso de haber conseguido llevarlo adelante, porque no sólo me ha enseñado que es posible conseguir lo que uno se propone, sino que además me ha permitido adquirir importantes conocimientos tanto en programación como en dominios y problemas de planificación.

6. FUTURAS LÍNEAS/TRABAJOS

En este capítulo se comentan las posibles mejoras que se pueden realizar al proyecto en un futuro.

A continuación se muestran las líneas/mejoras que se pueden aplicar a este proyecto:

- Realización de la aplicación sin la utilización de la consola, lo cual implicaría la utilización de una sola pantalla para el usuario.
- Posibilidad de que el usuario elija el idioma que desee, ya que por defecto el idioma es el inglés.
- Permitir en los dominios la ausencia de “ – object” a la hora de la definición de los tipos.
- Extensión de la aplicación para poder soportar PDDL 3.1. Implicaría añadir nuevos requisitos, acciones durativas, preferencias y restricciones sobre la trayectoria del plan, entre otras.
- Hacer una comprobación más exhaustiva de los requisitos y los operadores que cada uno permite, ya que en el estado actual sólo se comprueban los más utilizados (como se puede apreciar en el apartado 3.4.1 de la memoria).
- Representar gráficamente las acciones.
- Mejorar la apariencia gráfica de la aplicación, permitiendo por ejemplo la selección de un icono distinto para cada tipo de objeto o para cada tipo de predicado o función.
- Permitir la representación mediante un gráfico animado de un plan, de forma que se vea cómo la aplicación de las sucesivas acciones modifica los predicados y funciones desde el estado inicial al estado final.
- Además de poder borrar, permitir editar los predicados o funciones seleccionados para formar parte del estado inicial o final del problema.
- Permitir la modificación de un dominio o problema ya creados mediante la interfaz gráfica de la aplicación.

7. PLAN DE TRABAJO Y PRESUPUESTO

En este apartado se detallan las tareas que componen el proyecto, así como el tiempo empleado en la realización de cada una de ellas y los detalles del presupuesto del proyecto.

7.1 Descomposición en tareas y duración

En este apartado se detallan las tareas por las que se compone el proyecto así como su duración.

Para cada tarea no se ha empleado el mismo horario, por lo que las horas empleadas varían según la tarea que ha sido realizada. En la siguiente tabla se muestran las horas empleadas por cada tarea:

| NOMBRE DE LA TAREA | HORAS EMPLEADAS |
|---|------------------------|
| Estudio del lenguaje PDDL | 40 |
| Análisis de requisitos | 45 |
| Realización diseño de la interfaz gráfica de la planificación | 15 |
| Realización subobjetivo “creación de dominio” | 450 |
| Realización subobjetivo “lectura de dominio” | 136 |
| Realización subobjetivo “lectura de problema” | 64 |
| Realización subobjetivo “creación de problema” | 426 |
| Pruebas de creación de dominio | 30 |
| Pruebas de lectura de dominio | 10 |
| Pruebas de creación de problema | 34 |
| Pruebas de lectura de problema | 9 |
| Realización de la memoria | 188 |
| Realización de la presentación | 20 |
| TOTAL | 1.467 |

Tabla 19 Horas empleadas por tarea

A continuación se muestra con detalle la descomposición de tareas y la fecha de comienzo y fin de las mismas:

| | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|--|--|-----------------|---------------------|---------------------|--------------|
| | 1 Inicio del proyecto | 0 días | lun 20/09/10 | lun 20/09/10 | |
| | 2 Representación gráfica de dominios de planificación | 380 días | lun 20/09/10 | vie 02/03/12 | |
| | 3 Planteamiento del problema | 10 días | lun 20/09/10 | vie 01/10/10 | |
| | 4 Estudio del lenguaje PDDL | 10 días | lun 20/09/10 | vie 01/10/10 | 1 |
| | 5 Análisis | 10 días | lun 04/10/10 | vie 15/10/10 | |
| | 6 Análisis de requisitos | 10 días | lun 04/10/10 | vie 15/10/10 | 4 |
| | 7 Diseño | 15 días | lun 18/10/10 | vie 05/11/10 | |
| | 8 Realización diseño interfaz gráfica de la aplicación | 15 días | lun 18/10/10 | vie 05/11/10 | 6 |
| | 9 Implementación | 278 días | lun 08/11/10 | mié 30/11/11 | |
| | 10 Realización subobjetivo "creación de dominio" | 115 días | lun 08/11/10 | vie 15/04/11 | 8 |
| | 11 Realización subobjetivo "lectura de dominio" | 34 días | lun 18/04/11 | jue 02/06/11 | 10 |
| | 12 Realización subobjetivo "lectura de problema" | 8 días | vie 03/06/11 | mar 14/06/11 | 11 |
| | 13 Realización subobjetivo "creación de problema" | 121 días | mié 15/06/11 | mié 30/11/11 | 12 |
| | 14 Pruebas | 46 días | jue 01/12/11 | jue 02/02/12 | |
| | 15 Pruebas de creación de dominio | 15 días | jue 01/12/11 | mié 21/12/11 | 13 |
| | 16 Pruebas de lectura de dominio | 5 días | jue 22/12/11 | mié 28/12/11 | 15 |
| | 17 Pruebas de creación de problema | 17 días | jue 29/12/11 | vie 20/01/12 | 16 |
| | 18 Pruebas de lectura de problema | 9 días | lun 23/01/12 | jue 02/02/12 | 17 |
| | 19 Realización de la memoria | 365 días | lun 04/10/10 | vie 24/02/12 | 4 |
| | 20 Realización de la presentación | 5 días | lun 27/02/12 | vie 02/03/12 | 19 |
| | 21 Fin de proyecto | 0 días | vie 02/03/12 | vie 02/03/12 | 20 |

Ilustración 15 Descomposición y duración de tareas

El diagrama de Gantt de la descomposición de tareas es el siguiente:

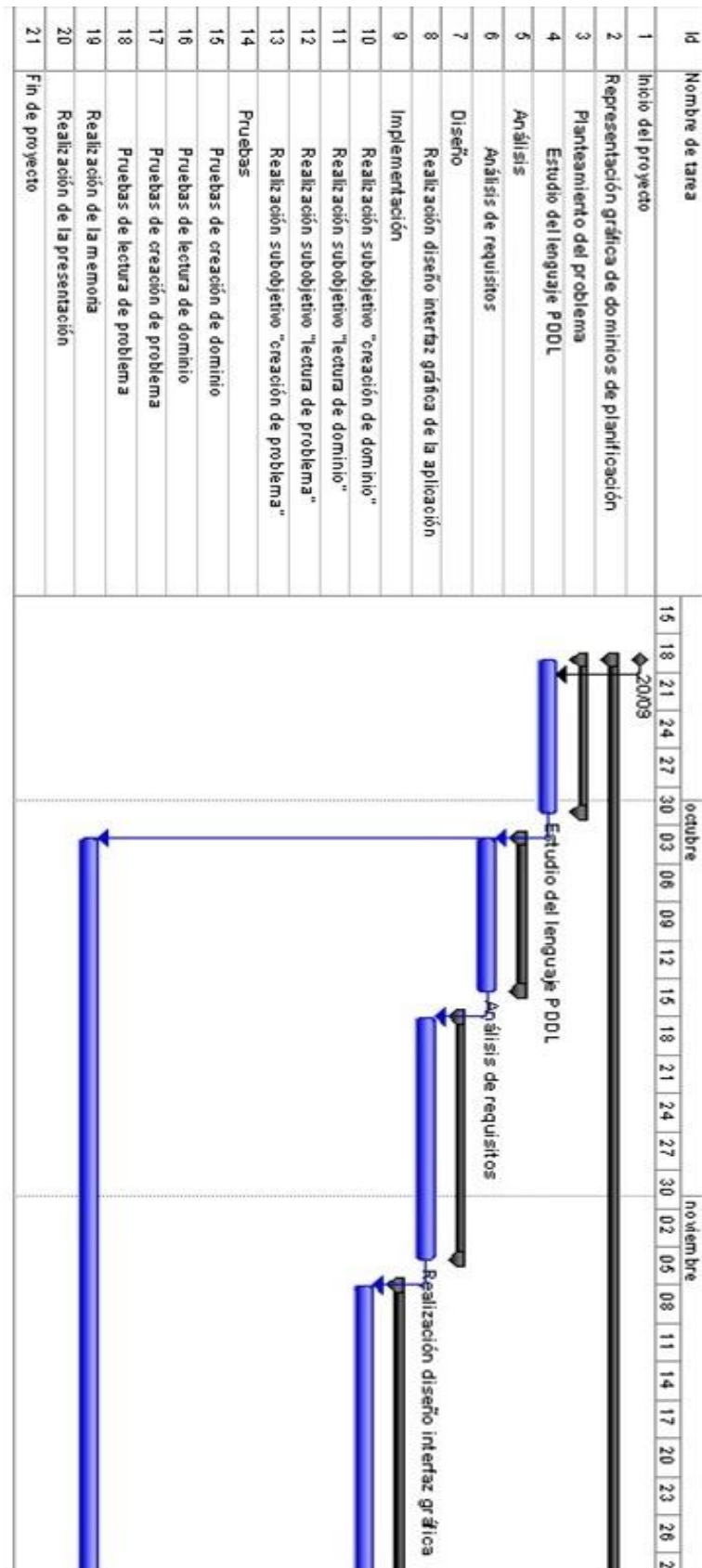


Ilustración 16 Diagrama de Gantt 1/5

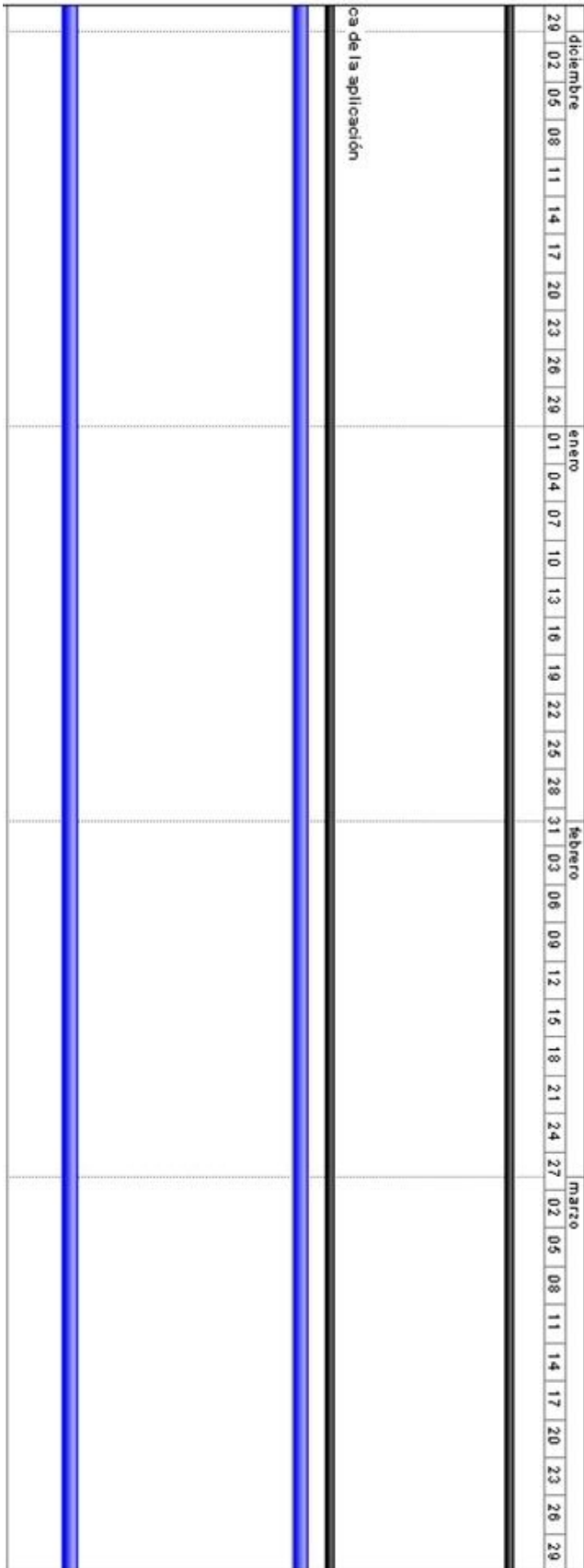


Ilustración 17 Diagrama de Gantt 2/5

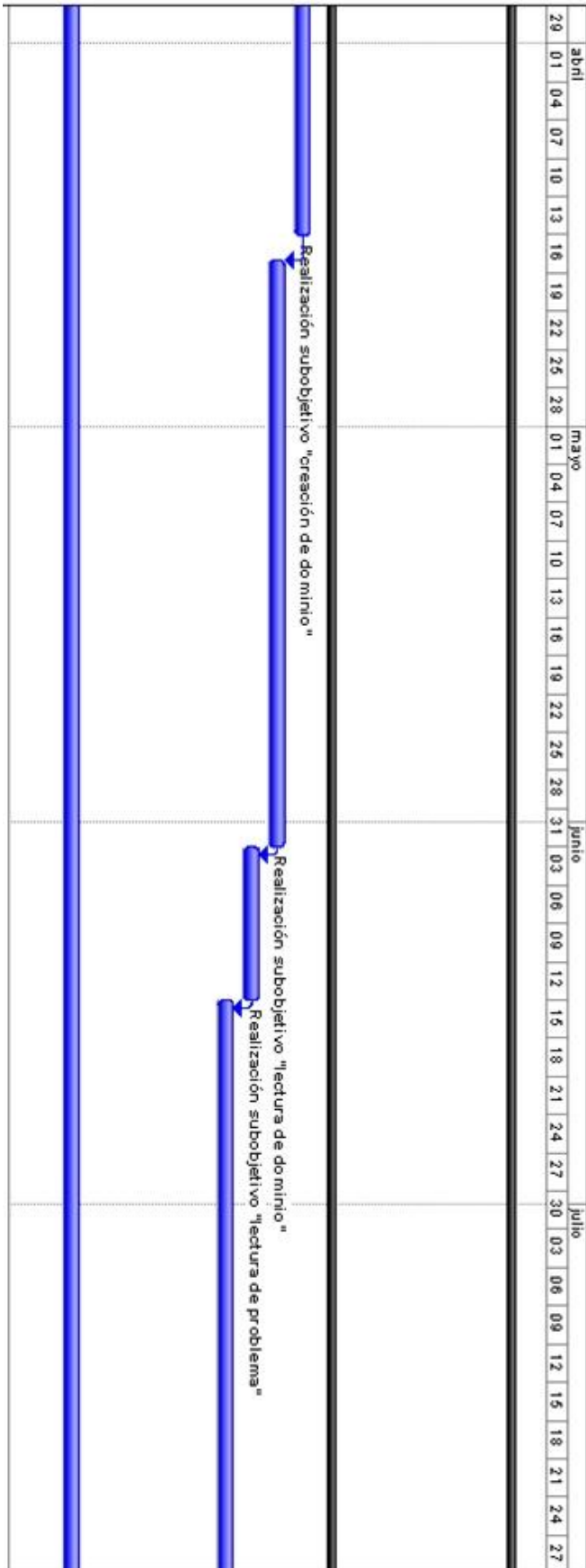


Ilustración 18 Diagrama de Gantt 3/5

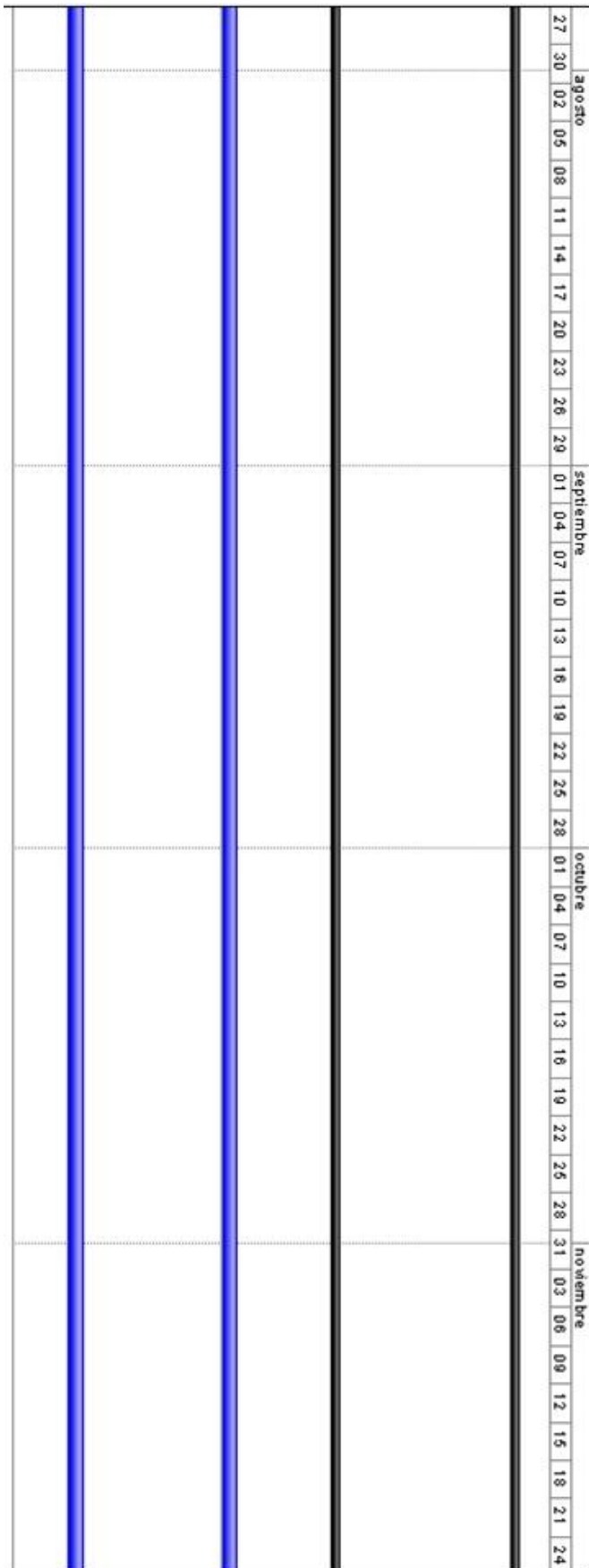


Ilustración 19 Diagrama de Gantt 4/5

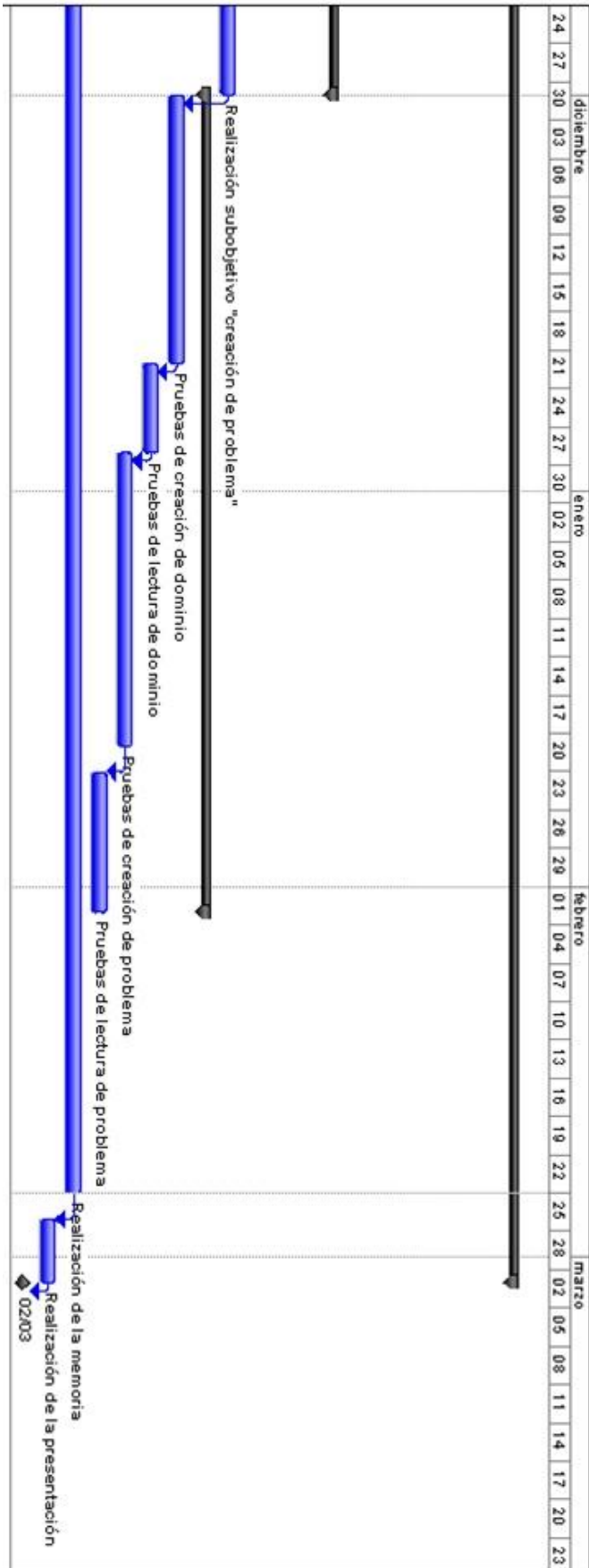


Ilustración 20 Diagrama de Gantt 5/5

Además, hay que destacar que además de haberse mantenido en contacto cuando ha sido necesario por correo con el tutor del proyecto, Ángel García Olaya, se han realizado reuniones de seguimiento, durante los siguientes días:

- El día 20 de septiembre de 2010.
- El día 1 de octubre de 2010.
- El día 15 de octubre de 2010.
- El día 11 de noviembre de 2010.
- El día 13 de diciembre de 2010.
- El día 29 de febrero de 2011.
- El día 3 de marzo de 2011.
- El día 8 de marzo de 2011.
- El día 24 de marzo de 2011.
- El día 4 de abril de 2011.
- El día 20 de junio de 2011.
- El día 1 de julio de 2011.
- El día 18 de julio de 2011.
- El día 28 de julio de 2011.
- El día 13 de septiembre de 2011.
- El día 21 de septiembre de 2011.
- El día 29 de septiembre de 2011.
- El día 20 de octubre de 2011.
- El día 14 de febrero de 2012.

Por tanto la duración del proyecto ha sido de 380 días, empleando un total de 1467 horas en él.

7.2 Presupuesto

Para analizar el presupuesto que ha supuesto el desarrollo del proyecto se dividió en dos grupos, el grupo de los recursos humanos y el de los recursos materiales.

7.2.1 Recursos humanos

En este apartado se detallará el coste que ha supuesto el proyecto en cuanto a las personas implicadas en él.

La realización del proyecto ha sido únicamente realizada por una persona, un ingeniero técnico en informática de gestión.

A continuación se muestra una tabla con las bases de cotización [11] de 2010, 2011 y la de 2012:

| Año | Grupo de cotización | Categorías profesionales | Bases mínimas – Euros/mes | Bases máximas – Euros/mes |
|------------|----------------------------|--|----------------------------------|----------------------------------|
| 2010 | 2 | Ingenieros Técnicos, Peritos y Ayudantes Titulados | 855,90 | 3.198,00 |
| 2011 | 2 | Ingenieros Técnicos, Peritos y Ayudantes Titulados | 867,00 | 3.230,10 |
| 2012 | 2 | Ingenieros Técnicos, Peritos y Ayudantes Titulados | 867,00 | 3.262,50 |

Tabla 20 Bases de cotización 2010, 2011 y 2012

Para el desarrollo de la aplicación únicamente ha sido necesaria una persona, cuyo coste depende de las horas que ha trabajado en el proyecto. De acuerdo con las bases de cotización expuestas anteriormente, y teniendo en cuenta como se ha dicho que se trata de un ingeniero técnico sin experiencia, que ha realizado las funciones de analista y de programador, se ha considerado una retribución estimada de 20€/hora. En consecuencia:

| NOMBRE | CATEGORÍA | €/HORA | HORAS TOTALES | COSTE TOTAL (€) |
|----------------------|---|---------------|----------------------|------------------------|
| Sergio Morato Villar | Ingeniero Técnico en Informática de Gestión | 20 | 1.467 | 29.340 |

Tabla 21 Coste de recursos humanos

Por tanto el coste por recursos humanos asciende a: 29.340€.

7.2.2 Recursos materiales

También ha sido necesario para el desarrollo de la aplicación el uso de material, que se detallará en este apartado.

Se ha hecho una división de los recursos materiales en el coste de los equipos [12] y el coste del material fungible.

En la siguiente tabla se muestran detallados los costes de los equipos:

| NOMBRE | COSTE POR UNIDAD (€) | CANTIDAD | PERIODO AMORTIZACIÓN (meses) | DURACIÓN PROYECTO (meses) | SUBTOTAL (€) |
|--|-----------------------------|-----------------|-------------------------------------|----------------------------------|---------------------|
| Ordenador sobremesa Windows Vista, Intel Quore Q6600, 4096MB RAM | 600 | 1 | 36 | 18 | 300 |
| Lazarus-0.9.30 | 0 | 1 | - | - | 0 |
| Lenguaje Free-Pascal versión 2-4-2. | 0 | 1 | - | - | 0 |
| Mozilla Firefox | 0 | 1 | - | - | 0 |
| Paquete Microsoft Office 2007 | 300 | 1 | - | - | 300 |
| Memoria USB 8GB | 14 | 1 | 36 | 18 | 7 |
| Multifunción HP | 60 | 1 | 36 | 1 | 1,67 |
| TOTAL (€) | | | | | 608,67 |

Tabla 22 Coste de los equipos

En la siguiente tabla se muestran detallados los costes del material fungible:

| MATERIAL | COSTE POR UNIDAD (€) | CANTIDAD | SUBTOTAL (€) |
|----------------------|-----------------------------|-----------------|---------------------|
| Papel A4 (500 hojas) | 4 | 1 | 4 |
| Tinta impresora | 20 | 2 | 40 |
| Encuadernación | 3 | 3 | 9 |
| CD | 0.5 | 2 | 1 |
| TOTAL (€) | | | 54 |

Tabla 23 Coste del material fungible

Por tanto el total correspondiente a la suma de los costes de equipo y el material fungible ascenderá a:

| Costes equipos (€) | Material fungible (€) | TOTAL (€) |
|---------------------------|------------------------------|------------------|
| 608,67 | 54 | 662,67 |

Tabla 24 Suma del coste de equipos con el material fungible

Por consiguiente, el coste total de los recursos materiales asciende a 662,67€.

7.2.3 Coste total

En este apartado se muestra el coste total del proyecto, correspondiente a la suma de los recursos humanos y los recursos materiales descritos anteriormente, a los que hay que añadir los costes indirectos, que se han considerado de un 5%. Además, a esta cantidad resultante se ha de sumar el I.V.A. equivalente al 18%.

| CONCEPTO | COSTE (€) |
|------------------------------------|------------------|
| Recursos humanos | 29.340 |
| Recursos materiales | 662,67 |
| Suma recursos humanos y materiales | 30.002,67 |
| Costes indirectos (5%) | 1.500,13 |
| Total sin I.V.A. | 31.502,80 |
| I.V.A. (18%) | 5.670,50 |
| Total con I.V.A. | 37.173,30 |

Tabla 25 Coste total del proyecto

Por lo tanto, el presupuesto total del proyecto asciende hasta la cantidad de TREINTA MIL CIENTO SETENTA Y TRES EUROS CON TREINTA CÉNTIMOS DE EURO.

ANEXOS

En este apartado se describen dos manuales. El *manual de usuario* permitirá orientar a los usuarios en su familiarización con la aplicación, mientras que el *manual de referencia* orientará a posteriores desarrolladores de la aplicación.

A. Manual de usuario

Con este manual se procede a realizar una explicación de la utilización de la aplicación a cualquier usuario con un mínimo conocimiento sobre dominios y problemas de planificación.

Para poder inicializar la aplicación es necesario buscar en la carpeta *PFC* el ejecutable “*proyecto.exe*”, y una vez encontrado pulsar dos veces de seguido con el ratón sobre él.

Nada más inicializar la aplicación el usuario se encontrará una interfaz gráfica tal y como muestra la ilustración 10 de esta memoria.

Por tanto hay cinco posibles opciones a realizar partiendo del menú que aparece en pantalla. Para acceder a cada una de las acciones es necesario o pulsar con el ratón sobre una de ellas, o pulsar la tecla “*Alt*” + la letra que se encuentre subrayada del nombre de las opciones.

En los siguientes apartados se detallan las cinco posibles opciones a realizar por el usuario en la aplicación.

A.1 Crear un dominio

Para poder realizar este subobjetivo hay que haber pulsado la opción “*Create a domain*”. Hay que recordar que una vez pulsada la opción del menú son desactivadas el resto salvo “*Close Program*”, por tanto, si el usuario quiere elegir otra opción del menú o salir de la misma, deberá pulsar el botón con la etiqueta “*Restart*” o la opción de menú “*Close Program*” para salir directamente de la aplicación.

Una vez pulsada la opción “*Create a domain*”, la pantalla principal desaparecerá y quedará sólo la consola, a través de la cual el usuario podrá crear el dominio.

El usuario debe guiarse por las instrucciones detalladas que aparecen según vaya introduciendo los datos hasta que finalmente se cree el dominio deseado. Estas instrucciones han sido creadas según los requisitos del sistema y permiten guiar al usuario a la realización del objetivo propuesto.

En primer lugar se le pedirá el nombre del dominio al usuario, el cual lo escribirá por teclado, y una vez escrito pulsará la tecla “INTRO”. A continuación se mostrará en la ilustración un ejemplo de cómo el usuario puede introducir un dominio.

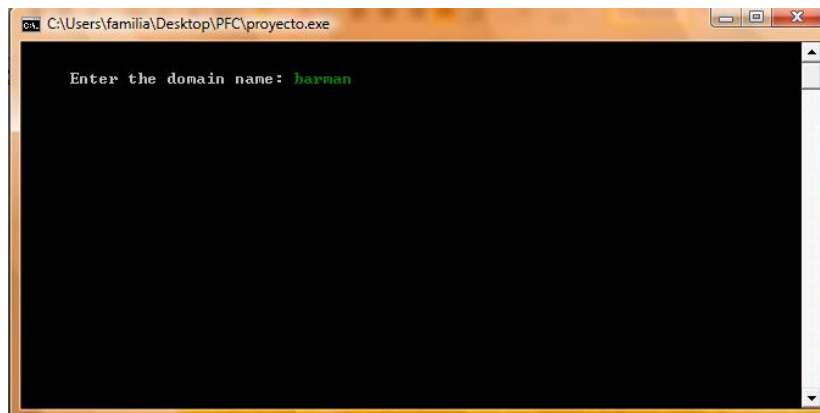


Ilustración 21 Introducción del dominio “barman”

Una vez introducido el nombre del dominio, el usuario tendrá que insertar los requisitos. Para ello, se le muestra una tabla con todos los posibles requisitos del sistema. El usuario para poder seleccionar un requisito deberá introducir su número correspondiente, y si quiere introducir más de uno, deberá separar los números mediante comas. Cuando ya los haya seleccionado, deberá pulsar la tecla “INTRO”. Partiendo de que el usuario quiere introducir por ejemplo los requisitos “:strips”, “:typing”, y “:action-costs”, en la ilustración siguiente, se podrá observar cómo sería la introducción:

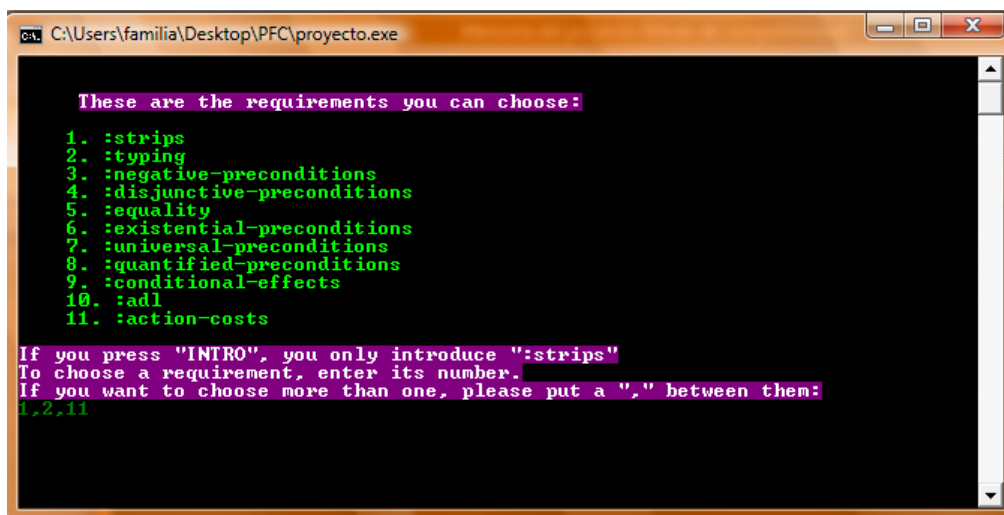
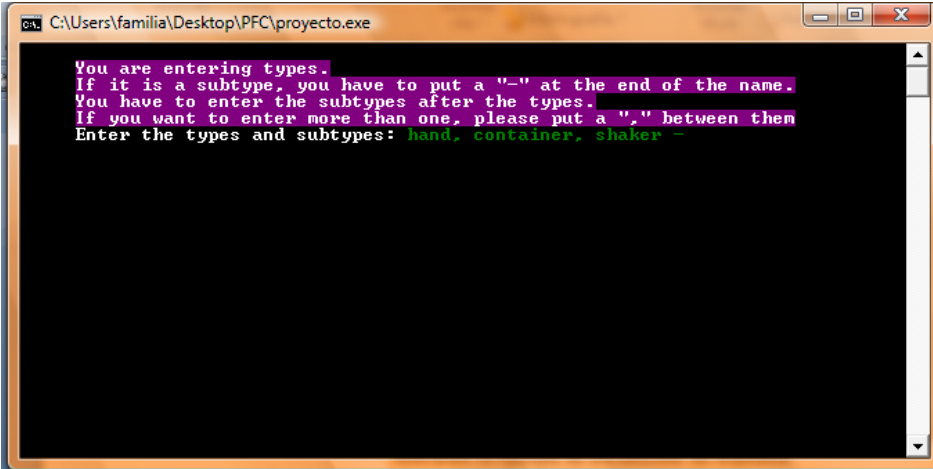


Ilustración 22 Ejemplo de introducción de requisitos

Una vez introducidos los requisitos hay que introducir los tipos y los subtipos. Para introducir un subtipo, se deberá poner el nombre del mismo, un espacio y a continuación un guión. En caso de que el usuario quiera introducir más de un tipo o un subtipo, deberá separarlos por comas.

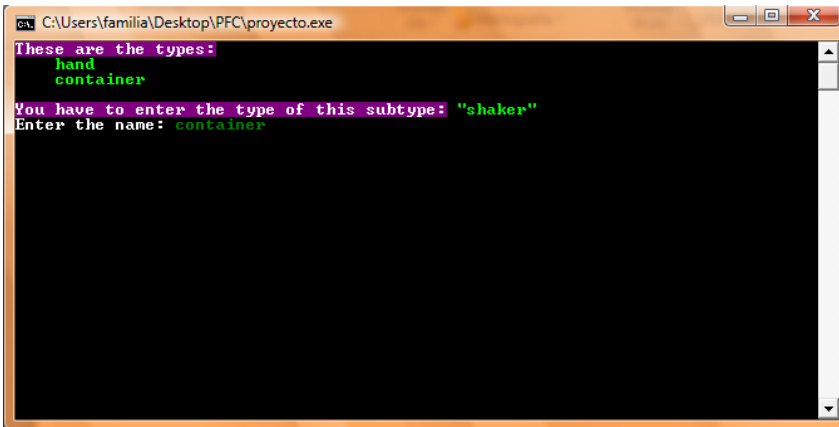
Un ejemplo es que el usuario quiere introducir los tipos *hand* y *container*, además del subtipo *shaker* que pertenece a *container*. En la ilustración se muestra su introducción:



```
C:\Users\familia\Desktop\PFC\proyecto.exe
You are entering types.
If it is a subtype, you have to put a "-" at the end of the name.
You have to enter the subtypes after the types.
If you want to enter more than one, please put a "." between them
Enter the types and subtypes: hand, container, shaker -
```

Ilustración 23 Ejemplo inserción tipos

Una vez pulsada la tecla “INTRO” aparecerá la lista de tipos a la que pueda pertenecer el subtipo *shaker*. Entonces se tendrá que introducir el tipo al que pertenece, en este caso *container*.



```
C:\Users\familia\Desktop\PFC\proyecto.exe
These are the types:
hand
container
You have to enter the type of this subtype: "shaker"
Enter the name: container
```

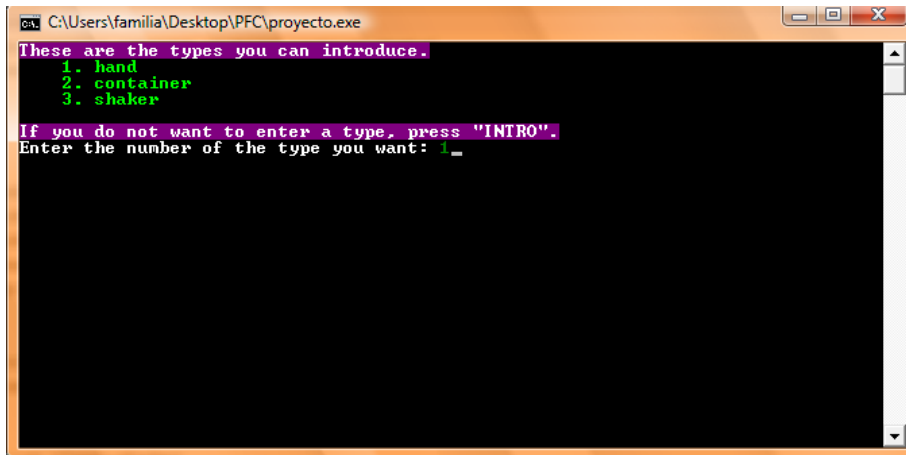
Ilustración 24 Ejemplo de asignación de tipo a subtipo

A continuación se le preguntará al usuario si desea introducir constantes. Si el usuario pulsa la tecla “c” es que quiere introducir alguna, mientras que si pulsa cualquier otra tecla es que no lo desea.

Si el usuario pulsa la tecla “c”, aparecerán los tipos y subtipos que se han introducido anteriormente, de tal forma que el usuario deberá elegir a cuál va a pertenecer esa o esas constantes que quiere introducir. Hay que mencionar que se van a insertar ordenadas por tipos, por tanto en caso de que el usuario quiera introducir dos constantes que no pertenezcan al mismo tipo, primero elegirá el tipo de la primera

constante y la introducirá, y para la siguiente hará los mismos pasos (no se pueden seleccionar al mismo tiempo dos tipos distintos).

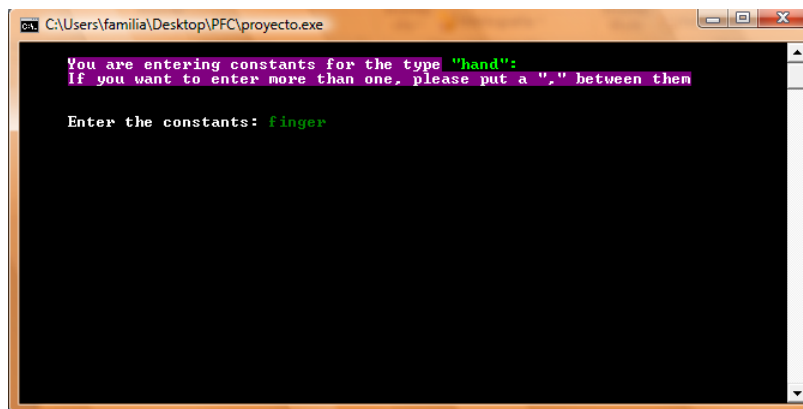
Para el ejemplo tratado, el usuario quiere introducir la constante “*finger*”. Por tanto, se ha pulsado la tecla “*c*”, y a la hora de la selección de tipos se introduce “1” que es el tipo al que pertenece:



```
C:\Users\familia\Desktop\PFC\proyecto.exe
These are the types you can introduce.
1. hand
2. container
3. shaker
If you do not want to enter a type, press "INTRO".
Enter the number of the type you want: 1_
```

Ilustración 25 Ejemplo de selección de tipo de la constante

En la siguiente ilustración se muestra cómo quedaría la pantalla una vez pulsada la tecla “*INTRO*”. Como en el ejemplo sólo se quiere insertar una constante del tipo *hand* no hace falta separar las constantes por comas.



```
C:\Users\familia\Desktop\PFC\proyecto.exe
You are entering constants for the type "hand":
If you want to enter more than one, please put a "," between them

Enter the constants: finger
```

Ilustración 26 Ejemplo de introducción de constante

Como ya no se requiere introducir más constantes, se pulsa la tecla “*INTRO*” sin introducir nada para pasar a los predicados tal y como indican las instrucciones de la consola.

En primer lugar se pedirá el nombre del predicado. En el ejemplo propuesto, el nombre del mismo será *holding*:

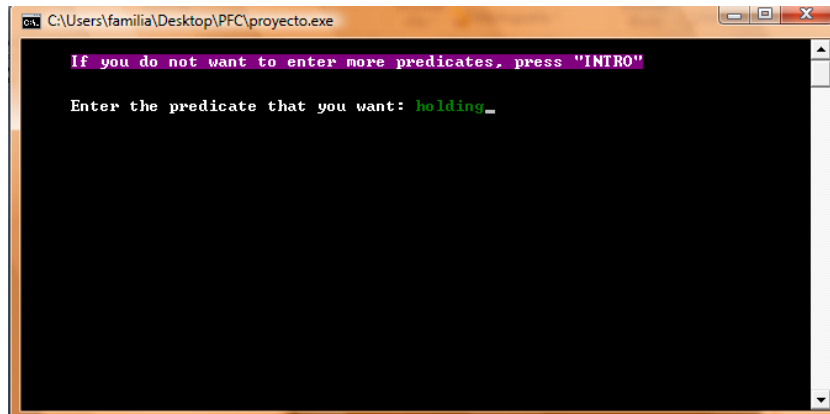


Ilustración 27 Introducción de nombre de predicado

A continuación se mostrarán los posibles tipos que puede elegir el usuario que formarán parte de ese predicado. El procedimiento será muy parecido al de la introducción de requisitos descrito anteriormente. Se mostrará la lista de tipos y el usuario introducirá el número correspondiente al tipo que desee, teniendo en cuenta que si quiere introducir más de uno deberá separar los números por comas. En caso de que se quiera repetir un tipo determinado, se introducirá el mismo número separado por comas tantas veces como se requiera.

En el ejemplo, el predicado *holding* estará compuesto por los tipos *hand* y *container*:

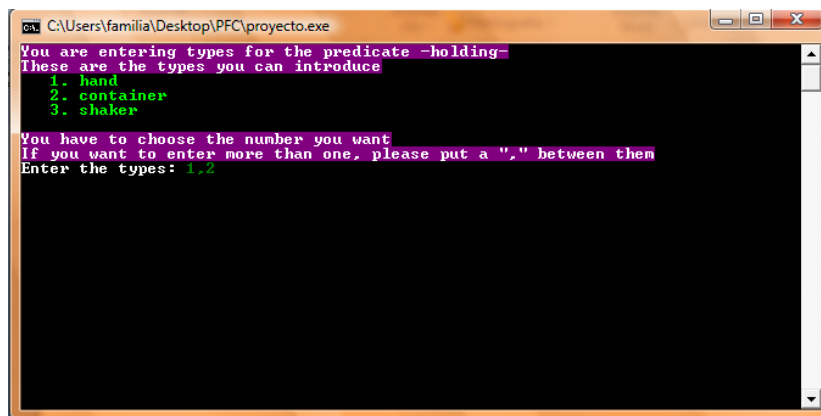


Ilustración 28 Ejemplo inserción tipos para predicado

Una vez pulsada la tecla “*INTRO*”, se procederá nuevamente a pedir el nombre de otro predicado, y así sucesivamente hasta que el usuario en lugar de introducir un nombre pulse la tecla “*INTRO*”, lo cual significa que se van a introducir las funciones.

Para la introducción de las funciones el procedimiento será exactamente el mismo que para los predicados, con la única diferencia lógicamente de que se trata de una función.

En el ejemplo, el usuario desea introducir la función *total-cost*, pero sin tipos. El procedimiento será el mismo que anteriormente para el predicado *holding*, con la diferencia de que a la hora de insertar los tipos se pulsará “*INTRO*” sin introducir nada, lo cual significa que esa función no tiene tipos que la componen. Como no se quieren introducir más funciones, se pulsará la tecla “*INTRO*” a la hora de introducir el nombre de la función, lo cual significa que el usuario no quiere introducir más funciones y se pasa a la introducción de acciones.

La introducción del nombre de las acciones se realiza de la misma forma explicada anteriormente para predicados y funciones. Una vez introducido el nombre de las acciones, se pasa a la introducción de parámetros, un proceso muy parecido al descrito anteriormente en la introducción de requisitos, con la diferencia de que esta vez sí se puede repetir el mismo número insertado.

A continuación se muestra el ejemplo de la introducción de parámetros. Se quieren introducir como parámetros dos de *hand* y uno de *shaker*:

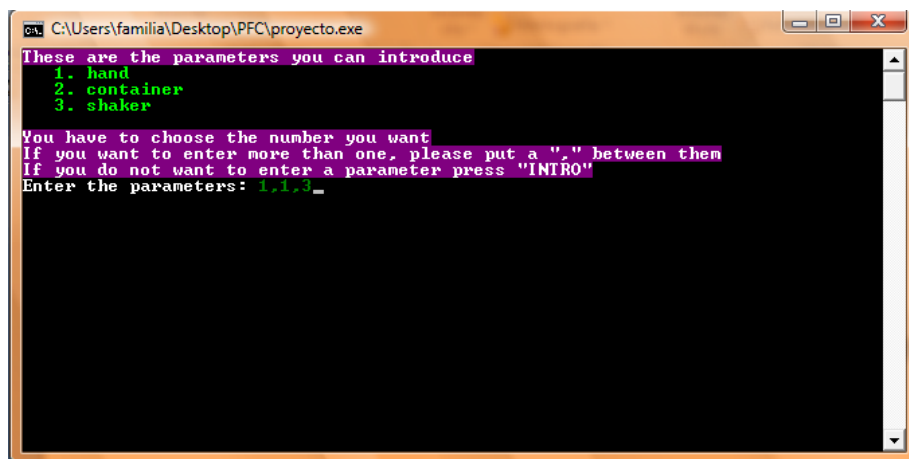


Ilustración 29 Ejemplo de introducción de parámetros para una acción

Una vez introducidos los parámetros, toca introducir la precondition de la misma. La precondition puede estar compuesta por una o más condiciones, por tanto lo primero es elegir el operador adecuado.

En el ejemplo, se quiere insertar una precondition compuesta por una sola condición, por la cual no se necesita operador ya que tampoco se encuentra negada. En tal caso, se pulsará en la pantalla la tecla “*INTRO*”. Tampoco se quieren insertar “fluents”, en consecuencia, se pulsa cualquier tecla menos la “*f*”, correspondiente a ese caso. Es hora de elegir el predicado (de la lista que se muestra por pantalla) que se quiere poner como condición, en este caso concreto *holding*. Por tanto se escribe el nombre del predicado y a continuación se pulsa la tecla “*INTRO*”.

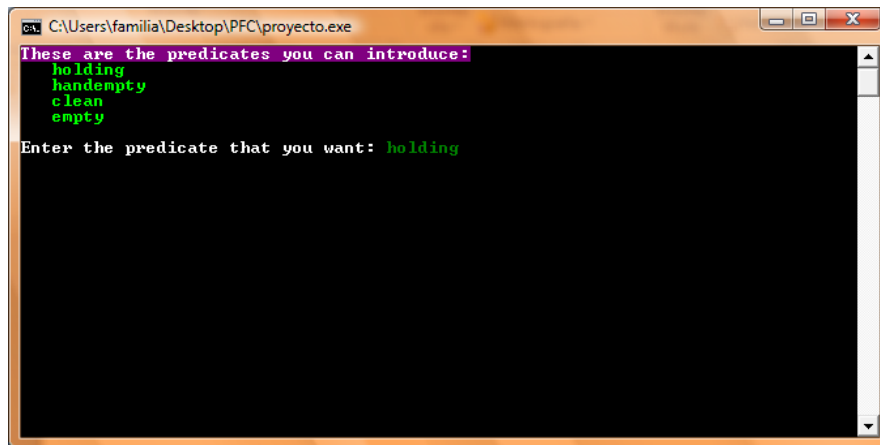


Ilustración 30 Ejemplo de selección de predicado

Una vez seleccionado el predicado, se pasa a introducir sus tipos correspondientes según los parámetros que han sido introducidos en la acción. Se debe tener en cuenta que las constantes también cuentan como parámetros, por tanto en el ejemplo dispuesto se van a considerar como tipos la constante *finger* perteneciente al tipo *hand* y el parámetro correspondiente a *shaker*. Se introducirán separados por comas, la constante con su nombre y para el parámetro deberá introducirse el identificador determinado, que aparece después del signo “?”.

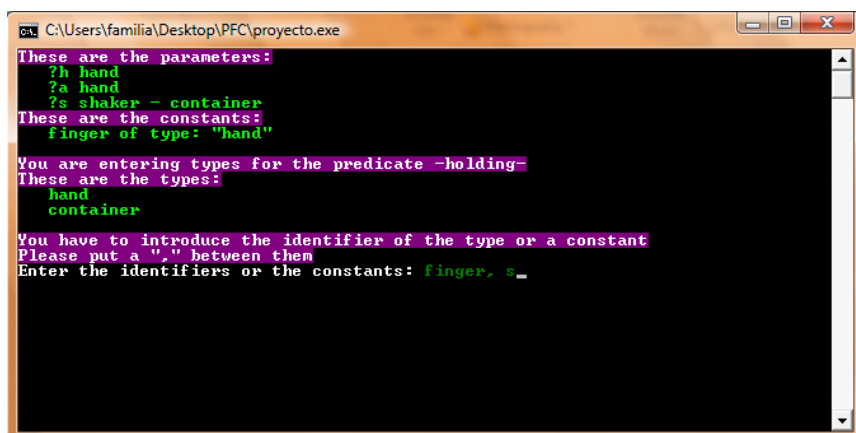


Ilustración 31 Ejemplo de introducción de tipos para predicado de una condición

Ya es hora de ir a los efectos. Su forma de inserción es la misma que las precondiciones, a diferencia de que no poseen los mismos operadores (en el apartado “Lenguaje PDDL” de la memoria se pueden observar los distintos operadores).

En este caso, el efecto va a estar compuesto por dos condiciones, por lo que se debe poner el operador “and”.


```
C:\Users\familia\Desktop\PFC\proyecto.exe

Remember the operators of a "effect" are:

-and
-not
-forall
-when

If you want to stop, press "i"
If you do not want to enter an operator, press "INTRO"
To choose an operator, enter its name:
and
```

Ilustración 32 Ejemplo de selección de operador de un efecto

A continuación habrá que introducir el número de condiciones por las que está compuesto, en este ejemplo, dos:

```
C:\Users\familia\Desktop\PFC\proyecto.exe

Enter the number of conditions you want to introduce: 2
```

Ilustración 33 Ejemplo de selección de condiciones

Como en este ejemplo la primera condición será el predicado *clean* con el parámetro correspondiente a *shaker* no hay que seleccionar operador, y el proceso de selección del tipo para el predicado sería el mismo explicado anteriormente para las precondiciones.

Pero la segunda condición del efecto de este ejemplo sí que no se ha visto anteriormente, que es incrementar la función *total-cost* en una unidad.

Después de haber pulsado la tecla “INTRO” cuando se pedía el operador, se muestra el siguiente mensaje, en el cual debemos pulsar del teclado la tecla “I”:

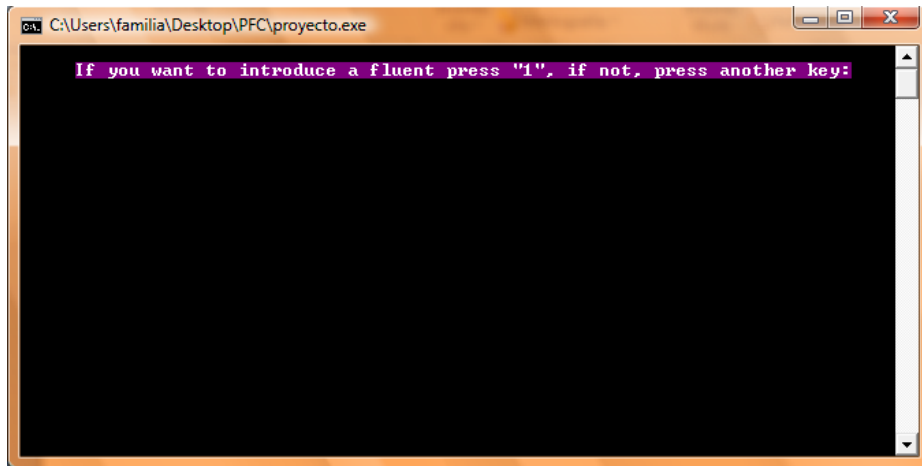


Ilustración 34 Pantalla que comunica al usuario qué debe hacer si quiere introducir un *fluent*

Tras pulsarla, nos encontramos un caso parecido a la introducción de requisitos, en la que se debe elegir el “fluent” determinado. En este ejemplo concreto, es *increase*.

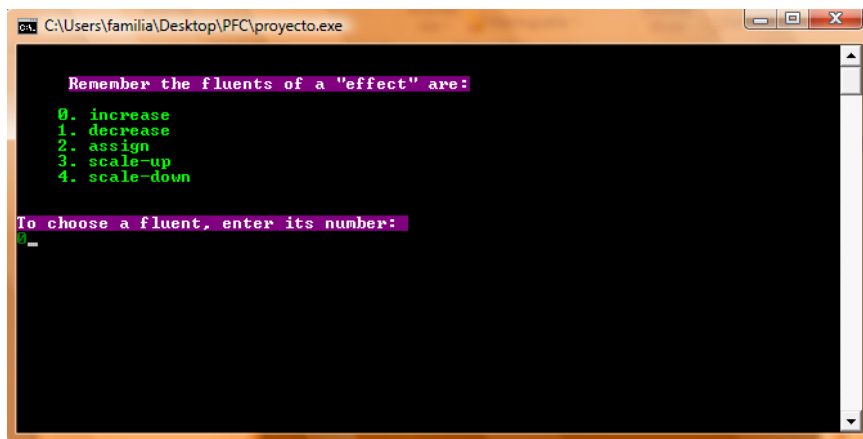


Ilustración 35 Ejemplo de selección de *fluent*

Tras pulsar la tecla “INTRO” se introduce la función que corresponda, en este caso *total-cost*. A continuación se pide que se introduzca u otra función (recordar que se pueden incrementar con el valor de otra función) o un número, y en este ejemplo era un número, concretamente una unidad.

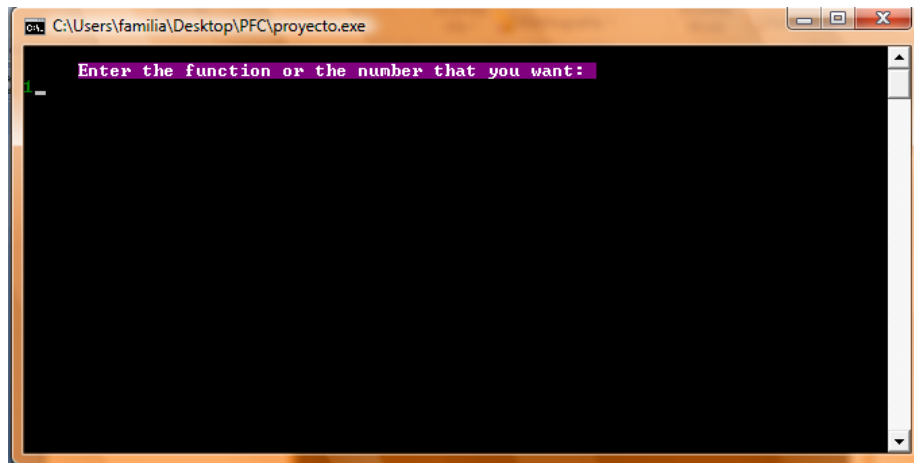


Ilustración 36 Ejemplo de introducción de valor para la función

Y tras pulsar la tecla “*INTRO*” habrá concluido el proceso de introducir la acción determinada.

Si se quisiera introducir otra acción se deberían seguir los pasos descritos anteriormente desde la introducción de su nombre.

Para acabar con la creación del fichero, en lugar de introducir un nombre para la acción, se pulsa la tecla “*INTRO*”, y aparecerá un mensaje en la pantalla advirtiendo al usuario que se ha creado correctamente el dominio y por consiguiente almacenado en el fichero “domain.pddl” tal y como los requisitos lo requieren.

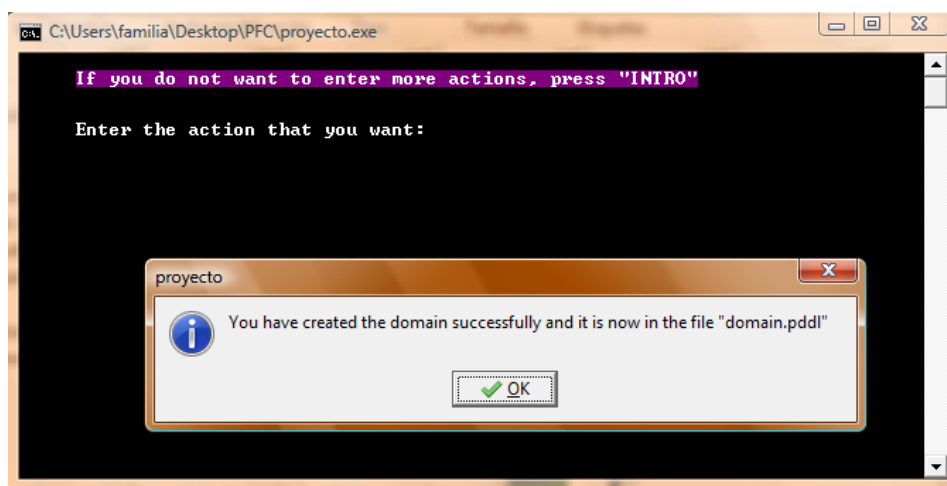


Ilustración 37 Mensaje de creación del dominio y del fichero satisfactoriamente

Una vez pulsado “*OK*”, vuelve a mostrarse otra vez la pantalla del formulario de nuevo con el menú y sus opciones activadas.

A.2 Leer un dominio

Para poder realizar este subobjetivo, se debe haber seleccionado la opción del menú “*Read a domain*”. Hay que recordar que una vez pulsada la opción del menú son desactivadas el resto salvo “*Close Program*”, por tanto, si el usuario quiere elegir otra opción o salir de la misma, deberá pulsar el botón con la etiqueta “*Restart*” o la opción de menú “*Close Program*” para salir directamente de la aplicación.

En primer lugar, el usuario deberá saber el directorio en el que va a trabajar. Por defecto el directorio será la carpeta donde se encuentra el ejecutable que lanza la aplicación, en nuestro caso, la carpeta *PFC*. Por tanto, si quiere cambiar de directorio para la lectura de los dominios, deberá pulsar el botón la etiqueta “*Click to change folder*”, y aparecerán los posibles directorios a través de los cuales puede trabajar, y tendrá que seleccionar el que desee. Para facilitar al usuario su interacción con la aplicación, se mostrará siempre el directorio actual donde se encuentra.

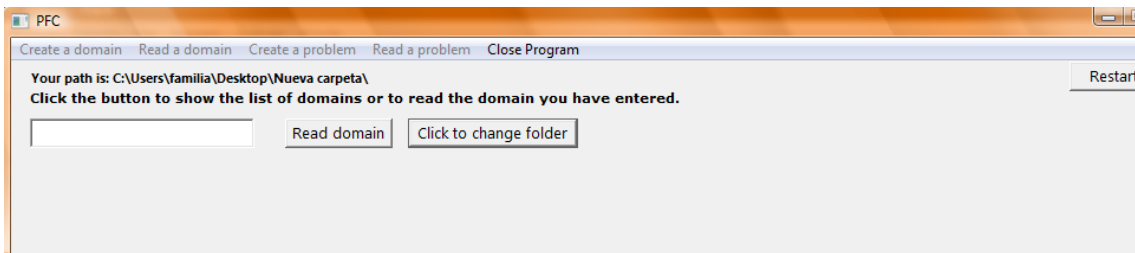


Ilustración 38 Lectura de dominio: Muestra del nuevo directorio tras haber cambiado de carpeta

El usuario deberá introducir en el “*edit*” (la caja que aparece en pantalla que se muestra a continuación) el nombre de uno de los dominios existentes en la carpeta.

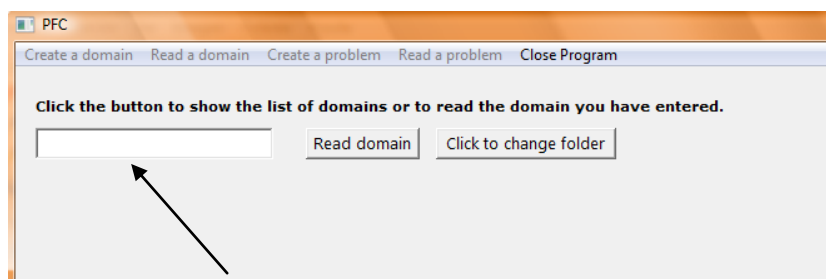


Ilustración 39 Lectura de dominio: “Edit” donde el usuario introducirá el dominio

En caso de que el usuario no conozca los dominios existentes en la carpeta, puede pulsar directamente el botón con la etiqueta “*Read domain*”, y se mostrará una lista con los dominios existentes en el directorio donde se encuentre.

Siempre que se pulse el botón con la etiqueta “*Read domain*”, y el dominio introducido no exista en la carpeta, también aparecerá la lista mencionada anteriormente.

Para mayor comodidad del usuario, una vez que aparezca la mencionada lista, el usuario podrá pulsar con el ratón situado encima del dominio que desee y automáticamente aparecerá escrito el dominio, para posteriormente pulsar de nuevo el botón anterior (si no lo desea, puede escribir el dominio en el “*edit*” sin problemas).

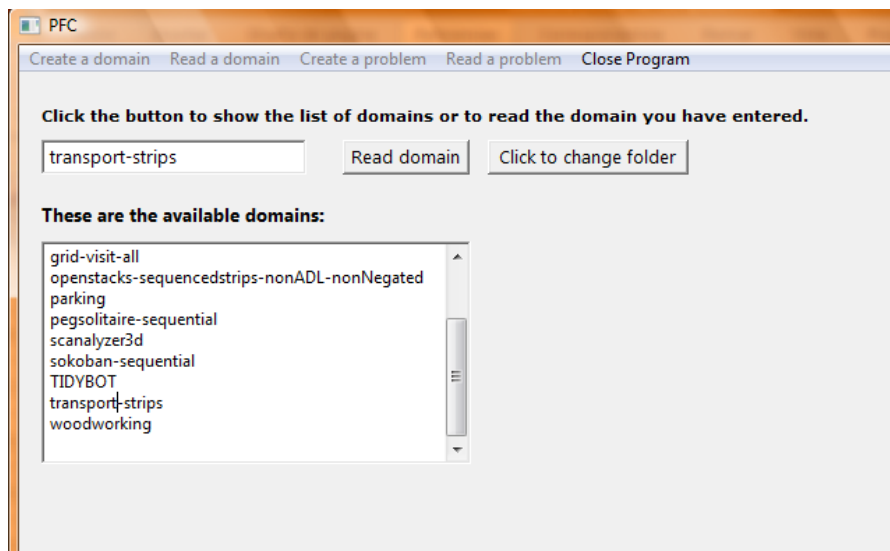


Ilustración 40 Lectura de dominio: El usuario pulsando sobre el dominio también lo puede añadir

Una vez pulsado el botón y en caso de que el dominio exista, se mostrará por pantalla. En la parte izquierda se puede observar que se ha añadido una leyenda de acuerdo con los requisitos de usuario para que el usuario pueda asimilar los conceptos representados. En la parte central-derecha, se mostrarán de acuerdo con la leyenda los tipos, constantes, predicados y funciones del dominio determinado.

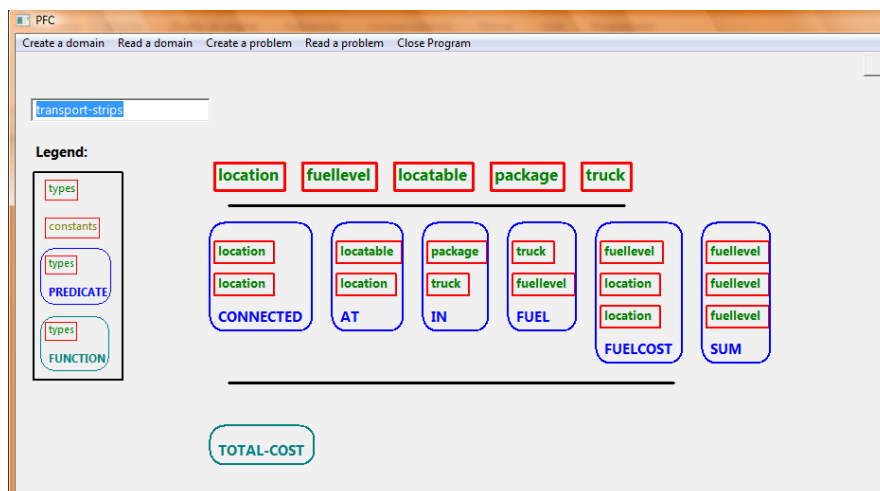


Ilustración 41 Lectura de dominio: La leyenda se muestra en la izquierda y el dominio en la parte central derecha

A.3 Crear un problema de planificación

Para poder realizar este subobjetivo, se debe haber seleccionado la opción del menú “*Create a problem*”. Hay que recordar que una vez pulsada la opción del menú son desactivadas el resto salvo “*Close Program*”, por tanto, si el usuario quiere elegir otra opción o salir de la misma, deberá pulsar el botón con la etiqueta “*Restart*” o la opción de menú “*Close Program*” para salir directamente de la aplicación.

Ahora el usuario se encuentra con dos posibilidades:

- Si previamente ha pulsado la opción “*Read a domain*” y no ha pulsado el botón “*Restart*” ni ha cerrado la aplicación, automáticamente se creará un problema en base al dominio de planificación que se ha leído.
- Por el contrario, si no ha sido leído ningún dominio, será necesaria su introducción, de la misma forma que en la opción previamente explicada en el apartado A.2. También es posible cambiar el directorio actual por otro que desee el usuario.

El siguiente paso es seleccionar los objetos que contendrá el problema. Los posibles tipos y constantes se mostrarán en la parte izquierda de la pantalla, debajo de la leyenda. Para seleccionar un tipo o constante, el usuario simplemente deberá pinchar sobre el que desee, y aparecerá representado gráficamente en la parte derecha de la imagen. A continuación se muestra una ilustración de cómo aparecerán representados gráficamente los tipos y constantes seleccionadas:

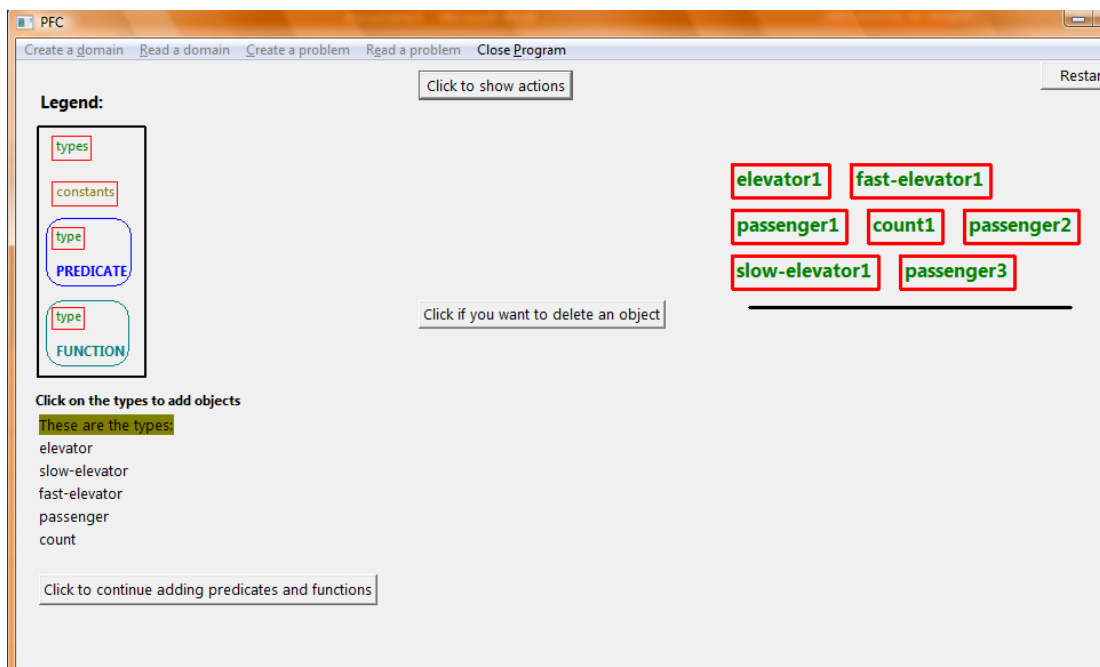


Ilustración 42 Creación de problema: Representación de los tipos seleccionados

Además, se pueden observar en la pantalla dos nuevos botones. El situado en la parte central superior con la etiqueta “*Click to show actions*” al ser pulsado muestra las acciones del dominio del que se está creando el problema. Una vez que se muestran, la etiqueta del botón pasa a ser “*Click to hide actions*”, y si se pulsa el botón las acciones ya no se muestran.

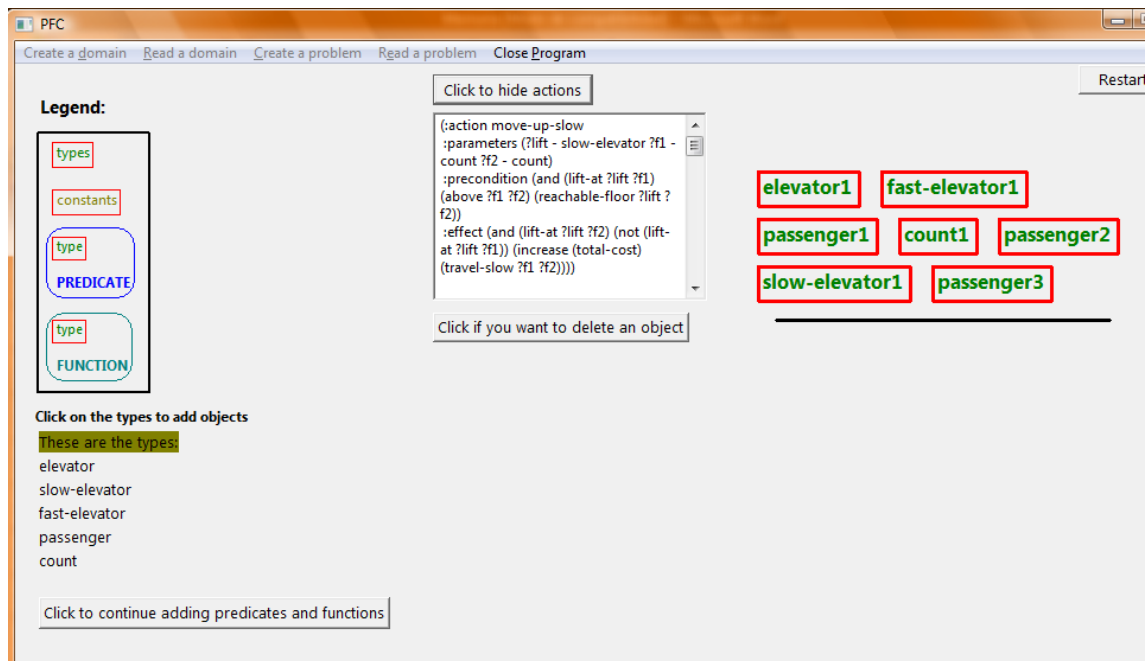


Ilustración 43 Creación de problema: Muestra de acciones

El otro botón que aparece tiene la etiqueta “*Click if you want to delete an object*” que al ser pulsado muestra los objetos que han sido seleccionados para el problema final en un “checkbox”, de tal forma que puedan ser seleccionados.

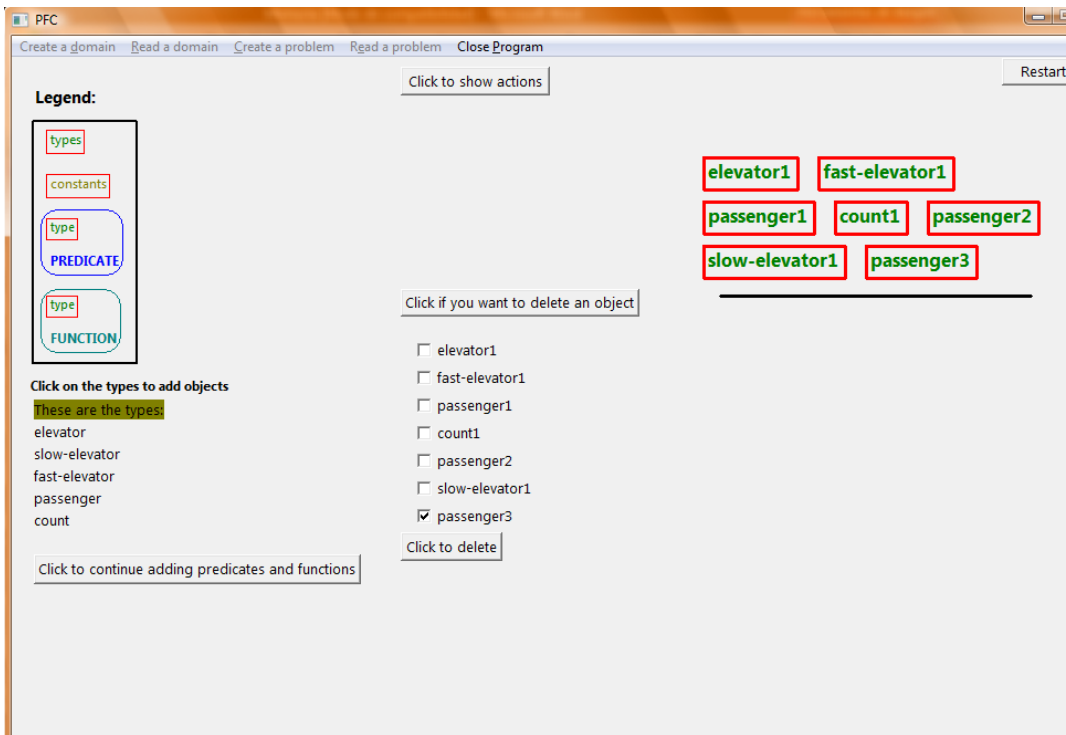


Ilustración 44 Creación de problema: CheckBox de los objetos seleccionados

Aparece también otro botón con la etiqueta “Click to delete”. Al pulsarlo los objetos que ha seleccionado el usuario serán borrados del problema, y por tanto ya no serán representados. Además, la “checkbox” ya no se muestra hasta que el usuario vuelva a pulsar el botón.

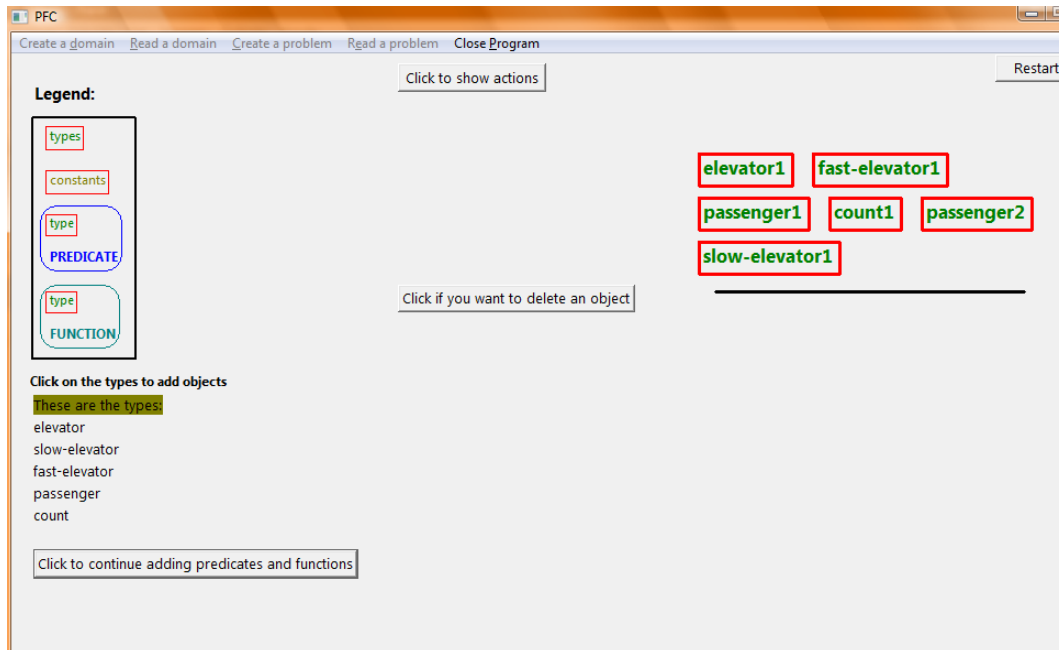


Ilustración 45 Creación de problema: Resultado de pulsar el botón para borrar

El usuario repetirá estos pasos hasta que finalmente haya introducido los objetos necesarios para el problema. Cuando así sea, deberá pulsar el botón que aparece debajo de los tipos y constantes del dominio, con la etiqueta “*Click to continue adding predicates and functions*”.

Una vez pulsado este botón, ya no se mostrarán los tipos y constantes del dominio, ahora se mostrarán los predicados y funciones del dominio. Al lado de cada nombre (tanto de los predicados como de las funciones) se muestran los tipos por los que está formado el predicado o función entre paréntesis, para facilitar al usuario a la hora de su elección. En la ilustración siguiente se muestra un ejemplo de cómo quedaría la pantalla:

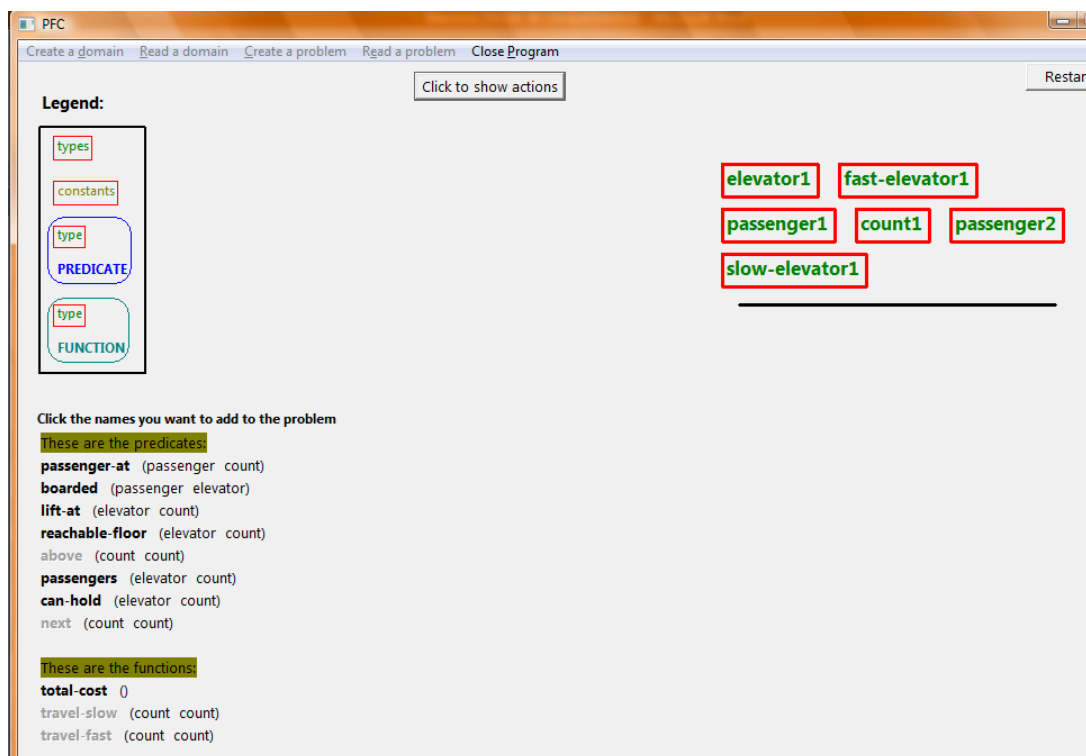


Ilustración 46 Creación de problema: Muestra de predicados y funciones del dominio

Hay que observar que algunos vienen en negrita y otros no. Los que se encuentran en negrita son los predicados y funciones que se pueden añadir al problema, ya que con los objetos seleccionados para el problema previamente se pueden completar los tipos que los componen. Por tanto los que no están en negrita, y además se encuentran desactivados, son aquellos que no poseen objetos del problema para formar parte de sus tipos.

Para poder añadir un predicado o función al estado inicial del problema el usuario pinchará sobre su nombre en negrita, y en el centro de la pantalla aparecerá una “checkbox” con los posibles objetos que han sido seleccionados previamente para el problema y que se pueden seleccionar para formar parte de sus tipos.

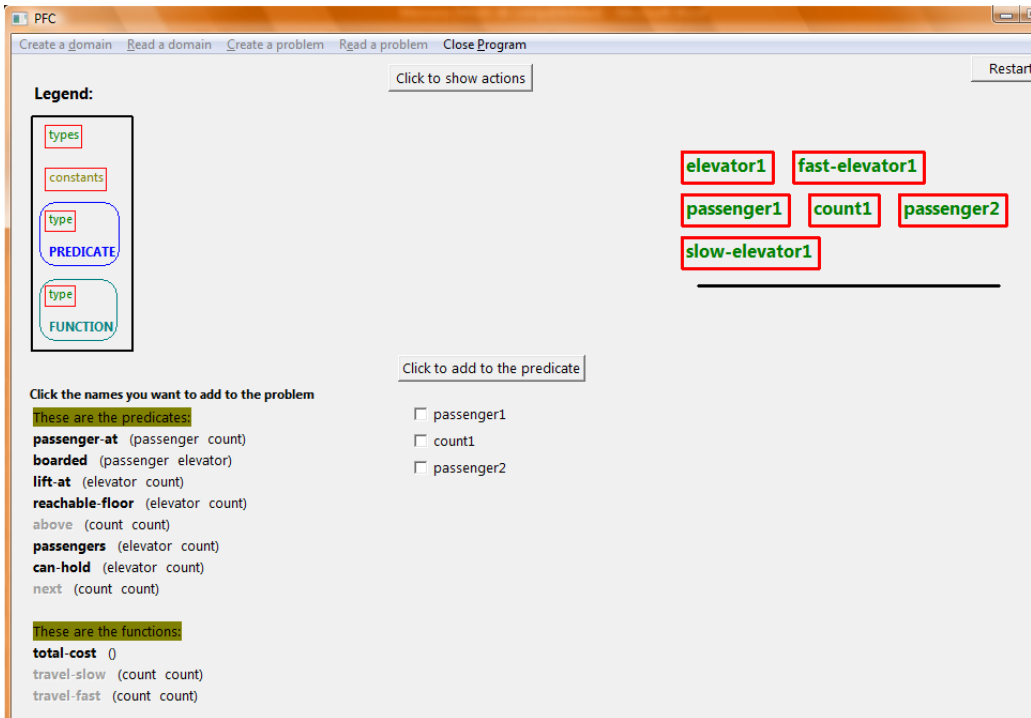


Ilustración 47 Creación de problema: Situación pulsar sobre el predicado "passenger-at"

Tras haber pulsado el botón con la etiqueta “*Click to add to the predicate*” los objetos seleccionados se añaden al predicado, y se representa debajo de los objetos del problema en la parte derecha de la pantalla. Hay que recordar que en caso de seleccionar más objetos para un tipo determinado del predicado, se almacenarán en el predicado por orden de aparición de los seleccionados en el “checkbox”.

A su vez, aparecen dos nuevos botones con las etiquetas “*Click to delete a predicate or a function*” y “*Click if you want to start with the goal*” respectivamente. En caso de pulsar el primer botón, aparecerá un “checkbox” que contendrá los nombres de los predicados y funciones ordenados según su representación gráfica, y podrán ser seleccionados para ser borrados del problema. Para ello, aparece un nuevo botón con la etiqueta “*Click to delete*”, que al ser pulsado borrará del problema todos los predicados y funciones que hayan sido seleccionados en la “checkbox”. El procedimiento para su borrado es igual que para el borrado de los objetos, es decir, desaparecerá también su representación gráfica.

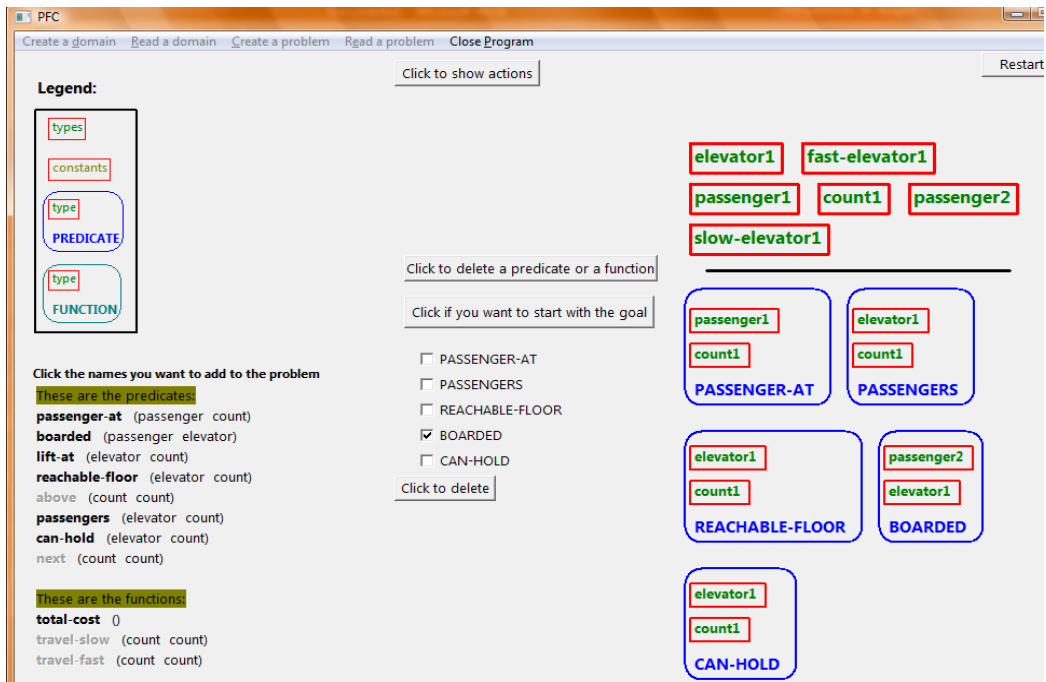


Ilustración 48 Creación de problema: Situación al pulsar el botón de borrar un predicado o una función

Para pasar a introducir los predicados y funciones para el estado meta, se debe pulsar el segundo botón mencionado anteriormente con la etiqueta “*Click if you want to start with the goal*”. Una vez pulsado, se pasan a seleccionar los predicados y funciones para el estado final del problema. Para seleccionarlos, se realizará el mismo proceso descrito anteriormente para el estado inicial.

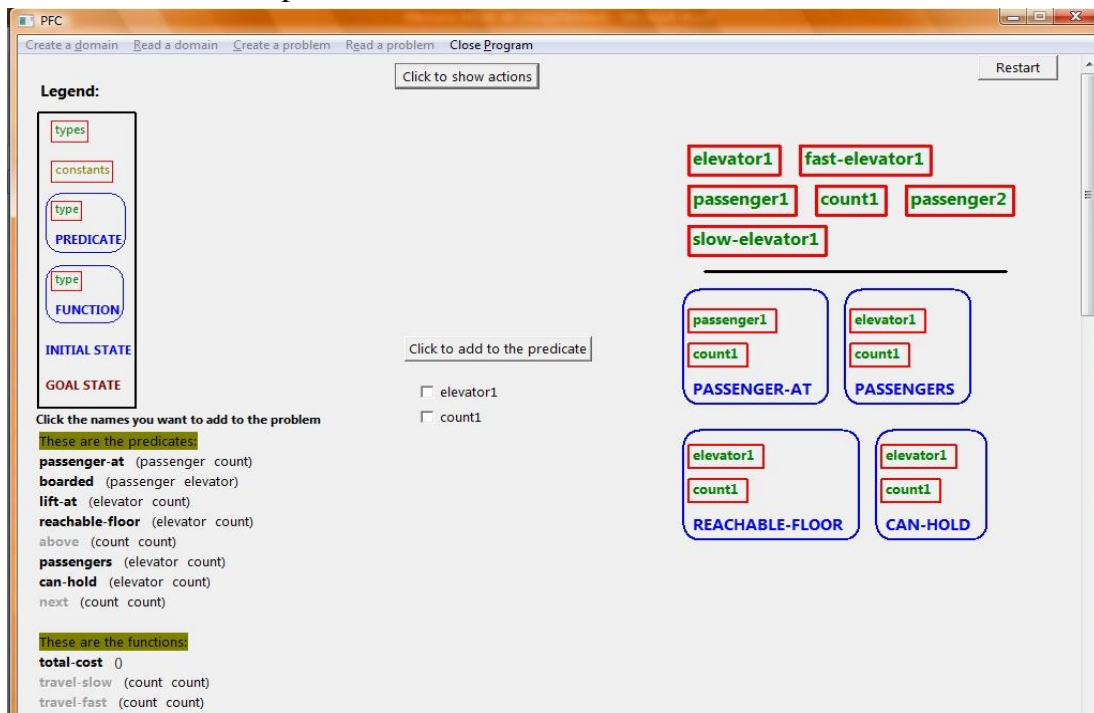


Ilustración 49 Creación de problema: Situación tras haber pulsado el botón del estado meta y haber seleccionado el predicado "lift-at"

También hay que destacar que la leyenda ahora muestra que los predicados y funciones del estado inicial se mostrarán en color azul, mientras que los del estado final se mostrarán en marrón.

Una vez seleccionado al menos un predicado o función para el estado final, aparecen dos nuevos botones con la etiquetas “Click to delete” y “Click to finish” respectivamente. En caso de pulsarse el primer botón, se produce el mismo proceso que el descrito anteriormente para el borrado de predicados y funciones del estado inicial del problema, pero en este caso para los predicados y funciones seleccionados para el estado final.

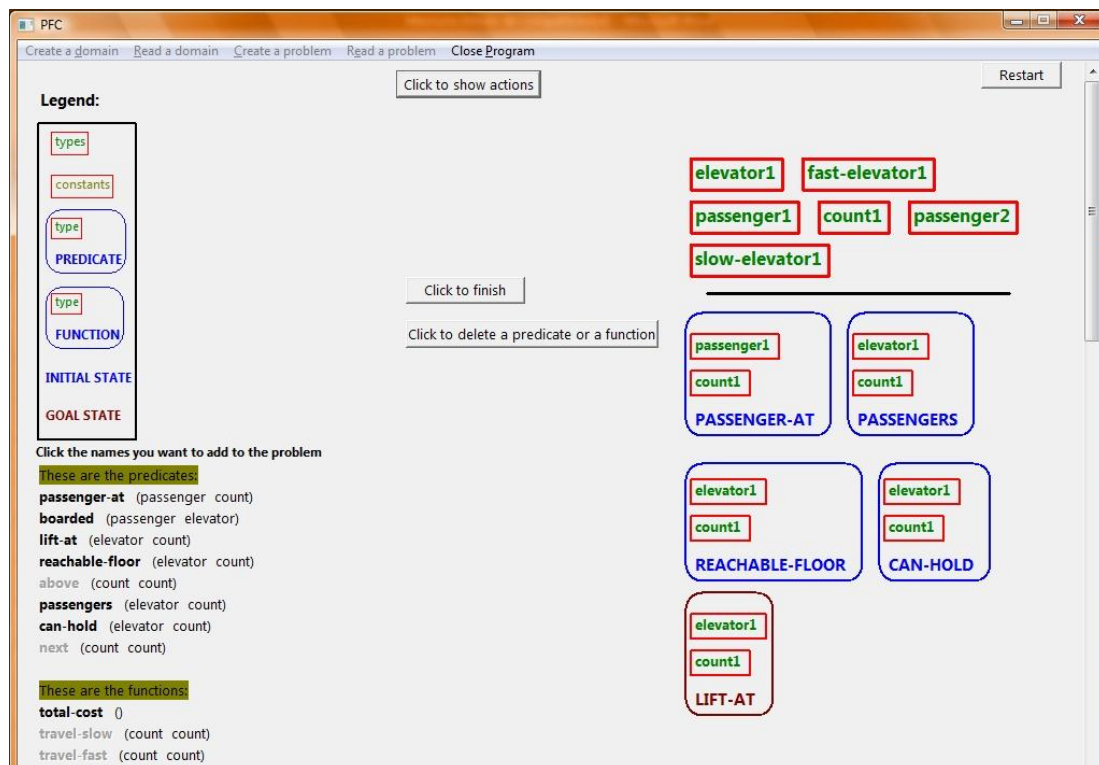


Ilustración 50 Creación de problema: Botón con la etiqueta "Click to finish" tras haber insertado el predicado "lift-at"

Al ser pulsado el segundo botón mencionado anteriormente, se pide la métrica del problema de planificación que se está creando en un “edit” (un ejemplo de “edit” fue explicado en el apartado A.2 de la memoria). Una vez escrita la métrica en el “edit” deberá pulsar en el botón con la etiqueta “Insert”, de tal forma que si no la ha introducido correctamente se mostrará un mensaje por pantalla al usuario sobre su error y se pedirá que la vuelva a introducir de nuevo.

Después de haber insertado la métrica correctamente se pasa a pedir el nombre del problema de planificación, con el mismo procedimiento que para la métrica.

Finalmente, se mostrará al usuario un mensaje sobre la creación satisfactoria del problema de planificación y su almacenamiento en un fichero con el mismo nombre que

el problema pero con la extensión “.pddl” tal y como se acordó en los requisitos de usuario.

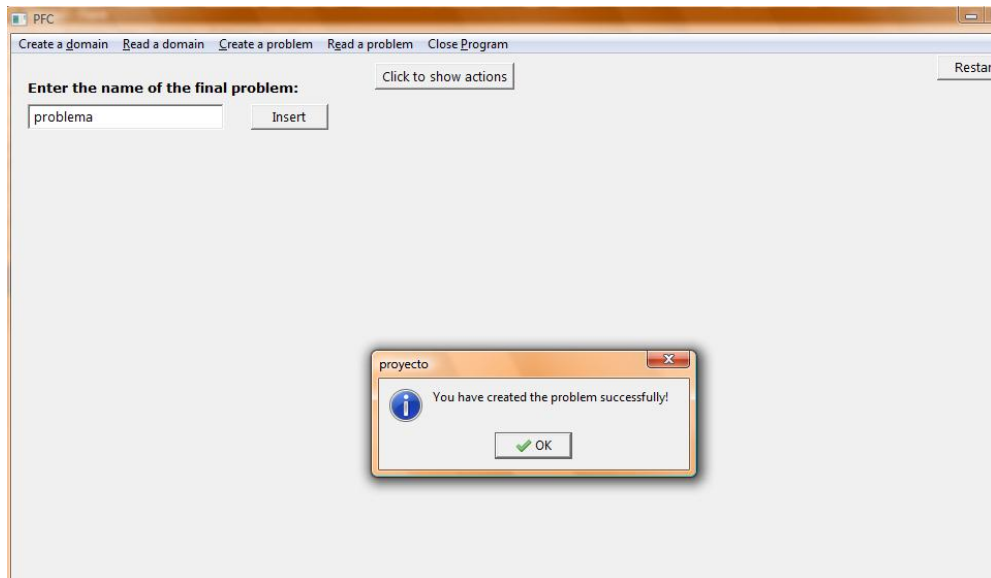


Ilustración 51 Creación de problema: Mensaje de creación satisfactoria del problema

Una vez pulsado “OK” del mismo o habiendo pulsado la tecla “INTRO”, se volverá al menú principal con las opciones del menú nuevamente activadas.

A.4 Leer un problema de planificación

Para poder realizar este subobjetivo, se debe haber seleccionado la opción del menú “*Read a problem*”. Hay que recordar que una vez pulsada la opción del menú son desactivadas el resto salvo “*Close Program*”, por tanto, si el usuario quiere elegir otra opción o salir de la misma, deberá pulsar el botón con la etiqueta “*Restart*” o la opción de menú “*Close Program*” para salir directamente de la aplicación.

Hay que tener en cuenta que el directorio por defecto es la carpeta en la que esté el ejecutable, pero es posible que el usuario elija otro. Para ello, deberá pulsar el botón con la etiqueta “*Click to change folder*”, y una vez pulsado deberá seleccionar el nuevo directorio, el cual será mostrado a continuación para que pueda conocer cual ha elegido para facilitarle su interacción con la aplicación.

El usuario deberá introducir en el “edit” (en la ilustración 39 se muestra un ejemplo) el nombre del fichero existente en la carpeta que almacena el problema que quiere ser leído.

En caso de que el usuario no conozca los ficheros que contienen problemas de planificación de la carpeta, puede pulsar directamente el botón con la etiqueta “*Read file*”, y se mostrará una lista con los ficheros existentes en el directorio actual que contienen un problema de planificación.

Siempre que se pulse el botón con la etiqueta “*Read file*”, y el nombre del fichero introducido no exista en la carpeta, también aparecerá la lista mencionada anteriormente.

Para mayor comodidad, una vez que aparezca la mencionada lista, el usuario podrá pulsar con el ratón situado encima del nombre del fichero que desee y automáticamente aparecerá escrito ese nombre, para posteriormente pulsar de nuevo el botón anterior (si no lo desea, puede escribir el nombre del fichero sin problemas).

Una vez pulsado el botón y en caso de que el fichero exista, se mostrará por pantalla. En la parte izquierda se puede observar que se ha añadido una leyenda de acuerdo con los requisitos de usuario para que el usuario pueda asimilar los conceptos representados. En la parte central-derecha, se mostrarán de acuerdo con la leyenda los objetos, estado inicial y estado final del problema de planificación determinado.

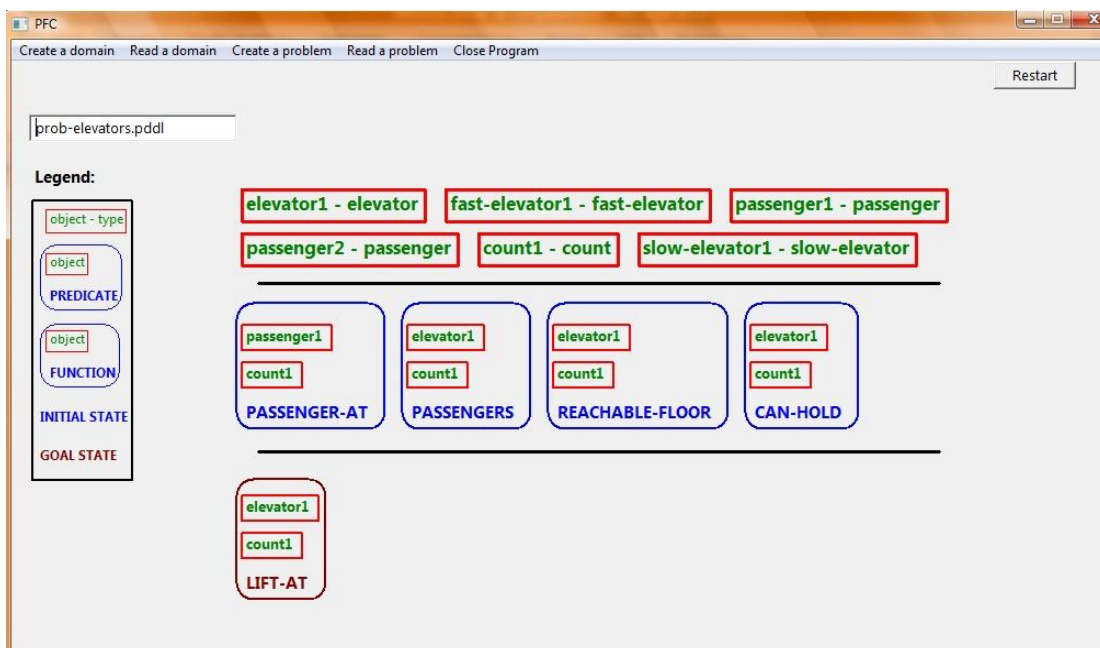


Ilustración 52 Lectura de problema: En la parte izquierda se muestra la leyenda y en la parte central-derecha el problema

B. Manual de referencia

Con este manual se procede a realizar una explicación técnica de la aplicación, para permitir a un desarrollador ampliar la aplicación posteriormente.

B.1. Elementos necesarios

Para el correcto funcionamiento de la aplicación son necesarios los siguientes elementos en la carpeta donde se encontrará la aplicación, como se ha mencionado anteriormente denominada *PFC*:

- El ejecutable “*proyecto.exe*”. Sin él no sería posible lanzar la aplicación.

Además del mencionado, dependiendo de las opciones que elija el usuario podrá ser o no necesaria la existencia de ficheros que contengan o un dominio o un problema de planificación.

B.2. Programa utilizado

En primer lugar hay que recordar que para haber podido realizar el objetivo del proyecto hay que trabajar principalmente con ficheros y gráficos por pantalla. Debido a la complejidad de programar con los mismos, se han tenido que valorar los programas y lenguajes de programación estudiados durante la carrera para tener un mayor conocimiento del mismo y poder llevar a cabo la realización del proyecto lo antes posible.

Como el lenguaje de programación estudiado durante la realización de la carrera ha sido el lenguaje *Object Pascal* en el entorno *Delphi7*, se decidió realizar el proyecto en este lenguaje y entorno.

En un principio no surgieron problemas a la hora de la realización, pero a la hora de empezar a trabajar con los gráficos, por problemas de licencia, no se podía realizar todo lo deseado correctamente mediante *Delphi7*.

Debido a este motivo, se empezó a valorar el cambio a un entorno de desarrollo que no tuviera problemas de licencia y que pudiera tener una continuación del código realizado hasta este momento. Dentro de esta valoración, surgió el entorno de desarrollo *Lazarus*, y finalmente, tras una reunión con el tutor, se decidió trasladar el código al mismo y realizar el resto de la implementación del proyecto en el mismo, que se describe brevemente a continuación.

Lazarus [8] es una herramienta de desarrollo rápido de aplicaciones (RAD) que como previamente se ha dicho se basa en el lenguaje de programación *Object Pascal*.

Es compatible con los sistemas operativos Windows, GNU/Linux y Mac OS X. Para la realización de este proyecto ha sido utilizado el sistema operativo Windows Vista.

Lazarus es un sistema de desarrollo de código abierto que trabaja sobre el compilador *Free Pascal* (FPC) agregando un entorno integrado de desarrollo (IDE) que incluye un editor de código con resalte de sintaxis y un diseñador de formulario visual, además de una librería de componentes altamente compatible con la librería de componentes visual de Delphi (VCL).

Se ha comentado anteriormente que se trabaja sobre el compilador *Free Pascal* (FPC) que es un compilador de Pascal de código abierto que también tiene un alto grado de compatibilidad con Delphi, (no sólo la ayuda para el mismo lenguaje de programación, sino también para muchas bibliotecas de rutinas y de clases por las que Delphi es conocido) y está disponible para gran variedad de plataformas, tal y como se dijo previamente.

Por tanto *Free Pascal* y *Lazarus* están escritos en *Pascal*, y ambas son herramientas de uso general, lo que significa que se pueden desarrollar una amplia variedad de programas con ellos, como se describe a continuación:

- **Aplicación de consola:** No poseen interfaces gráficas de usuario (GUI). Sólo se lanza la consola, leen su entrada en la consola y escriben su salida en la consola (esta ha sido la aplicación de consola utilizada en el proyecto). También pueden ser utilizadas por los programas de procesos de datos que no necesitan una GUI porque son arrancados por otros programas o desde archivos por lotes, sin embargo no se ha profundizado en el proyecto en este aspecto.
- **Librerías cargables dinámicamente (dll):** son generalmente una colección de funciones compiladas que se pueden llamar por otros programas. No se han utilizado para la realización de este proyecto.
- **Aplicaciones con interfaces gráficas de usuario (GUI):** Al desarrollar con GUI desde *Lazarus*, se crean no solamente unidades en código Pascal, también se diseñan los formularios que contienen controles visuales tales como botones o cajas de lista, y el diseño del formulario se hace visualmente. El proyecto se basa prácticamente en una GUI.

En conclusión, se puede decir que *Lazarus* es una alternativa libre y gratuita a Delphi, motivo por el cual finalmente ha sido elegido para la realización de la implementación del proyecto.

C. Unidades utilizadas

En este apartado se describirán brevemente las unidades utilizadas para la implementación del proyecto.

- “*Unit1.pas*”: Es la unidad principal. En ella se trata el formulario general, y por tanto está compuesta por todos los procedimientos y funciones que están relacionados con el mismo.
- “*Comprobaciones.pas*”: En esta unidad se encuentran procedimientos y funciones que se encargan de realizar comprobaciones y tareas que son necesarias a lo largo del programa, como por ejemplo comprobar si un nombre es permitido, se encuentra un comentario en un fichero, quitar los espacios de una línea, etc.
- “*Lists.pas*”: En esta unidad se encuentra la estructura para poder almacenar una lista, así como los predicados y funciones que se encargan de tratar con las mismas.
- “*Ficheros.pas*”: En esta unidad se encuentran los procedimientos que se encargan de escribir en los ficheros la información almacenada en memoria.
- “*LeePartes.pas*”: Esta unidad tiene algunos procedimientos y funciones para la lectura de datos concretos, como por ejemplo leer el nombre del problema, leer el dominio del fichero problema, leer la métrica, etc.
- “*Tipos.pas*”: Esta unidad contiene los procedimientos y funciones necesarios para el tratamiento de los tipos y constantes.
- “*Acciones.pas*”: En esta unidad se encuentran los procedimientos y funciones relacionados con la creación, tratamiento, y almacenamiento de acciones. Además, en ella se encuentra declarada la estructura de una acción, que estará compuesta por unos parámetros, unas precondiciones y unos efectos.
- “*ListPredFunc.pas*”: En esta unidad se encuentran los procedimientos y funciones que tratan con los predicados y funciones del programa, además de contar con la estructura para poder almacenar los mismos.
- “*Mensajes.pas*”: En esta unidad se encuentran los procedimientos y funciones encargados de mostrar los mensajes por consola.
- “*Operadores.pas*”: Esta unidad contiene la estructura para poder almacenar y tratar las precondiciones y efectos que puede tener una acción. Además también contiene los procedimientos y funciones que se encargarán de su tratamiento.
- “*Requirements.pas*”: Además de contener la estructura para poder almacenar los requisitos y poseer los requisitos permitidos que puede almacenar el programa, esta unidad contiene los procedimientos y funciones que se encargan del tratamiento de los requisitos.

- “*LectComp.pas*”: Esta unidad contiene los procedimientos y funciones que permiten el tratamiento de las cadenas, como por ejemplo saber la posición del siguiente paréntesis de apertura o de cierre, devolver los paréntesis que contiene, devolver una palabra determinada de la cadena, etc.

REFERENCIAS

En este apartado se definen las referencias utilizadas a la hora de la realización del proyecto así como de la documentación.

[1] Descripción de la Inteligencia Artificial [en línea] Web:
<<http://galahad.plg.inf.uc3m.es/~iat/unidad-didactica.pdf>>
[Última consulta: 24/9/2010]

[2] Concepto de planificación [en línea] Webs:
<<http://www.davidam.com/docu/aplic-ia/planific.html>>
[Última consulta: 24/9/2010]

<<http://www.cs.us.es/cursos/ia1-2006/temas/tema-08.pdf>>
[Última consulta: 24/9/2010]

[3] Lenguaje PDDL [en línea] Web:
<<http://ipc.informatik.uni-freiburg.de/PddlResources?action=AttachFile&do=view&target=mcdermott-et-al-tr-1998.pdf>> [Última consulta: 23/01/2012]

[4] Planificación y lenguajes de especificación [en línea] Web
<<http://users.dsic.upv.es/~osapena/papers/Tesis.pdf>>
[Última consulta: 15/10/2010]

[5] Requisitos PDDL [en línea] Web:
<<http://www.plg.inf.uc3m.es/~pad/transparencias/Representation.pdf>>
[Última consulta: 23/12/2011]

[6] Descripción de PDDL 3.1 [en línea] Web:
<<http://www.plg.inf.uc3m.es/ipc2011-deterministic/resources>>
[Última consulta: 23/12/2011]

[7] Definición ciclo en cascada [en línea] Web:
<<http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node10.html#SECTION00331000000000000000>> [Última consulta: 18/10/2010]

[8] Información sobre Lazarus y Pascal [en línea] Web:
<http://wiki.lazarus.freepascal.org/index.php/Lazarus_Documentation/es>
[Última consulta 30/11/2011]

[9] Tutoriales Canvas [en línea] Web:

<http://www.schoolfreeware.com/Free_Pascal_Tutorials.html>

[Última consulta 11/03/2011]

[10] Dominios de la International Planning Competition 2011 [en línea] Web:

<<http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential>>

[Última consulta 01/12/2011]

[11] Bases cotización [en línea] Webs:

<<http://www.boe.es/boe/dias/2010/01/18/pdfs/BOE-A-2010-729.pdf>>

[Última consulta 15/02/2012]

<[http://www.seg-](http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecoti)

[social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecoti](http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecoti)
[za36537/index.htm](http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecoti)>

[Última consulta 15/02/2012]

<[http://www.seg-](http://www.seg-social.es/prdi00/groups/public/documents/normativa/163375.pdf)

[social.es/prdi00/groups/public/documents/normativa/163375.pdf](http://www.seg-social.es/prdi00/groups/public/documents/normativa/163375.pdf)>

[Última consulta 15/02/2012]

[12] Cálculo de la amortización de los equipos [en línea] Web:

<<http://cooperacionunjfsc.wordpress.com/2010/01/15/calculo-de-los-costes-de-un-proyecto-fuente-curso-de-buenas-practica-de-cooperacion-universidad-y-empresa-upv-ingenio-caeu-oei/>> [Última consulta 15/02/2012]

[13] Herramienta itSIMPLE [en línea] Web:

<<http://code.google.com/p/itsimple/>> [Última consulta 19/10/2010]

[14] Herramienta VLEPPO [en línea] Web:

<http://www.dit.hua.gr/~raniah/vleppo_en.html> [Última consulta 20/10/2010]