



BENEMÉRITA UNIVERSIDAD

AUTÓNOMA DE PUEBLA

Facultad de Ciencias de la Computación

“Planeación nutricional desde tu móvil”

Tesis Profesional que para obtener el título de:

Lic. en Ciencias de la Computación

Presenta:

Erick Madrid Cruz

Asesor:

Dr. Fernando Zacarías Flores

Puebla, Pue.

Primavera 2013

INDICE

INTRODUCCIÓN.....	- 1 -
JUSTIFICACIÓN.....	- 6 -
OBJETIVOS	- 7 -
CAPÍTULO 1: AGENTES INTELIGENTES	- 8 -
1. Agentes	- 8 -
1.1 El concepto de agente	- 9 -
1.2 Propiedades de los agentes	- 11 -
1.3 Taxonomías de agentes	- 12 -
1.3.1 Agentes de interfaz.	- 12 -
1.3.2 Agentes colaborativos.	- 13 -
1.3.3 Agentes móviles.....	- 13 -
1.3.4 Agentes de recuperación de información.	- 14 -
1.4 Arquitecturas para agentes Deliberativos.....	- 16 -
1.5 Fundamentos del modelo BDI	- 18 -
1.5.1 Conceptos y propiedades del modelo BDI	- 19 -
1.5.2 Actitudes mentales en el modelo BDI	- 20 -
1.5.3 Agentes deliberativos del juego	- 20 -
1.6 Ejemplos de modelos de agentes:.....	- 21 -
CAPÍTULO 2: ANSWER SET PROGRAMMING – ASP	- 28 -
2. Introducción	- 28 -
2.1 ASP – Nuestro paradigma lógico.....	- 31 -
2.2 Lógica proposicional	- 31 -
2.3 Tipos de cláusulas.....	- 32 -
2.4 Teoría	- 34 -
2.5 Definiciones básicas	- 36 -
2.6 Ejemplos de cálculo de answer sets	- 38 -
CAPÍTULO 3: SISTEMA PARA MODELADO LÓGICO – DLV	- 46 -
3.1 Lenguaje base de DLV	- 46 -
3.2 Base de datos y Programación lógica.....	- 47 -
3.2.1 Base de datos Extensional	- 47 -
3.2.2 Base de datos Intencional.....	- 48 -
3.3 Reglas y hechos	- 49 -
3.3.1 Ejemplos básicos usando reglas	- 49 -

3.4	Negación y suposición de un mundo completo	- 51 -
3.5	Negación verdadera.....	- 51 -
3.6	Restricciones de Integridad	- 52 -
3.7	Restricciones Débiles	- 53 -
3.8	Aritmética de predicados en DLV	- 54 -
3.9	DLV ^K – front-end de planeación.....	- 56 -
3.9.1	Descripción.....	- 56 -
3.9.2	Ejemplos.....	- 60 -
CAPITULO 4: J2ME Y SERVLETS		- 62 -
4.1	J2ME.....	- 62 -
4.1.1	Componentes de J2ME.....	- 62 -
4.1.2	Máquina virtual	- 63 -
4.1.3	Configuración.....	- 63 -
4.1.5	Arquitectura J2ME.....	- 65 -
4.1.6	MIDlet.....	- 66 -
4.2	GPRS	- 68 -
4.2.1	Conexión a redes	- 70 -
4.3	Servlets o aplicación Web.....	- 76 -
4.2.1	Concepto de Servlet.....	- 76 -
4.2.2	Ventajas de los Servlets	- 77 -
4.3.1	Arquitectura de los Servlets	- 78 -
4.3.2	Ejemplos de aplicación de los servlets	- 81 -
CAPITULO 5: PLANEACIÓN NUTRICIONAL		- 83 -
5	Introducción	- 83 -
5.2	Arquitectura	- 83 -
5.3	Aplicación.....	- 84 -
5.2.1	Ejecutándolo en DLV ^K	- 89 -
CONCLUSIONES Y TRABAJO A FUTURO		- 91 -
BIBLIOGRAFÍA.....		- 92 -

INTRODUCCIÓN

Más del 70% de los internautas buscan con frecuencia información sobre salud en internet, esto es lo que afirman los últimos estudios sobre el comportamiento de los internautas en países latinos (México en particular). Si a esto le sumamos la alta penetración de los teléfonos móviles en México y el crecimiento del mercado de las aplicaciones móviles sobre salud, pronto, todos llevaremos nuestras aplicaciones sobre salud y nutrición en nuestros dispositivos móviles.

Webs, Foros, Blogs, Redes Sociales, etc. son utilizadas a diario por profesionales o pacientes para informarse, intercambiar conocimiento o conocer/mejorar su estado de salud en el caso de padecer alguna enfermedad.

Jano, Medline Plus, Tu otro médico, Diario Médico, Forum Clinic o RedPacientes son solo algunos ejemplos.

Y ¿qué está ocurriendo con la tecnología móvil?

- el **93,3% de los hogares** dispone de al menos un **teléfono móvil activo**;
- la media europea de penetración del smartphone (teléfono inteligente) es de un 47,6%, y según comScore, **el 55,2 % de los europeos apuestan por ellos**;
- **en México**, según datos oficiales mostrados en el periódico MILENIO, **se realizan más de 100 millones de descargas, entre aplicaciones, películas, series de TV, etc., cada día.**

Por tanto, no es difícil de entender que las aplicaciones móviles sobre salud constituyen un nicho de oportunidad con un potencial enorme de usuarios en México, Europa y todo el mundo.

Según el ONTSI (Observatorio Nacional de las Telecomunicaciones y de la Sociedad de la Información), en menos de tres años, se han lanzado más de 6.000 aplicaciones distintas, tan solo para temas relacionados con la salud. Algunas previsiones hablan de un crecimiento de un 800% en el próximo trienio en torno a descargas de “aplicaciones” en los teléfonos móviles de más de 500 millones de personas en todo el mundo.

Hablando de aplicaciones, a principios de verano Sanitas, lanzaba el concurso Ideas Sanitas en el que los internautas podían enviar y votar sus

aplicaciones móviles preferidas sobre diferentes aspectos relacionados con la salud. Una de las temáticas a tratar era la nutrición.



Figura 1. Resultados obtenidos del concurso Ideas Sanitas

Las aplicaciones ganadoras fueron las siguientes:

- **Glutenmed:**

Pensada tanto para el profesional de la salud como para el paciente celíaco y sus cuidadores/familiares, la aplicación permite administrar con seguridad medicamentos a celíacos.

- **Social Diabetes:**

Es una aplicación para la autogestión de la Diabetes Mellitus tipo 1

- **Glucómetro:**

Aplicación Android para controlar la diabetes. Seguimiento del nivel de glucosa con un medidor de glucosa bluetooth o de forma manual.

- **Tu peso ideal:**

Aplicación gratuita disponible en la AppleStore, que estima cuál es tu peso ideal, ofreciendo dos, uno basado en el Índice de Masa Corporal (el que recomiendan los médicos) y otro basado en el gusto popular.

Finalmente, el proyecto de investigación y desarrollo propuesto pretende contribuir al consumo razonado y equilibrado de los alimentos de que disponemos. Esto responde a la necesidad de contar con prácticas nutricionales sanas a la mano de todas las personas, ya que actualmente existen aproximadamente 852 millones de personas de todo el mundo que no pueden obtener suficientes alimentos para llevar una vida sana y productiva.

PLANTEAMIENTO DEL PROBLEMA

Este trabajo de tesis propone el desarrollo de un sistema nutricional para la planeación estratégica de una dieta personalizada desde un dispositivo móvil. El sistema pretende apoyar a los millones de usuarios que día a día se enfrentan al dilema nutricional correcto, es decir, que debemos consumir como parte de nuestra alimentación diaria sin rebasar los límites adecuados para una alimentación balanceada, nutricional y rica tanto en proteínas, vitaminas, minerales, así como en sabor y sencillez. Para el desarrollo de este trabajo, proponemos el uso del paradigma de programación lógico disyuntivo Answer Set Programming a través del lenguaje de acción de planeación declarativa y representación de conocimiento llamado “*K*”. Lenguajes como este nos permiten la formalización de problemas de planeación complejos que involucran no determinismo y conocimiento incompleto de una manera flexible. *K* adopta un punto de vista de programación lógica donde los flujos representan el estado epistémico de un agente que puede ser cierto, falso o indefinido en cada estado. Además, mostraremos como este punto de vista de los estados de conocimiento puede ser provechosamente aplicado a diversos ámbitos de planificación conocidos en la literatura, y particularmente aplicado a dominios novedosos como la planeación estratégica nutricional.

Por otra parte, el trabajo propone hacer un esfuerzo adicional por llevar esta tipo de desarrollos que, ya de por si son complejos en entornos de computadoras de escritorio, a entornos móviles, es decir, poner en las manos de todo el mundo (usuarios de dispositivos móviles e internet) una herramienta que nos permita mejorar de manera sustancial la salud de los mexicanos y de todo el mundo.

Para abordar el tema móvil, proponemos el diseño de una arquitectura que nos permita la integración de las tecnologías en su versión stand-alone (independientes en PC's) y móviles. Para esto, proponemos integrar las tecnologías de protocolo de control de transmisión (TCP – Transmission Control Protocol) y el protocolo de acceso inalámbrico (WAP – Wireless Access Protocol). Buscando una propuesta innovadora que permita contar con una herramienta ubicua, es decir, que permita la integración de las TIC's en el entorno de las personas, de forma que las computadoras involucradas no se perciban como objetos diferenciados, es decir, el usuario debe percibir a nuestra aplicación de la misma manera en que percibe la aplicación para realizar llamadas o envío de mensajes de texto.

Además, la herramienta propuesta debe ofrecer un servicio acertado (always accurate), oportuno (always appropriate), permanente (anytime), en todo lugar (anywhere) y de cualquier forma (anyway – cualquier dispositivo móvil). El protocolo en que basaremos nuestra propuesta es el conocido como protocolo de acceso inalámbrico WAP, del que proponemos hacer uso del servicio general de radio paquetes (GPRS – General Packet Radio Service).

JUSTIFICACIÓN

Uno de los problemas que más aquejan a nuestro país es la obesidad. La obesidad se ha triplicado en las tres últimas décadas en México. Así, en este trabajo de tesis hemos desarrollado un sistema compuesto por una parte llamada Back-end compuesta por un sistema basado en lógica que permita hacer planeación nutricional a la medida del usuario, permitiéndole conocer en todo momento los beneficios de su alimentación. En una segunda componente incorporamos la parte Front-end basada en tecnologías móviles que permiten poner en la mano del usuario la herramienta que le permita planear sus alimentos diarios y cuyo objetivo es el de alcanzar un estatus lo más cercano al peso ideal de este. Así, nuestra propuesta incluye el modelado y desarrollo de una herramienta que permita coadyuvar en la solución del problema de obesidad mundial y particularmente de nuestro país.

OBJETIVOS

Objetivo General

- Demostrar la viabilidad de generar planes nutricionales que incluyan dietas apropiadas para cada individuo a través de dispositivos móviles.

Objetivos Específicos

- Las dietas se planearán en un esquema basado en 3 o 5 comidas al día.
- Las dietas podrán ser modificadas de manera interactiva en tiempo real, para lo que se deberá ejecutar un ajuste en la dieta del día en cuestión, con el objetivo de mantener el consumo de calorías requerido para el día.
- Se podrá consultar la cantidad de nutrientes de los diferentes alimentos basado en las etiquetas alimenticias y/o etiquetas de nutrición
- Hacer una selección cualitativa de los alimentos, es decir, elegir las porciones adecuadas para cada comida con el objetivo de conseguir un aporte nutricional completo al estar representados todos los grupos de alimentos.
- Para que una dieta sea equilibrada cualitativamente, deben formar parte de ella todos los grupos de alimentos: energéticos, plásticos o constructores y reguladores o protectores.

CAPÍTULO 1: AGENTES INTELIGENTES

1. Agentes

Los agentes en la Inteligencia Artificial son sistemas de cómputo que interactúan con un ambiente externo, observando los cambios y el desempeño de las acciones. Así, un agente racional basado en el paradigma lógico Answer Set Programming es aquel que usa la lógica para asimilar las observaciones de su ambiente, para poder razonar acerca de sus metas u objetivos y poder decidir cuál es la acción correcta a realizar. ASP es uno de los paradigmas que más auge ha tenido debido a la proliferación de sistemas basados en este paradigma. ASP proporciona el soporte necesario para modelar las características básicas de los agentes racionales [5].

El paradigma de agentes virtualmente ha invadido cada sub-campo de la Ciencia de la Computación. El concepto de agente recientemente ha incrementado su influencia en la investigación y desarrollo de sistemas computacionales basados en lógica. La programación lógica (LP) permite una clara representación del conocimiento estático. Recientemente también se han obtenido mejoras significativas en la eficiencia de las implementaciones de programación lógica para razonamiento no monótono [4]. Existen algunas implementaciones en DLV y Smodels que nos permiten tener una semántica procedural y declarativa unificada, haciendo de esta forma mas corta la distancia que separa la teoría de la práctica. Además, algunas extensiones a la Programación Lógica en el campo de la no monotonía, como aprendizaje inductivo, revisión de creencias, argumentación, preferencia, etc.

Por otro lado, científicos de la computación deben desarrollar una metodología sistemática para el diseño y construcción de componentes de software basados en agentes racionales; estos deben ser sistemas capaces de basar su comportamiento en el conocimiento de su medio ambiente y de sus propias capacidades, poder intercambiar información y servicios con sistemas similares así como con humanos. Gran cantidad de trabajo teórico en el área de ASP se ha llevado a cabo por connotados investigadores como son: Gelfond, Lifschitz, Kowalski, Pereira, Alferes, Eiter, Zacarías y otros [3] [5] [14].

Nuestro país no puede quedarse al margen de este desarrollo. Por lo que investigadores de nuestro país se han involucrado en este vertiginoso desarrollo dado en relación al paradigma de agentes, lo han hecho tanto en la investigación teórica así como también implementando aplicaciones. Para unificar esfuerzos se han creado, en algunas universidades del mundo grupos de investigación teórica y práctica, tal es el caso de grupos como: Texas Action Group (TAG) en USA, CENTRIA, en Portugal, Knowledge Base Systems (KBS)

en Austria y GMLogyC en México, donde los investigadores han obtenido importantes avances teóricos y prácticos en lógica y agentes. La noción de agente racional ha empezado a proveer una metodología más unificada y coherente para resolver problemas en el campo de la Inteligencia Artificial.

La Inteligencia Artificial ha agregado dinamismo al manejo del conocimiento. En Inteligencia Artificial como en cualquier otra ciencia, la lógica tiene un importante rol. De hecho, toda ciencia implícita o explícitamente hace uso de la lógica y el razonamiento para construir argumentos, contra argumentos y establecer disputas. La Inteligencia Artificial ha ayudado a los investigadores a explorar nuevas cuestiones y métodos, y combinar modalidades de razonamiento dispar dentro de un marco de trabajo uniforme y unificado, así como a tratar con información incompleta, imprecisa, contradictoria y cambiante. Conforme el mundo avanza, deben hacerlo los programas que representan el conocimiento respecto a este. Se destacan dos marcos abstractos para tratar con modificaciones en una base de conocimiento representada por una teoría preposicional, una establecida por Katsuno y Mendelzon en [6]. Este trata con mundos cambiantes, y es conocido como *actualización de teorías*.

Como se describió anteriormente, construir software basado en agentes inteligentes no es un problema sencillo. Este paradigma exige que sistemas basados en agentes presenten algunas (o todas) características que nos permitan considerarlos “inteligentes”. En primera instancia necesitamos definir que entendemos por agentes. Para esto, presentamos algunas posiciones dadas por connotados investigadores del área.

1.1 El concepto de agente

La palabra "Agente" se refiere a todo ente que posee la habilidad, capacidad y autorización para actuar en nombre de otro. A diario, los agentes humanos asisten a las personas en tareas que requieren recursos especializados o conocimiento específico en un dominio. Por ejemplo, una secretaria atiende y resuelve situaciones en nombre de su jefe: administra la agenda, coordina las reuniones, recibe a los visitantes.

Los agentes cumplen con los requerimientos para los cuales fueron entrenados. El usuario "delega" en el agente una o varias tareas que debe llevar a cabo quedando a la espera de los resultados. Dichas tareas, son a menudo fácil de especificar pero - en algunos casos – complejas de realizar. Los investigadores en el campo de los agentes computacionales han dado varias definiciones al término, cada uno desde su óptica particular,

fundamentada básicamente en la línea de investigación en la cual trabajan (Inteligencia Artificial, Ingeniería de Software, Sistemas Autónomos).

El término de Agente racional ha sido acuñado en los últimos años y normalmente se relaciona con sistemas desarrollados en las áreas de: economía, teoría de juegos, teoría de decisión, e inteligencia artificial. En este contexto, un agente racional es aquel software que tiene claro cuáles son sus preferencias, modelos e incertidumbres vía valores esperados y tiene la cualidad de elegir siempre la acción que resulta óptima de entre todas las acciones posibles para sí mismo. Algunas otras áreas que la han acogido dentro de su desarrollo son: Las ciencias cognitivas, la ética, y la filosofía [13].

Existen algunas otras definiciones acerca del concepto de agente que se basan en el contexto en que estos se estudian y desarrollan. Algunas de las definiciones que sobresalen son:

Wooldrige y Jennings (Wooldrige, 1998) definen agente como "un programa autocontenido capaz de controlar su proceso de toma de decisiones y de actuar, basado en la percepción de su ambiente, en persecución de uno o varios objetivos".

Según Nwana (Nwana, 1996) "El término agente se refiere a un componente de software y/o hardware que es capaz de actuar para poder ejecutar tareas en nombre de un usuario".

Pattie Maes (Maes, 1994), del AI Lab del MIT (Massachusetts Institute of Technology) definió a los agentes autónomos como sistemas computacionales que habitan en algún ambiente dinámico y complejo, registrando su estado y actuando autónomamente, llevando a cabo una serie de objetivos o tareas para los cuales fueron diseñados".

Según Frankling y Graesser (Frankling, 1996): "Un agente autónomo es un sistema situado dentro (o es parte) de un medioambiente, que sensa tal medio ambiente y actúa sobre éste, por un tiempo, en persecución de su propia agenda y afectando así lo que sensará en el futuro".

Russell y Norvig (Russell, 1995): "Un agente puede ser visto como algo que percibe su ambiente a través de sensores y actúa contra este ambiente a través de efectores"

Nicholas Negroponte, director del Media Lab del MIT, hizo la observación que el futuro de la computación será de "delegar a" y no de "manipular" computadoras. Dentro de este marco el término agente se redefine para cada aplicación que persiga dicho objetivo.

Resumiendo las diferentes posturas encontradas en la literatura nosotros adoptamos la siguiente definición sobre el concepto de agentes:

Definición de agente. Un agente es una entidad computacional (hw, sw) que percibe y actúa autónomamente en un ambiente abierto y dinámico, aprendiendo y cooperando con otros agentes (el usuario mismo) para brindar un beneficio a su usuario.

1.2 Propiedades de los agentes

En base a todas las definiciones anteriores es posible extraer algunas características que deben tener los agentes: deben ser parte de un ambiente dinámico, deben poder censar su entorno y actuar sobre él, y deben responder según los objetivos para los cuales fueron diseñados. Deben – entonces – poseer una serie de atributos o propiedades que lo definen como agente (Wooldridge, 1996):

- Autonomía
- Aprendizaje
- Colaboración

Enseguida damos las definiciones acerca de cada una de las características que un agente inteligente debe cumplir.

- **Autonomía:** Capacidad de actuar sin la intervención directa de una persona o de otro agente. Un agente debe poder controlar sus propias acciones y estado interno. Una vez que el usuario activa el agente indicando algún objetivo de alto nivel, éste actúa independientemente, seleccionando estrategias y monitoreando el progreso en busca de la meta. Si falla con una estrategia, usará otra, pero sin intervención humana o con la mínima indispensable.

- **Aprendizaje.** Un agente aprende en la medida que su conocimiento es completo y no presenta inconsistencias. Este aprendizaje es de parte del usuario, de su ambiente y de los otros agentes con que convive.

- **Colaboración:** Un agente debe colaborar. Debe tener habilidad para interactuar con otros agentes o incluso con alguna persona (el usuario), para solicitar información o bien para exponer los resultados obtenidos de la ejecución de las tareas agendadas. La naturaleza de la comunicación dependerá del tipo de agente con quien se comunique (humanos o no), en

ambos casos se deberá establecer un protocolo común de intercambio de información entre ambas partes.

1.3 Taxonomías de agentes

Los agentes pueden clasificarse de varias maneras, teniendo en cuenta algunas de las propiedades que poseen o bien haciendo hincapié en alguna en particular. De esta manera puede armarse un árbol taxonómico que abarque todas las combinaciones de propiedades y tareas que se quieran. Nosotros consideramos una clasificación de acuerdo a líneas de investigación y desarrollo de agentes, donde lo que se prioriza es la función u objetivo principal del agente. De acuerdo a lo anterior consideramos la siguiente clasificación:

- Agentes de interface
- Agentes colaborativos
- Agentes móviles
- Agentes de recuperación de información

1.3.1 Agentes de interfaz.

Un agente de interfaz es un software casi-inteligente que asiste a un usuario cuando interactúa con una o más aplicaciones. Son asistentes personales que reducen el trabajo por la sobrecarga de información, como por ejemplo el filtrado de los mensajes de correo electrónico o la recuperación de archivos de Internet [7].

Esta categoría de agentes apoyan y proveen asistencia a su usuario. El agente observa y monitorea las acciones que toma el usuario en la interfase, aprende nuevos atajos, y sugiere mejores formas de hacer las tareas. La idea es que el agente pueda adaptarse a las preferencias y hábitos de sus usuarios.

Enfatizan la autonomía y el aprendizaje para llevar a cabo tareas para sus dueños y trabajan en el mismo ambiente que estos [8].

A su vez, de los agentes de interface pueden encontrarse subdivisiones debido a diferentes tareas para las cuales son construidos. Las más comunes son:

1. **Asistentes:** Trabajan realizando tareas típicas como el manejo de la agenda. Estos agentes ayudan al usuario a planificar las reuniones. Sus acciones incluyen negociar, aceptar o rechazar reuniones.

2. **Filtros:** Su tarea principal es la de analizar información de acuerdo a un conjunto de reglas dadas por el usuario. La aplicación típica es el filtrado de mensajes de correo electrónico.

3. **Guías:** Asisten a los usuarios en el uso de una aplicación. Estos agentes monitorean las acciones de los usuarios e intentan sugerir qué pasos a realizar para alcanzar el objetivo. Algunos ayudan a navegar por Internet.

1.3.2 Agentes colaborativos.

Los agentes colaborativos constituyen un sistema multiagente, es decir existe más de un agente dedicado a satisfacer los requerimientos de sus usuarios. Para ello es necesario contar con esquemas de comunicación entre agentes que permitan la cooperación y el intercambio de conocimiento. Además, deben poseer un alto grado de autonomía para interactuar con los demás agentes.

La motivación detrás de la construcción de agentes colaborativos es que los sistemas construidos con unidades relativamente simples proveen mayor funcionalidad que un ente mayor, pudiendo extender la funcionalidad del sistema más allá de las capacidades de uno de sus miembros.

Estas arquitecturas posibilitan contar con mayor confiabilidad (debido a la redundancia) y mayor velocidad (debido al paralelismo) en el sistema conjunto.

Las áreas de aplicación de este tipo de agentes incluyen:

1. Resolución de problemas demasiado grandes.
2. Interconexión de múltiples sistemas.
3. Manejo de información proveniente de fuentes distribuidas.

1.3.3 Agentes móviles

Los agentes móviles son procesos capaces de “viajar” por una red de computadoras, interactuando con hosts externos, recolectando información en nombre de su dueño y retornando a “casa” luego de completar las tareas establecidas [8].

Los agentes forman un nivel de abstracción más para el usuario, detrás del cual se encuentran soluciones a cuestiones técnicas en algunos casos complicadas. Una de estas cuestiones es la distribución, es decir, como manejar recursos computacionales distribuidos. Con la idea de agentes móviles

los recursos distribuidos no están completamente ocultos al usuario pero tampoco completamente expuestos.

La noción de movilidad viene del objetivo de reducir el tráfico innecesario dentro de una red, con lo que se pueden reducir los costos de comunicación. Además, al aportar una nueva forma de computación distribuida posibilita el mejor aprovechamiento de los recursos de la red y permite que los usuarios tengan acceso a una cantidad mayor de recursos.

Por ejemplo, debido a que las sesiones en busca de un recurso determinado ocasionalmente son largas, la idea de agentes móviles provee una solución. Un usuario delega la tarea de búsqueda de información a un agente, establece una comunicación con la red y “envía” al agente a cumplir con su misión. La próxima vez que el usuario se conecte, el agente “retorna” con los resultados obtenidos.

Para soportar la movilidad, debe existir una infraestructura de transporte que mueva el código del agente de una ubicación a otra. Además, se debe contar con un entorno de ejecución de agentes, donde los agentes “viven”, compuesto por todas las computadoras que los proveen.

Finalmente, para construir sistemas con agentes móviles es necesario resolver algunas cuestiones fundamentales tales como [7]:

1. **Transporte:** ¿Cómo se mueven de lugar en lugar?
2. **Ejecución:** ¿Cómo ejecutar el agente de forma remota?
3. **Autenticación:** ¿Cómo saber si el agente es quien dice ser y a quién representa?
4. **Privacidad:** ¿Cómo asegurar que el agente mantenga resguardado su estado interno?
5. **Seguridad:** ¿Cómo protegerlo de virus?, ¿Cómo prevenir que el agente entre en bucles infinitos o falle?

1.3.4 Agentes de recuperación de información.

El objetivo principal de los agentes dedicados específicamente a la recuperación de información es obtener información por el usuario.

Las tecnologías de la información han expandido los horizontes de los usuarios en cuanto a las formas de generar y acceder a la misma. Pero esta amplia variedad de información distribuida plantea desafíos en cuanto a las formas de manejar su complejidad y heterogeneidad.

Hoy en día, la información se produce en múltiples contextos, se difunde por medios muy variados y se utiliza en todas partes. El rápido crecimiento de la cantidad de documentos en Internet (especialmente en la Internet) presenta la dificultad de poder acceder a la información relevante.

Se produce así el fenómeno conocido como “sobrecarga de información”, entonces se trata de mejorar, pero no eliminar el problema específico de la sobrecarga y administración de la información. Además, existe el problema que debido al gran volumen de información disponible, se mezclan el ruido o desperdicio con la información útil o necesaria, lo que determina la gran dificultad de hallar lo que realmente resulta de interés para las personas.

Las soluciones actuales a este problema se basan en la construcción de motores de búsqueda, con mecanismos de indexación de documentos, combinados con interfaces de consulta apropiadas a esta tarea, o bien, índices manuales multinivel (o directorios), los cuales presentan clasificaciones de los documentos según el criterio de sus autores. Ambas técnicas poseen debilidades visibles. Los primeros, son muy propensos a “recuperar” demasiada cantidad de documentos “no deseables” ya que trabajan mediante técnica de búsqueda de la ocurrencia de los términos buscados en los documentos. Por otra parte, los índices manuales, solamente tienen “parte” de los posibles documentos, debido al alto costo que posee la recuperación, manipulación y clasificación manual de los mismos.

La motivación es poder diseñar una técnica que permita describir los cientos de millones de documentos disponibles de manera precisa, creando un índice de alta calidad, con una forma eficaz y eficiente de acceder a éste (ya sea de manera manual o automática). Una de las soluciones posibles se basa en los agentes de recuperación de información. Estos agentes pueden asistir a un

usuario novato en la formulación de consultas avanzadas, en base a sus necesidades de información. Además, permiten acceder e integrar fuentes heterogéneas y manejar diferentes tipos de formatos de información.

Los agentes de recuperación de información poseen métodos para permitir el rápido acceso y recuperación de información relevante. Tienen la tarea de administrar, manipular y juntar información de fuentes distribuidas. Pueden tener mecanismos de búsqueda y navegación flexibles y algoritmos de clasificación poderosos.

El objetivo es construir agentes capaces de “armar” un diario personalizado, sabiendo dónde buscar, cómo encontrar lo buscado y cómo armarlo luego [8]. Los agentes se presentan como una herramienta muy útil en la tarea de resolver el problema de la sobrecarga de información, debido a que éstos pueden realizar sus tareas mucho más rápido que las personas y, además, se encuentran disponibles las veinticuatro horas.

1.4 Arquitecturas para agentes Deliberativos

En [6] se define a una arquitectura de agente deliberativo, como una que contiene un mundo representado explícitamente y un modelo lógico del mismo, y en la cual las decisiones (por ejemplo acerca de las acciones a realizar) son hechas por medio de un razonamiento lógico (o por lo menos pseudo-lógico), basado en concordancia de patrones y manipulación simbólica.

a) Agentes de planeación:

Desde inicios de los setenta, la comunidad de la Inteligencia Artificial dedicada a la Planeación ha estado fuertemente relacionada con el diseño de agentes. Parece razonable entonces, que muchas de las innovaciones en el diseño de agentes provengan de esa comunidad.

STRIPS [8] La arquitectura de este sistema, contiene una descripción simbólica del mundo y el estado deseado de objetivos, y del conjunto de descripciones de las acciones que caracterizan las pre y post condiciones asociadas con varias acciones. Tomando como base este conocimiento intenta encontrar una secuencia de acciones que lograrán el objetivo, utilizando un análisis simple. El algoritmo de planeación de STRIP es muy sencillo y se probó que es ineficiente en problemas de complejidad moderada.

Se han realizado varios intentos para construir agentes cuyo componente principal sea un planeador. Por ejemplo: el sistema Integrated Planning, Execution and Monitoring (IPEM) que está basado en un planeador sofisticado no lineal

Enseguida, presentamos la arquitectura que más se utiliza en el diseño e implementación de agentes lógicos, los conocidos como agentes BDI de sus siglas del inglés (Belief, Disires and Intention) [17].

b) BDI

BDI significa creencias (Beliefs), deseos (Desires) e intenciones (Intention), que son componentes mentales presentes en muchas arquitecturas de agentes. Las creencias representan el conocimiento del agente, los deseos representan los objetivos y las intenciones otorgan deliberación al agente. La exacta definición de estos términos varía de autor a autor [17].

Este tipo de arquitectura ve al sistema como un agente racional que tiene ciertas actitudes mentales, tales como: creencias, deseos e intenciones, representando respectivamente, los estados de información, motivacional y deliberativos del agente. Estas actitudes mentales determinan el comportamiento del sistema y son críticos para lograr el desempeño adecuado u óptimo cuando la deliberación está sujeta a recursos limitados. [8].

Las creencias de un agente representan el conocimiento del agente. El contenido del conocimiento puede ser cualquiera, por ejemplo conocimiento acerca del ambiente del agente o acerca de su historia. Los deseos son un conjunto de objetivos a largo plazo. Un objetivo es típicamente una descripción de un estado deseado del ambiente. Los deseos proveen al agente de la motivación para actuar. Los objetivos constituyentes de los deseos pueden ser contradictorios, entonces el sistema tiene que poder, de cierta forma, elegir qué objetivo alcanzar primero, es aquí donde aparecen las intenciones. Las intenciones pueden ser consideradas como un conjunto de planes para lograr los objetivos que constituyen los deseos.

Como se apunta en [5], no es necesario, que un sistema especificado en términos de creencias, deseos e intenciones sea diseñado por medio de estructuras de datos correspondientes a cada uno de estos componentes. Sin embargo tal diseño podría simplificar la construcción, mantenimiento y verificación del sistema resultante.

A modo de ejemplo presentamos una arquitectura abstracta con tres estructuras dinámicas representando cada una las creencias, los deseos y las intenciones del agente, junto con una cola de eventos. Se permiten operaciones de actualización y consulta sobre las tres estructuras. Los eventos que el sistema puede reconocer incluyen eventos internos y eventos externos. La siguiente secuencia de acciones se ejecuta en un ciclo infinito. Al inicio de cada ciclo, se lee la cola de eventos y se obtiene una lista de opciones. Seguidamente, se selecciona un subconjunto de opciones a ser adoptadas y se las agrega a la estructura de intenciones. Si existe una intención para ejecutar una acción atómica en ese punto, el agente la ejecuta. Cualquier evento externo que ha ocurrido durante este tiempo se agrega a la cola de eventos. Los eventos internos se agregan apenas ocurren. Posteriormente, el agente modifica las estructuras de intención y deseos de acuerdo al resultado de la ejecución de la intención.

1.5 Fundamentos del modelo BDI

Un agente tiene limitado sus recursos, su capacidad de comprensión y un conocimiento incompleto del entorno en el que vive. Tiene creencias sobre el mundo y deseos que satisfacer, que le conducen a formular intenciones de actuar.

Una intención es un compromiso para realizar planes que, en el momento de formularlo es sólo parcial, ya que depende del estado en el momento de su ejecución.

El agente ejecuta las acciones que intenta realizar sin volver a razonar, hasta que se ve forzado a revisar sus intenciones, dado que se producen cambios en sus creencias o deseos.

Procesos de razonamiento práctico en el modelo BDI.

1. Decidir qué objetivo se quiere lograr (proceso de liberación).
2. Determinar opciones disponibles, según creencias y deseos.
3. Elegir una o varias opciones.

4. Establecer el compromiso de seguir las intenciones adoptadas.

Decidir cómo se logra el objetivo (razonamiento medios-fines): Al comprometerse con una intención se debe:

Seguir una línea de acción razonable para conseguirla, es decir, hacer los que se cree que satisface mejor la intención.

- ⊗ Persistir en la intención (si falla la línea actual, intentar encontrar otra) hasta conseguirla o llegar a creer que no es posible conseguirla.
- ⊗ No perder tiempo considerando opciones incompatibles.

Cada cierto tiempo, reconsiderar las intenciones.

- ⊗ Encontrar el balance entre una actitud reactiva y una proactiva.

1.5.1 Conceptos y propiedades del modelo BDI

Creencias, deseos e intenciones

Actitudes sobre la información (creencia o conocimiento).

Proactitudes (deseos, intención, elección, compromiso, actuación, etc ...)

Propiedades.

- ⊗ Consistencia (entre intenciones, entre intenciones y creencias, etc.)
- ⊗ Persistencia.

Una intención es una elección a la que se compromete.

- ⊗ Por defecto, las creencias persisten.
- ⊗ Buena fe: Solo se compromete uno con lo que se siente capacitado.
- ⊗ Las intenciones son las que guían el razonamiento medios-fines.
- ⊗ Introspección para revisar periódicamente los estados mentales.

Intenciones como núcleo del modelo.

1.5.2 Actitudes mentales en el modelo BDI

Creencias

Componente informativo necesario para determinar el estado más probable del entorno (que quizá no coincida con el estado real). Es una noción poco diferente a la del conocimiento.

Deseos

Representación del estado que motiva al agente y costo/beneficio.

Se distingue de los objetivos del agente en que pueden ser muchos al mismo tiempo y pueden ser mutuamente incompatibles.

Intenciones

Captura el componente deliberativo del agente.

Incluye planes formados por acciones, para conseguir los objetivos.

⊕ **Beliefs:** Conjunto de creencias actuales o información que el agente tiene de su entorno.

⊕ **Bfr:** Función de revisión de creencias, que actualiza las creencias a partir de su percepción del entorno.

⊕ **Options:** Función de generación de opciones a partir de creencias e intenciones.

⊕ **Desires:** Conjunto de opciones actuales o deseos que terminan las líneas de acción.

⊕ **Filter:** Función filtro que representa el proceso de liberación del agente y actualiza sus intenciones según creencias, deseos e intenciones actuales.

⊕ **Intentions:** Conjunto de intenciones actuales, es decir, los asuntos con los que está comprometido.

⊕ **Action:** Función de selección de la acción a realizar.

1.5.3 Agentes deliberativos del juego

En el área de la Inteligencia Artificial distribuida se han desarrollado varias propuestas con distintas arquitecturas que permiten, mediante el uso de agentes, modelar el comportamiento de sistemas complejos. Es esta sección se explica cómo construimos el módulo de razonamiento de un agente racional para juegos. También se muestra la implementación de un conjunto de estrategias para juegos bipersonales de suma cero e información perfecta.

La idea principal en que se basa este desarrollo es que la inteligencia o comportamiento inteligente resulta de la interacción de sus partes más simples, entre ellas y con el medio ambiente. Particularmente, en el presente trabajo de tesis presentamos el diseño e implementación de un juego de dominó que basa sus estrategias de juego en agentes racionales, dos de los cuales se han diseñado para que colaboren entre sí, dado que juegan como pareja. Mientras que los otros dos han sido diseñados como jugadores (agentes) individualistas.

El agente aquí desarrollado basa su arquitectura en la estructura BDI (de sus siglas en inglés – Belief, Desire and Intentions). Las creencias que estos agentes manejan son el conocimiento (creencia) acerca de la mesa del juego, es decir, conocer por un lado, el número de las fichas que se encuentran colocadas en los extremos de la partida, y por otro lado, conocer como han sido las tiradas del resto de los agentes (jugadores). Respecto de los deseos, los agentes tienen como objetivo principal, el de ganar en pareja y como secundario el de ganar de manera individual. Las intenciones le proporcionan a nuestros agentes la habilidad de la deliberación.

1.6 Ejemplos de modelos de agentes:

1.6.1 Websailor

Es un agente del área de recuperación de información, posibilita a sus usuarios atacar el problema conocido como sobrecarga de información y su consecuente baja eficiencia a la hora de realizar consultas por medio de los modelos tradicionales. Websailor (Jaccard, 1999) actúa como intermediario entre el usuario y el espacio World Wide Web. Analiza el comportamiento de los usuarios al navegar, determinando la relevancia de cada una de las páginas accedidas, a los efectos de mantener un perfil. En cada nueva consulta toma ventaja de estos datos acerca de las preferencias del usuario, a los efectos de poder filtrar y reorganizar los links devueltos por buscadores tradicionales. Como interface de consulta genera páginas de respuesta adaptadas a los hábitos del usuario.

Se implementa del lado del usuario en la forma de un applet que un servidor proxy (adaptado especialmente) incluye en cada una de las páginas HTML que los navegadores soliciten. De esta forma se pueden capturar datos tales como: tiempo de lectura, utilización de vínculos, impresión o grabación de las páginas, etc. Este proyecto es parte de una línea de investigación que tiende a detectar

y aprovechar al máximo los hábitos de los usuarios con la finalidad de incorporar este conocimiento a asistentes personales.

1.6.2 Softbot

Softbot (Etzioni, 1994) es un ejemplo clásico de agentes de interface, de la subcategoría de los asistentes personales, desarrollado en la Universidad de Washington. Utiliza un shell UNIX y permite interactuar con diversos servicios de Internet. Sus efectores usan algunos servicios como ftp, telnet, mail y sus sensores otros como netfind, archie, gopher.

Softbot provee una interface integrada para utilizar los recursos de Internet, elige dinámicamente cuáles servicios invocar y en qué secuencia. Además, cambia su comportamiento en base a la información obtenida del medio (por ejemplo si se encuentra deshabilitado un servicio intentará con otros).

Como actúa como un asistente personal inteligente, el softbot permite que el usuario especifique una solicitud de alto nivel. Luego utilizará búsquedas, inferencias y conocimiento para determinar cómo satisfacer el requerimiento. La interface permite completar una especie de formulario que acepta incompletitud de datos (en algunos casos) que luego deberá tratar. Dicha interface se diseñó en base a los siguientes principios:

- **Orientada por objetivos:** La solicitud expresa "qué" desea el usuario. El "cómo" obtenerlo y "cuando" satisfacer el pedido es responsabilidad del softbot.
- **Flexible:** Cuando un objetivo no está completo o correcto, se intenta descifrarlo para luego satisfacerlo.
- **Balanceada:** El softbot balancea el costo de encontrar la información contra el tener que "molestar" al usuario con más preguntas.
- **Integrada:** Provee una interface uniforme a una amplia gama de servicios de Internet.

Ejemplo de utilización: Supóngase la necesidad de enviar un grupo de archivos sobre un tema a una determinada persona de una universidad. Supóngase además que no se sabe la dirección de correo de dicha persona. La interface del softbot permitirá completar los datos que se conocen (Apellido y nombre de la persona, universidad, nombre de los archivos a enviar, ubicación). Ya en acción el softbot puede emplear algún servicio tipo netfind para tratar de localizar a la persona, o bien revisar el site de la institución en busca de la misma. Una vez localizada puede enviarle el grupo de archivos vía mail (si lo

es que obtuvo su dirección) o vía ftp (si lo es que obtuvo es su directorio dentro del site).

1.6.3 Virtual Mattie

Otro ejemplo de asistente personal es Virtual Mattie (VMattie) (Franklin, 1996). Es un agente autónomo e inteligente, que "vive" en un ambiente UNIX. Está programado en Java y Perl y realiza las tareas de un bedel, liberando a la persona y aumentando la eficiencia. Trabaja sin intervención humana y se comunica mediante e-mail, utilizando lenguaje natural. VMattie cada semana envía un e-mail con el anuncio de los seminarios agendados en un departamento académico. Para ello, se comunica con los organizadores de los seminarios, arma los anuncios y los envía a todos los integrantes de una lista de correo que él mantiene. Si bien la tarea no es complicada para el bedel real, requiere un alto nivel de percepción para reconocer e interpretar el contenido de los mensajes de los organizadores y los interesados, los cuales poseen un formato libre. Además, debe manejar fechas y componer distintos tipos de mensajes a distintas personas: anuncios, erratas, recordatorios, confirmaciones. La arquitectura de VMattie consta de tres módulos centrales: Objetivos, Comportamientos y Registros de Atención.

VMattie posee varios objetivos operando en paralelo:

- 1) obtener a tiempo la lista de seminarios en la semana;
- 2) mantener completa la información acerca de cada seminario;
- 3) mantener actualizada la lista de correo;
- 4) confirmar cada mensaje entrante.

Estos objetivos tienen niveles de urgencia variados conforme lleguen mensajes, y se aproxime la fecha de envío del anuncio del seminario.

Los comportamientos son acciones de nivel medio, por ejemplo:

- 1) agregar-dirección-a-lista;
- 2) asociar-organizador-con-seminario;
- 3) componer-recordatorio.

Los registros de atención mantienen información extraída de los mensajes entrantes, por ejemplo:

- 1) nombre-del-organizador;
- 2) dirección-de-correo;
- 3) fecha;
- 4) orador;
- 5) nombre-del-seminario.

Existe un registro especial el cual registra el tipo de mensaje que ingresó. El dominio de los tipos de mensajes en VMattie es limitado. Por ejemplo, solicitudes de baja de una lista de e-mail, mensajes de los organizadores enviando el título de un seminario y/o el nombre del disertante. La información en los registros de atención provee un entorno de activación para los comportamientos, que operan con su contenido.

Un módulo especial (módulo de procesamiento de conocimiento) se encarga de interpretar los mensajes, "entendiendo" varios formatos de algunos vocablos que pueden escribirse completos o abreviados, con mayúsculas o minúsculas. La información almacenada en este módulo se obtuvo del análisis del cuerpo de los mensajes recibidos por el bedel real en dos años. Se identificaron distintos tipos de mensajes, abreviaturas utilizadas, costumbres y tratamientos, títulos, etc.

Luego de interpretar un mensaje entrante, VMattie envía un mensaje de confirmación con el contenido de los que entendió del mensaje, e invita a enviar las correcciones necesarias.

Finalmente, una base de conocimiento contiene las plantillas para el armado de los mensajes a enviar (al igual que con los mensajes entrantes existen unos pocos tipos de mensajes posibles de enviar), como así también la información necesaria para completar dichos mensajes y la lista de las direcciones de correo de los destinatarios. Esta base es actualizada por los comportamientos.

1.6.4 Visitorbot

La tarea de agendar las visitas a un laboratorio suele ser bastante sencilla y rutinaria, pero consume demasiado tiempo. La secuencia normal de actividades que involucra incluye la recepción de mensajes con las solicitudes de visita, la lista de las personas a quienes se desea ver y los horarios preferidos. Luego, es necesario avisar a dichas personas solicitando la respuesta y sus horarios.

Finalmente, se debe armar – según las restricciones impuestas – la visita, agendarla y enviarla a todos los participantes indicando lugar, teléfono y demás datos. Un agente llamado VisitorBot (Kautz, 1994) se especializa en manejar la tarea antes descripta. Inicialmente, el usuario se comunicaba directamente con el agente vía e-mail, lo que permitió observar lo siguiente:

- La comunicación vía e-mail resultaba dificultosa ya que se debían escribir los mensajes de una forma predefinida y el menor error de tipeo ocasionaba el mal funcionamiento, lo que determinó que la siguiente versión utilizaría una interfaz gráfica.
- Era necesario utilizar rutinas redundantes para el manejo de errores, por mensajes incompletos o incorrectos.
- La tarea de crear una buena agenda no era una tarea que requiera planificación avanzada sino que podía traducirse a un problema de programación entera.

Estas observaciones condujeron al desarrollo de "userbots", es decir, agentes (de interface) que se comuniquen con el usuario para permitir especificar los requerimientos y que luego los envíen al Visitor Bot de una forma normalizada. Esto se tradujo en la construcción de un sistema de agentes colaborativos destinados al objetivo final.

La interacción normal entre el usuario y el Visitor Bot se realiza de la siguiente forma. El Visitor Bot envía un mail a cada userbot. Éste determina el tipo de comunicación posible con el usuario (por ejemplo, una ventana si se trata de una interfaz gráfica) y le solicita si quiere agendar la solicitud hecha por un visitante. Si la respuesta es afirmativa, el userbot pasa la información al Visitor Bot, el cual solicita una lista de horarios preferidos por el usuario. Finalmente, el userbot genera un mensaje conteniendo las preferencias del usuario y lo envía de regreso al Visitor Bot.

Este diseño posee algunas ventajas. Primero, el Visitor Bot no necesita saber acerca del tipo de interfaz con la cual comunicarse con el usuario, reduciendo la complejidad de su construcción. Segundo, los usuarios pueden tener acceso a diferentes recursos, por lo cual tendrían distintos userbots.

Finalmente, el agente que realiza una tarea específica solicita qué información se necesita para llevar a cabo la tarea pero no cómo obtenerla, ya que esto queda a elección de userbot en función de sus medios de comunicación (voz, mail, fax) y las preferencias del usuario.

1.6.5 Web Watcher

Para una búsqueda en el espacio Web, los usuarios recurren generalmente a un motor de búsqueda que trabaja con el principio básico de la aparición (ocurrencia) de los términos buscados, para retornar una lista de hipervínculos que permitan acceder a la información. No siempre un usuario conoce con exactitud los términos que lo llevarán a lo buscado y en función de esto – luego de algunas búsquedas iniciales – va refinando los criterios para tratar de alcanzar su objetivo. Con la idea de mejorar las búsquedas existe WebWatcher (Joachims, 1995), un agente de interface (de la subcategoría de guías) que acompaña al usuario en su navegación por la World Wide Web, e interactivamente le va sugiriendo dónde continuar. Además, el usuario puede comunicarse con el sistema para retroalimentarlo. WebWatcher funciona como un aprendiz ya que observa y aprende de las acciones del usuario. Con el tiempo, registra partes de la WWW por donde pasó y los tipos de temas en los cuales usuarios anteriores tuvieron interés.

A partir de la página principal de la School of Computer Science, Carnegie Mellon University se puede solicitar un "tour" acompañado por WebWatcher. En primer lugar se debe ingresar una frase que corresponda al tema de interés, luego WebWatcher retorna a la página inicial y desde allí comienza el "tour". El sistema funciona como un servidor proxy, el cual entrega las páginas solicitadas por el usuario con algunos "agregados" y "modificaciones". En la parte superior de la página es agregado un conjunto de comandos para comunicarse con el agente. Cada hipervínculo es reemplazado por la URL del servidor de WebWatcher y finalmente, si alguno de los vínculos es altamente recomendado lo remarca como una sugerencia para el usuario.

Los consejos de WebWatcher se basan en conocimiento obtenido en tours anteriores. Cada vez que el usuario selecciona un vínculo (recomendado o no), WebWatcher lo acompaña a la siguiente página, registrando la selección como un ejemplo de entrenamiento para mejorar las recomendaciones futuras. Al finalizar el "tour" el usuario tiene la posibilidad de indicar si alcanzó el objetivo o no, brindando una última forma de retroalimentación para el agente. Para sugerir un vínculo apropiado dado el interés del usuario el sistema requiere del conocimiento de la siguiente función:

Calidad-del-Vínculo : Página x Interés x Vínculo ® [0, 1]

La cual expresa la probabilidad de que un usuario seleccione un Vínculo dada la Página actual y el Interés. Para aplicar dicha función se utilizan tres aproximaciones para aprender de la experiencia:

- Aprendizaje de tours anteriores;
- Aprendizaje de la estructura del hipertexto;
- Métodos que combinan las dos estrategias anteriores.

CAPÍTULO 2: ANSWER SET PROGRAMMING – ASP

2. Introducción

El pensamiento matemático ha sido desde siempre uno de los paradigmas más sorprendentes de la humanidad. Los pueblos árabes por ejemplo, han ejercido siempre una fuerte fascinación tanto por sus costumbres como por algunas de las áreas científicas predominantes: La matemática, la astronomía, la física y la medicina. También los griegos fueron maestros en el pensamiento, principalmente en la filosofía. Además, es sobresaliente como esos pueblos siempre procuraron desarrollar sus culturas y civilizaciones a través de monumentos que no solo muestran la inquietud artística sino que también, muestran su precisión matemática muchas veces, vacilante e indecisa [13].

Cuantas veces en la vida, se nos presentan problemas que parecen insolubles, como los que en su aspecto matemático nos ofrece el Hombre que Calculaba [13]. En una mesa de un restaurant se hallan tres viajeros. Uno de ellos parecía muy joven; era alto, delgado, de ojos claros y ostentaba un bellissimo turbante amarillo como yema de huevo, con una barra blanca donde lanzaba destellos una esmeralda de rara belleza; los otros dos eran bajos, de anchas espaldas y tenían la piel oscura, como los beduinos de África.

Se diferenciaban de los demás tanto por su aspecto como por sus vestidos. Estaban absortos en una discusión que a juzgar por los ademanes era enconada como ocurre cuando la solución al problema es difícil de hallar. El jeque dirigiéndose a los tres musulmanes, dijo:

- ¡ Aquí tenemos al eximio Calculador !

Luego señalando a éstos añadió:

- ¡ Aquí están mis tres amigos ! Son criadores de carneros y vienen de Damasco. Se les plantea ahora uno de los más curiosos problemas que haya visto en mi vida. Es el siguiente:

Como pago de un pequeño lote de carneros recibieron aquí en Bagdad, una partida de vino excelente, envasado en 21 vasijas iguales, de las cuales se hallan:

7 llenas

7 mediadas

7 vacías

Quieren ahora repartirse estas 21 vasijas de modo que cada una de ellos reciba el mismo número de vasijas y la misma cantidad de vino.

Repartir las vasijas es fácil. Cada uno se quedará con siete. La dificultad está, según entiendo, en repartir el vino sin abrir las vasijas; es decir, dejándolas exactamente como están. ¿Será posible, ¡oh Calculador!, hallar una solución satisfactoria a este problema?

Beremiz, después de meditar en silencio durante dos o tres minutos, respondió:

-El reparto de las 21 vasijas podrá hacerse, ¡oh jeque! sin grandes cálculos. Voy a indicarle la solución que me parece más sencilla. Al primer socio le corresponderán:

3 vasijas llenas;

1 mediada

3 vacías.

Recibirá así un total de 7 vasijas.

Al segundo socio le corresponderán:

2 vasijas llenas;

3 mediadas;

2 vacías.

Recibirá así también siete vasijas.

La parte que corresponderá al tercero será igual a la del segundo, esto es:

2 vasijas llenas;

3 mediadas;

2 vacías.

Según la división que acabo de indicar cada socio recibirá 7 vasijas e igual cantidad de vino. En efecto: Llamemos 2 –dos- a la porción de vino de una vasija llena, y 1 a la porción de vino de la vasija mediada.

El primer socio recibirá, de acuerdo con la división:

$$2 + 2 + 2 + 1$$

y esa suma es igual a siete unidades de vino.

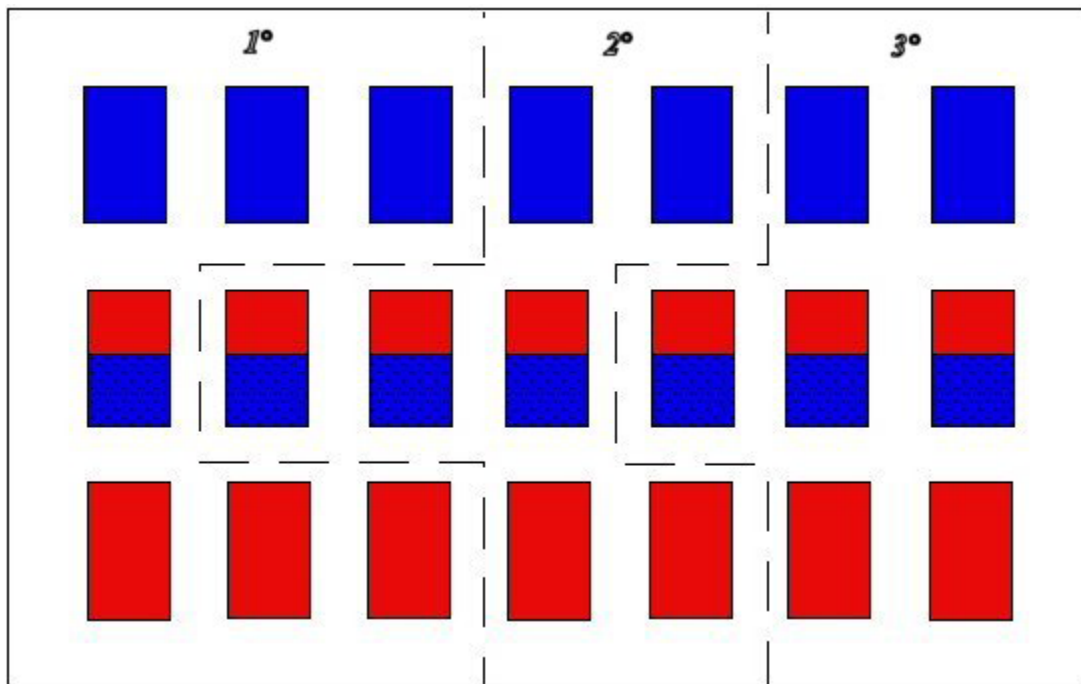
Cada uno de los otros dos socios recibirá:

$$2 + 2 + 1 + 1 + 1$$

y esa suma es también igual a 7 unidades de vino.

Esto viene a probar que la división que he sugerido es cierta y justa. El problema, que en apariencia es complicado, no ofrece la mayor dificultad en cuanto a su resolución numérica.

La solución presentada por Beremiz fue recibida con mucho agrado, no solo por el jeque, sino también por sus amigos damacenos.



Exposición gráfica de la resolución del Problema de las Veintiuna Vasijas. La primera hilera está constituida por las siete vasijas llenas, la segunda por las siete vasijas medianas y la tercera por las siete vasijas vacías. La partición propuesta deberá efectuarse siguiendo las líneas punteadas.

Como este problema, existen una gran variedad de problemas que se presentan de manera cotidiana en nuestras vidas, y sin embargo, como hemos visto con el ejemplo anterior, la solución de éstos, sólo depende de una correcta modelación que nos permita hallar la solución correcta.

Enseguida, presentamos uno de los paradigmas lógicos más prolíferos de los últimos años que han permitido modelar y resolver muchos de los problemas más difíciles encontrados en la literatura.

2.1 ASP – Nuestro paradigma lógico

Answer set programming (ASP) es el paradigma lógico que en los últimos años ha tenido un gran desarrollo [9, 10, 11, 12]. Además, se han obtenido resultados relevantes que establecen relaciones directas entre ASP y la lógica intuicionista y otras lógicas. Esto ha dado a este paradigma un gran impulso a su uso entre los miembros de la comunidad científica. También, han surgido diversas aplicaciones en el campo de la Inteligencia Artificial. En este capítulo damos un panorama general sobre ASP con la finalidad de establecer el marco teórico empleado en nuestro trabajo.

Consideramos que una teoría está compuesta por tres elementos fundamentalmente que son: Primero, el conjunto de axiomas lógicos (correspondientes a los axiomas que conforman la lógica que elijamos). Segundo, El conjunto de axiomas propios (en nuestro contexto lo conforman el conjunto de cláusulas comprendidas en un programa lógico). Tercero, Las reglas de inferencia empleadas para su aplicación en la obtención de nuevo conocimiento (en nuestro caso la regla de modus ponens). Más adelante definimos de manera formal estos tres ingredientes que son la parte fundamental del conocimiento básico necesario para entender este trabajo. Enseguida, damos algunos conceptos y definiciones que serán utilizados a lo largo de este trabajo.

2.2 Lógica proposicional

El lenguaje de la lógica proposicional tiene un alfabeto que consiste de:

- ⊗ Símbolos Proposicionales $p_0, p_1, \dots$
- ⊗ Conectivos $\wedge, \vee, \rightarrow, \perp$
- ⊗ Símbolos auxiliares “(”, “)”, “,”, “.”

donde $\wedge$, $\vee$, $\rightarrow$, son conectivos binarios y $\perp$ es un conectivo sin argumentos. También, el conectivo $\perp$ denota falso. Los símbolos proposicionales son también llamados átomos o proposiciones atómicas.

Una fórmula T (también llamada Top) es introducida como una abreviacion de $\perp \leftarrow \perp$. Representaremos la negación clásica con $\neg$ (también conocida como “not”). Consideramos que un programa es un conjunto de cláusulas tomando en consideración la siguiente convención:

$A \leftarrow B$ es otra forma de escribir $B \rightarrow A$.

Denotamos una cláusula de la siguiente manera:

$$H \leftarrow B$$

Refiriéndose a H como la cabeza (Head) y a B como el cuerpo (Body).

2.3 Tipos de cláusulas

Enseguida definimos los diferentes tipos de cláusulas que se puede encontrar en la literatura [11].

⊕ Si $H = A$ y $B = \bigwedge_{i=1}^m B_i$, B_i letra proposicional entonces $H \leftarrow B$ se le denomina cláusula positiva.

⊕ Si $H = \perp$ y $B = \bigwedge_{i=1}^m B_i$, B_i letra proposicional entonces $H \leftarrow B$ se le denomina cláusula Horn.

⊛ Si $H \leftarrow B$ donde $H = A$, $B = \bigwedge_{i=1}^m B_i, \bigwedge_{j=m+1}^n \neg B_j$ es decir, $A \leftarrow B_1, B_2, \dots, B_m, \neg B_{m+1}, \dots, \neg B_{m+n}$ le denominamos cláusula normal.

⊛ Si tenemos $H \leftarrow B$ donde $H = \bigvee_{k=1}^m A_k$, $B = \bigwedge_{j=m+1}^n B_j, \bigwedge_{l=n+1}^l \neg B_l$ es decir, $A_1, \dots, A_k \leftarrow B_1, B_2, \dots, B_m, \neg B_{m+1}, \dots, \neg B_j$ le llamamos cláusula disyuntiva.

⊛ Si tenemos $H \leftarrow B$ donde $H = \bigvee_{k=1}^m A_k, \bigvee_{i=m+1}^n A_i$, $B = \bigwedge_{j=n+1}^t B_j, \bigwedge_{l=t+1}^s \neg B_j$, i. e. $A_1 \vee \dots \vee A_m \vee \neg A_{m+1} \vee \dots \vee \neg A_n \leftarrow B_1, B_2, \dots, B_m, \neg B_{m+1}, \dots, \neg B_j$ le denominamos cláusula libre.

⊛ Si $H = B$, H y B son anidaciones de $\vee$, $\wedge$, y $\neg$, es una cláusula aumentada.

Cada uno de estos tipos de cláusulas definen una clase de programa [11], por ejemplo:

$a \vee b \leftarrow e \wedge d \wedge \neg c$ Disyuntiva, Libre

$\perp \leftarrow p \vee q$. Restricción

Existen algunos tipos de cláusulas especiales tales como las siguientes [11]:

Si $B_1, \dots, B_m, \neg B_{m+1}, \dots, \neg B_{m+n}$ es T entonces identificamos la regla con A y le llamamos hecho.

Si A es $\perp$ entonces decimos que la regla es una restricción.

Una literal L es un átomo A o un átomo fuertemente negado $\sim A$.

Así, un programa P es un conjunto finito de reglas normales, donde A y cada B_i son literales.

A es llamada la cabeza de un regla r , denotada por $H(r)$, y a $B_1, \dots, B_m, \neg B_{m+1}, \dots, \neg B_{m+n}$ le llamamos el cuerpo de la regla r y lo denotamos por $B(r)$.

2.4 Teoría

Dentro del paradigma ASP consideramos que una teoría se compone de tres elementos fundamentalmente [11] como lo son:

1. Conjunto de axiomas lógicos, que normalmente son definidos de manera concreta para cada sistema axiomático existente.
2. El conjunto de axiomas propios, que en nuestro caso lo constituye el programa lógico representado en DLV¹.
3. EL tercer y último componente lo constituye la regla de inferencia llamada “modus ponens”.

Axiomas lógicos.- Existen diferentes sistemas axiomáticos.

Axiomas propios.- Pueden ser las cláusulas de un programa.

Regla de inferencia.- Modus Ponens.

1) $A \rightarrow B$

2) A

3) B

Esta regla significa que si tenemos como hipótesis A y $A \rightarrow B$ entonces podemos concluir B .

¹ <http://www.dbai.tuwien.ac.at/proj/dlv>

Veamos un ejemplo de lo que sería una teoría en nuestro marco de trabajo.

Ejemplo 2.1 Supongamos que P es un programa que solo contiene átomos positivos en el cuerpo, A y B letras proposicionales, entonces:

$P = \{ A1 \leftarrow B11, \dots, B1n.$

...

...

...

$Ak \leftarrow Bk1, \dots, Bkn. \}$

Consideremos a la lógica clásica como nuestro contexto teórico, por tanto nuestros axiomas lógicos son:

Axioma 1) $A \rightarrow (B \rightarrow A)$

Axioma 2) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

Axioma 3) $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$

Y como axiomas propios las cláusulas de nuestro programa P :

$A1 \leftarrow B11, \dots, B1n.$

•

•

•

$Ak \leftarrow Bk1, \dots, Bkn.$

Como única regla de inferencia el Modus Ponens.

Así, tendríamos nuestra teoría.

Enseguida, definimos algunos de los conceptos teóricos empleados en el desarrollo de este trabajo.

2.5 Definiciones básicas

Definición 2.1 FORM (conjunto de formulas bien formadas) [11]:

A es una fórmula bien formada:

Si A y B son fórmulas bien formadas, así también lo son:

$A \vee B,$ $A \wedge B,$ $A \leftarrow B,$ $A \leftrightarrow B,$ $A \rightarrow \perp$

Definición 2.2 Dado σ un conjunto de cláusulas decimos que existe una deducción de A supuesto σ , notación $\sigma \vdash A$ si existe $B_1, B_2, \dots, B_n$ fórmulas bien formadas tales que para cada $i \in \{1, \dots, n\}$, B_i es un axioma (A_1, A_2, A_3) o $B_i \in \sigma$ o existe B_k [11]

$B_k := B_j \rightarrow B_i$

B_j

entonces B_i .

donde $k, j < i$.

Definición 2.3 Toda letra proposicional, $\perp$ y T , ($T := \perp \rightarrow \perp$) son denominadas fórmulas elementales. Cualquier fórmula construida con $\wedge$ y $\vee$ sobre fórmulas elementales es denominada fórmula básica [11].

Definición 2.4 Un conjunto de letras proposicionales X satisface una fórmula básica F ($X \models F$) si:

Para F fórmula elemental, $X \models F$ si $F \in X$ o $F = T$ [11].

$X \models F \vee G$ si $X \models F$ y $X \models G$

$X \models F \wedge G$ si $X \models F$ o $X \models G$

Definición 2.5 Dado P un programa básico constituido por fórmulas básicas. Un conjunto de letras proposicionales X se dice cerrado bajo P si para toda cláusula $H \leftarrow B \in P$ tenemos que $X \models H$ siempre y cuando $X \models B$ esto es, si $X \models B$ entonces $X \models H$ [11].

Definición 2.6 Sea X un conjunto de letras proposicionales y P un programa constituido por fórmulas básicas, X es denominado un Answer Set de P si el conjunto X es mínimo entre los conjuntos de átomos cerrados bajo P [11].

Definición 2.7 (Answer Set) La reducción de un programa normal (disyuntivo, libre, aumentado) relativo al conjunto de letras proposicionales X , es definido de la siguiente manera:

Para una fórmula elemental F [11]:

$$F^X = F$$

$$F \wedge G)^X = F^X \wedge G^X$$

$$(F \vee G)^X = F^X \vee G^X$$

$$(\neg F)^X = \perp \text{ si } X \models F; T \text{ en otro caso.}$$

$$P^X = (H \leftarrow B)^X \mid H \leftarrow B \in P$$

Definición 2.8 Sea P un programa y X un conjunto de átomos. X es un answer set de P si este

es un answer set de la reducción P^X [11].

Sea P un programa, $M \subseteq L_P$, M answer set de P y L_P un conjunto de átomos:

$M \cup \neg\{L_P - M\}$ es el modelo de P [11].

Definición 2.9 Dado un conjunto de literales A y P un programa. Denotamos $\neg A = \{\neg a \mid a \in A\}$. También definimos $\bar{A} = A \cup \neg\{Lit_P - M\}$ [11].

Definición 2.10 Decimos que dos reglas r_1 y r_2 están en conflicto (denotado por $r_1 \nabla \sim r_2$) sí y sólo si ambas tienen la misma cabeza pero una es la contrapuesta de la otra, es decir, si $H(r_1) = \sim H(r_2)$ [11].

Definición 2.11 (Principio de rechazo causal). Sean P_1 y P_2 programas y sea S answer set de la actualización de P_1 y P_2 . Definimos y denotamos al conjunto de reglas a rechazar como:

$$Rej(S, (P_1, P_2)) = \{r \in P_1 \mid \exists r' \in P_2, \text{ tal que } r \nabla \sim r' \text{ y } S \models B(r) \cup B(r')\} \text{ [11]}$$

2.6 Ejemplos de cálculo de answer sets

Enseguida, damos algunos ejemplos que muestran la forma en que se aplica la reducción anterior a las reglas de un programa para calcular sus answer sets.

Ejemplo 2.2 Consideremos el siguiente ejemplo.

Sea P :

$$a \leftarrow \neg \neg a.$$

$$\neg b \leftarrow c \vee b.$$

Propongamos que nuestro answer set sea $X = \emptyset$ entonces, aplicando la reducción a nuestro programa P para verificar si lo satisface.

P^X :

$$a^X \leftarrow \neg \neg a^X$$

$$\neg b^x \leftarrow (c \vee b)^x$$

P^x :

$$a \leftarrow T$$

$$T \leftarrow c \vee b$$

Así, $X = \emptyset$ no satisface a P^x por lo tanto X no es un answer set de P

Si $X = \{a\}$

P^x :

$$a^x \leftarrow \neg \neg a^x$$

$$\neg b^x \leftarrow (c \vee b)^*$$

P^x :

$$a^x \leftarrow T$$

$$T \leftarrow c \vee b$$

por lo tanto X es un answer set de P .

Ejemplo 2.3 Ahora, consideremos el siguiente ejemplo.

$$P: a \leftarrow \neg a$$

Sea $X = \emptyset$

$$P^x: a^x \leftarrow (\neg a)^x$$

$$P^x: a \leftarrow T$$

Por lo tanto X no es un answer set de P .

Ejemplo 2.4 Consideremos el siguiente ejemplo.

Sea P :

$$a \leftarrow \neg b$$

$$b \leftarrow \neg a$$

Sea $X = \emptyset$

P^X :

$$a \leftarrow T$$

$$b \leftarrow T$$

por lo tanto $X = \emptyset$ no es un answer set de P

Sea $X = \{a\}$

P^X :

$$a \leftarrow T$$

$$b \leftarrow \perp$$

por lo tanto $X = \{a\}$ es un answer set de P

Sea $X = \{b\}$

P^X :

$$a \leftarrow \perp$$

$$b \leftarrow T$$

por lo tanto $X = \{b\}$ también es un answer set de P

Ejemplo 2.5 Consideremos el siguiente ejemplo.

Sea P :

$$a \leftarrow \neg b$$

$$b \leftarrow \neg a$$

$$p \leftarrow \neg p$$

$$p \leftarrow \neg a$$

Sea $X = \emptyset$

P^X :

$$a \leftarrow T$$

$$b \leftarrow T$$

$$b \leftarrow T$$

$$b \leftarrow a$$

por lo tanto $X = \emptyset$ no es un answer set de P .

Sea $X = \{a\}$

P^X :

$$a \leftarrow T$$

$$b \leftarrow \perp$$

$b \leftarrow T$

$b \leftarrow a$

por lo tanto $X = \{a\}$ no es un answer set de P .

Sea $X = \{b\}$

P^X :

$a \leftarrow \perp$

$b \leftarrow T$

$b \leftarrow T$

$b \leftarrow a$

por lo tanto $X = \{b\}$ no es un answer set de P

Sea $X = \{p\}$

P^X :

$a \leftarrow T$

$b \leftarrow T$

$b \leftarrow \perp$

$b \leftarrow a$

por lo tanto $X = \{p\}$ no es un answer set de P

Sea $X = \{p, a\}$

P^X :

$a \leftarrow T$

$b \leftarrow \perp$

$b \leftarrow \perp$

$b \leftarrow a$

por lo tanto $X = \{p, a\}$ si es un answer set de P

Sea $X = \{p, b\}$

P^X :

$a \leftarrow \perp$

$b \leftarrow T$

$b \leftarrow \perp$

$b \leftarrow a$

por lo tanto $X = \{p, b\}$ no es un answer set de P

Ejemplo 2.6 Consideremos el siguiente ejemplo.

Sea P :

$\neg b \leftarrow \neg a$

Sea $X = \emptyset$

P^X :

$T \leftarrow T$

por lo tanto $X = \emptyset$ es un modelo de P .

2.7 Intuicionismo

La lógica intuicionista tiene características excelentes para el desarrollo de la teoría de answer set programming. De la misma forma es la base para desarrollar teorías que son usadas en la caracterización de programas lógicos y transformaciones de estas [11].

Intuicionismo esta basado en en concepto de prueba, más que en verdad como en el caso de lógica clásica para explicar el significado y uso de conectivos lógicos. Enseguida damos la definición formal de Intuicionismo [15].

Definición 2.12 La teoría formal del cálculo proposicional intuicionista es: Símbolos lógicos $\leftarrow \vee, \wedge$ y la negación clásica “not” (denotada también como $\neg$). Modus ponens como la única regla de inferencia y el siguiente esquema de axiomas [15]:

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $A \wedge B \rightarrow A$
4. $A \wedge B \rightarrow B$
5. $A \rightarrow (B \rightarrow (A \wedge B))$
6. $A \rightarrow (A \vee B)$
7. $B \rightarrow (A \vee B)$
8. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$
9. $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$
10. $\neg A \rightarrow (A \rightarrow B)$

David Pearce [] demostró que $\mathbf{M}$ un conjunto de átomos, $\mathbf{M}$ es un answer set de $\mathbf{P}$ si y solo si $\mathbf{P} \cup \neg \mathbf{M} \setminus \mathbf{M}$.

Nosotros hemos seleccionado answer set programming en esta tesis debido a que esta es la realización de mucho trabajo teórico sobre razonamiento no monótono y aplicaciones de inteligencia artificial de programación lógica. Además, existen contribuciones recientes y relevantes que establecen relaciones estrechas entre answer set programming y lógica intuicionista. Esto nos permite que podamos utilizar toda la maquinaria existente de intuicionismo en ASP.

CAPÍTULO 3: SISTEMA PARA MODELADO LÓGICO – DLV

Como hemos visto en el capítulo 2, answer set programming es un nuevo desarrollo que ha mostrado tener un impacto significativo en la práctica de la programación lógica y de la representación de conocimiento. Esto obedece primordialmente a que sus fundamentos teóricos son bien entendidos, incluyendo complejidad y poder expresivo. Esto le permite ser aplicada a un amplio rango de problemas de búsqueda y decisión. Además, está estrechamente relacionado a la programación con restricciones. Finalmente, pero no por eso menos importante, actualmente contamos con implementaciones prometedoras que han mostrado en la práctica ser muy eficientes, obteniendo resultados con tiempos iguales o incluso mejor que implementaciones ad hoc desarrolladas en lenguajes como C++.

DLV es una de las implementaciones más prolíferas que cuenta con implementaciones de algoritmos para calcular modelos estables (answer sets) de programas lógicos disyuntivos. DLV es un sistema de bases de datos deductivas basado en programación lógica y que ofrece varios front-ends para varios formalismos avanzados de representación de conocimiento. Formalmente, DLV es datalog disyuntivo extendido con restricciones, negación fuerte y consultas (queries).

3.1 Lenguaje base de DLV

Los elementos básicos de DLV son las constantes, que refieren entidades u objetos que son almacenados en bases de datos relacionales.

Definición 3.1 Una constante debe iniciar con una letra minúscula y le pueden seguir letras, underscore y dígitos. Además, los números son por definición constantes.

not es una palabra reservada y no es una constante.

Definición 3.2 Una variable inicia con una letra mayúscula y le pueden seguir letras, underscore y dígitos. Una variable especial es la llamada variable anónima denotada por "_" (underscore) . Cada ocurrencia de "_" representa una nueva y única variable.

Definición 3.3 Un término es una constante o una variable.

Los siguientes son ejemplos de elementos básicos en DLV:

constantes: a1, 1, 9862, aBc1, c

variables: A, V2f, Vi X3

literales: a, not $\sim$ b(8,K), not weight(X,1,kg)

3.2 Base de datos y Programación lógica

DLV combina dos paradigmas poderosos ampliamente probados en aplicaciones reales. Por tal razón, DLV puede ser visto como un sistema de programación lógica o como un sistema de base de datos deductivo. Visto de esta manera, DLV consta de dos partes fundamentalmente: La bien conocida "Base de datos Extensional" y la "Base de datos Intensional".

3.2.1 Base de datos Extensional

Respecto a la base de datos extensional (BDE), DLV permite exportar e importar bases de datos relacionales a través del controlador ODBC. La BDE puede contener solo hechos tal como se muestra con el siguiente ejemplo.

Ejemplo 3.1. Supongamos que deseamos representar un grafo dirigido como el siguiente:

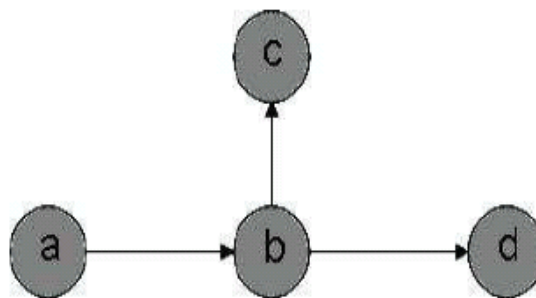


Figura 3.1 Grafo dirigido

Como podemos observar (figura 3.1), lo que deseamos representar es cada uno de los arcos dirigidos de que consta el grafo. Por tanto, la representación sería:

arco(a,b).

arco(b,c).

arco(b,d).

Aquí, arco es llamado un símbolo predicado o símbolo de relación, las partes entre paréntesis son constantes. Estas constantes, como se mencionó previamente, deben iniciar con una letra minúscula.

El anterior símbolo predicado constituye la BDE que puede ser guardado en el archivo "grafo.bde" que podría ser interpretado por DLV de la manera siguiente:

```
> DLV -silent grafo.bde
```

Obteniendo el siguiente answer set: arc(a,b), arc(b,c), arc(b,d)

O bien de la manera siguiente, donde le indicamos a DLV que no nos interesa conocer los símbolos predicados contenidos en la BDE.

```
> DLV -silent -nofacts grafo.bde
```

Obteniendo el answer set: fg, que significa que la respuesta es vacía, es decir, DLV no despliega los hechos definidos en la BDE. Por tanto, no hay información adicional que pueda deducir DLV.

3.2.2 Base de datos Intencional

En esta sección, mostraremos como definir conocimiento en que puede hacer uso de la BDE. A este conocimiento, se le conoce generalmente como el programa, o Base de Datos Intencional - BDI. Así, las reglas denotan relaciones entre literales. La BDE corresponde a los datos de entrada para la BDI o programa. Las reglas tienen la forma siguiente:

$$h_1 \vee \dots \vee h_n :- b_1, \dots, b_m.$$

donde, h_1 a h_n representa explícitamente literales, donde $n > 0$ (es decir, debe haber al menos uno de éstos). Los símbolos " $:-$ " denotan la implicación y $b_1, \dots, b_m$ representan literales en general, donde $m \geq 0$ debe suceder (es decir, el cuerpo puede ser omitido completamente). La parte antes de " $:-$ " es referida como cabeza, la parte después de " $:-$ " como cuerpo. Note que la negación como falla puede ocurrir sólo en el cuerpo. Además, las reglas deben ser seguras.

De manera informal, si el cuerpo evalúa verdadero, la cabeza debe evaluar verdadero también. Los símbolos de negación como falla pueden leerse como "No hay evidencia de que xxx evalúe a verdadero", como en la lógica de default. Así mismo, la cabeza representa una disyunción y el cuerpo una conjunción.

Normalmente, una BDI depende de los datos de entrada definidos en la BDE. Enseguida, comenzaremos presentando programas simples a través de ejemplos.

3.3 Reglas y hechos

Supongamos que deseamos modelar que en todo momento alguien nos dice un chiste, reímos. Además, alguien ahora nos dice un chiste. Esto se puede hacer de la forma siguiente:

broma.

rie :- broma.

La primer línea es un hecho y expresa que una broma es verdadera (una palabra o hecho que tiene el valor de verdad verdadero es llamada una proposición). La segunda línea es llamada una regla. Esta se lee como sigue: Si la broma es cierta, rie debe ser cierto. El signo “:-” representa una flecha hacia la izquierda, en la versión de programación lógica es una implicación.

3.3.1 Ejemplos básicos usando reglas

Enseguida mostramos algunos ejemplos sencillos escritos en DLV. El objetivo de estos ejemplos es introducir a un usuario nuevo en programación lógica desde las reglas más simples, de tal manera que pueda asimilar la idea general de este estilo de programación.

Sean dos funciones $f, g : \mathbb{N} \rightarrow \mathbb{N}$, definidas de la siguiente manera:

$$\begin{aligned} \bullet \quad & f(n) = f(n-1) + g(n) && \text{si } n > 1 \\ & f(1) = 1 \end{aligned}$$

$$\begin{aligned} \bullet \quad & g(n) = g(n-1) + 2 && \text{si } n > 1 \\ & g(1) = 1 \end{aligned}$$

1a) Diseñe dos funciones recursivas para evaluar f y g para cualquier entero dado $n > 0$.

1b) Diseñe también una función recursiva que evalúe la siguiente función:

$$Z(n) = f(1) + f(2) + f(3) + \dots + f(n) \quad \text{para un entero dado } n > 0$$

Solución a)

$f(1, 1)$.

% Condición de paro

```
f(N, M) :- g(N, W), +(N1, 1, N), f(N1, M1), +(M1, W, M). % Def. Recursiva
```

```
g(1, 1). % Condición de paro
g(N, M) :- +(N1, 1, N), g(N1, M1), +(M1, 2, M). % Def. Recursiva
```

Enseguida se mostrará como determinar si un número es par o impar, partiendo del hecho de que el 0 es considerado par y el 1 impar.

```
par(0). % El menor par en los naturales
impar(1). % El menor impar en los naturales
```

```
par(X):- +(F1,1,X),impar(F1). % Def, de número par
impar(Y):-+(F2,1,Y),par(F2). % Def, de número impar
```

El siguiente ejemplo es una de las funciones típicas a definir en cualquier lenguaje lógico.

La función de Ackermann es una función recursiva que toma dos números naturales como argumentos y devuelve un único número natural. Como norma general se define como sigue:

$$A(m, n) = \begin{cases} n + 1, & \text{si } m = 0; \\ A(m - 1, 1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m - 1, A(m, n - 1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

La función se define en DLV como sigue:

```
ackerman(M, N, Z) :- M==0, +(N, 1, Z).
ackerman(M, N, Z) :- M>0, N==0, +(M1, 1, M), ackerman(M1, 1, Z).
ackerman(M, N, Z) :- M>0, N>0, +(M1, 1, M), +(N1, 1, N),
ackerman(M, N1, W), ackerman(M1, W, Z).
```

Finalmente, mostramos como definir una regla que liste todos los pares de enteros positivos que son la suma de un número dado.

p.e. 7 = 6 + 1, 5 + 2, 4 + 3

Nota: No se pueden repetir las parejas, por ejemplo, 6+1 y 1+6.

La regla que lo resuelve en DLV se define como sigue
#maxint = 7.

```
pares(X,Y) :- +(X,Y,#maxint), X>0 , Y >0 , X<=Y .
```

Esta regla mostrara las parejas que suman 7 y además valida que no se repitan tal como se pidió en el ejercicio.

La solución da como answer set:

{pares(1,6), pares(2,5), pares(3,4)}

3.4 Negación y suposición de un mundo completo

Este tipo de negación es conocida como negación como falla y es empleada cuando no estamos conscientes sobre si algo sucede o no. Esta es una de las negaciones ampliamente aceptada como una propuesta razonable: La semántica de modelos estables (answer set programming – ASP).

La negación es tratada como “negación como falla”, en otras palabras, Si un átomo no es cierto en algún modelo, entonces su negación deberá ser considerada como cierta en ese modelo. La idea de cómo aplica este tipo de negación lo veremos con el siguiente ejemplo:

nodo(X) :- arco(X,_).

nodo(Y) :- arco(_,Y).

complemento_arcos(X,Y) :- nodo(X), nodo(Y), not arco(X,Y).

Aquí, la regla complemento_arcos describe el conjunto de arcos en el grafo complementario. Tal que un arco debe ir desde un nodo a otro (posiblemente el mismo), y este arco no debe estar contenido en el arco original.

Note que nodo(X) y nodo(Y) necesitan estar incluidos en el cuerpo en el orden de satisfacer los siguientes requerimientos de seguridad para las reglas. Variables, las cuales ocurren en una literal negada, deben también ocurrir en una literal positiva en el cuerpo.

Ejecutando la regla anterior con el nodo de la figura 3.1 la ejecución será como la siguiente:

```
$ DLV -silent -nofacts simple_graph compl_graph
```

Dando como respuestas:

```
{node(1), node(2), node(3), node(4), comparc(1,1), comparc(1,3), comparc(1,4),  
comparc(2,1), comparc(2,2), comparc(3,1), comparc(3,2), comparc(3,3), comparc(3,4),  
comparc(4,1), comparc(4,2), comparc(4,3), comparc(4,4)}
```

3.5 Negación verdadera

DLV implementa otra noción de negación: Negación verdadera.

La negación como falla, la cual fue introducida antes, no soporta aseveración explícita. Más bien, Sí no hay evidencia de que un átomo es verdadero, este es considerado como falso.

Sin embargo, hay varias situaciones en las cuales la negación como falla no es apropiada porque es necesario que en algunas ocasiones sepamos que algo es explícitamente conocido que es falso. Por esta razón, la negación verdadera es algunas veces referida como *negación explícita*.

Esta negación es denotada precediendo un átomo con “~” o “-”.

Ejemplo 3.2

Imagine una simple situación, en la cual un agente tiene que cruzar una vía de ferrocarril. El agente deberá cruzar está si ningún tren se aproxima. Con esta descripción, uno puede especificarlo a través del siguiente programa:

```
cruza_vía :- not tren_aproxima.
```

Dado que no se tiene otra información, tren_aproxima no sucederá, y cruza_vía será derivado, es decir, si ejecutamos ese programa tendremos:

```
$ DLV -silent rail_naf
{cross_railroad}
```

Pero esto es un problema, ya que la regla indica que el agente cruce, lo que puede causar que el agente sea arrollado por el tren si este se aproxima dado que el agente cruzará (según indica el programa).

La forma de resolver este tipo de situaciones es a través de la negación verdadera, la cual nos permite modelar este mismo problema de la manera siguiente:

```
cruza_vía :- -tren_aproxima.
```

En este caso, -tren_aproxima puede ser considerado como un átomo separado, y dado que no hay información de que esto suceda, cruza_vía no puede ser derivado, es decir, si ejecutamos el programa este dará:

```
$ DLV -silent rail_naf
{ }
```

3.6 Restricciones de Integridad

Las restricciones en nuestro marco de trabajo especifican que no puede ser verdadero en un modelo. En otras palabras, las restricciones son formulaciones de posibles inconsistencias. Este mecanismo es muy útil en conjunción con reglas disyuntivas. Las reglas disyuntivas sirven como generadores para diferentes modelos y las restricciones son usadas para seleccionar solo lo deseado.

La sintaxis de las restricciones es similar a la de las reglas, solo que no tienen cabeza. Como con las reglas, las restricciones deben reunir requerimientos de seguridad. Veamos el siguiente ejemplo:

Ejemplo 3.3 El problema consiste en colorear un mapa que es representado por un grafo. La restricción es, que dos nodos conectados por un arco no pueden tener el mismo color.

Para formular este problema con reglas DLV, reconsideraremos algunas de las reglas previas.

```
nodo(X) :- arco(X,_).  
nodo(Y) :- arco(_,Y).  
color(X,rojo) v color(X,verde) v color(X,azul) :- nodo(X).
```

Los modelos (respuestas) de este programa corresponden a todas las posibles combinaciones de coloreado. Entonces, nosotros solo necesitamos agregar una restricción la cual descartará las combinaciones donde dos nodos adyacentes tengan el mismo color. Esto corresponde a la siguiente restricción.

```
:- arco(X,Y), color(X,C), color(Y,C).
```

Esta restricción nos brindará el resultado deseado, ya que eliminará las combinaciones que no deseamos (nodos adyacentes no pueden ser del mismo color).

3.7 Restricciones Débiles

Una de las características sobresalientes de DLV son las restricciones llamadas “débiles”. Estas nos permiten formular muchos problemas de optimización de una manera natural y sencilla. Mientras las restricciones estándar siempre tienen que ser satisfechas, las restricciones débiles expresan el objetivo, es decir, estas deben ser satisfechas sólo si es posible, pero su violación no elimina modelos (respuestas).

Los answer sets (respuestas) de un programa P con un conjunto W de restricciones débiles son aquellos answer sets de P los cuales minimizan el número de restricciones violadas. Estas son llamadas los mejores modelos de (P,W). Nótese que un programa P puede tener varios mejores modelos (violando el mismo número de restricciones débiles).

Sintácticamente, las restricciones débiles son especificadas como sigue:

$:\sim \text{Conj. [Weight:Level]}$

Donde Conj es una conjunción de literales, y Weight y Level son enteros positivos.

Los pesos y niveles de prioridad se permiten que sean variables, previendo que esas variables también aparezcan en una literal positiva en Conj. El usuario puede omitir el peso o la prioridad o ambos, pero todas las restricciones débiles deben tener la misma forma sintáctica (es decir, el usuario es libre de especificar solo pesos o solo prioridades, o ambas, pero todas las restricciones del programa deben tener la misma forma sintáctica).

Ejemplo 3.4 Considere el siguiente programa contenido en el archivo example1.

```
a v b.  
c :- b.  
:\sim a.  
:\sim b.  
:\sim c.
```

Dado que los pesos y niveles de prioridad son omitidos, sus valores son asignados a 1 por default. Si ejecutamos este programa en DLV, obtenemos la siguiente salida.

```
$ DLV -silent example1  
  
Best Model: {a}  
Cost ([Weight:Level]): <[1:1]>
```

Note que los answer sets de $\{a \vee b, c :- b.\}$ son $\{a\}$ y $\{b, c\}$. La presencia de restricciones débiles descarta $\{b, c\}$ porque este viola dos restricciones (mientras que $\{a\}$ viola solo una restricción).

3.8 Aritmética de predicados en DLV

El razonamiento y cómputo sobre un conjunto finito de rangos enteros es posible con los predicados de DLV.

#int, #succ, +, *.

Estos operadores aritméticos son solo posibles si la aritmética entera es turnada a activa dando un número límite N sobre la línea de comando. DLV solo trabaja con números enteros, es decir, mayores o iguales que cero y menores o iguales que la N definida en la línea de comando.

Ejemplo 3.5 Definamos el predicado de Fibonacci.

```
fibonacci(N,F) :-    #succ(N2,N1),
                    #succ(N1,N),
                    fibonacci(N1,F1),
                    fibonacci(N2,F2),
                    +(F1,F2,F).
```

Como podemos observar, la manera en que se formula esta definición es literalmente escrita de manera directa sobre DLV.

Ejemplo 3.6 La división y módulo no han sido implementados aún. Sin embargo, estos pueden ser fácilmente simulados usando las primitivas existentes como sigue:

```
div(X,Y,Z) :- XminusDelta = Y*Z,
              X = XminusDelta + Delta,
              Delta < Y.
```

```
mod(X,Y,Z) :- div(X,Y,XdivY),
              XminusZ = XdivY * Y,
              X = XminusZ + Z.
```

Note que, al contrario de la construcción de funciones, la adición de estas reglas predicado div y mod en el modelo (esto es, si #maxint es 100, entonces cada answer set del programa contendrá 100*100 hechos para ambos div y mod).

Existe otro conjunto de características igual de importantes que las tratadas en este capítulo, sin embargo, invitamos al lector a referirse a los documentos originales que pueden ser vistos en la siguiente dirección electrónica con acceso libre [18].

Como hemos podido observar, DLV puede ser usado como una implementación para resolver problemas de representación declarativa. En este trabajo de tesis hemos empleado DLV como un lenguaje para la representación de conocimiento relacionado al juego del dominó. Particularmente, hemos representado el conocimiento de un agente cuyo objetivo es el de buscar la mejor tirada en el juego bipersonal llamado dominó. Este tipo de problemas son muy comunes en diversas áreas del conocimiento, tal como se verá en el siguiente capítulo.

3.9 DLV^K – front-end de planeación

El lenguaje K se usa para la planeación bajo conocimiento incompleto. Se le nombro así para destacar que describe las transiciones entre estados de conocimiento (knowledge), más que entre los estados del mundo. A saber, el lenguaje C y muchos otros se basan en extensiones de lógicas clásicas y describen las transiciones entre estados posibles del mundo. Aquí, un estado del mundo se caracteriza por los valores de verdad de una serie de fluentes, es decir, predicados que describen las propiedades relevantes del dominio del discurso, en el que cada fluente es necesariamente verdadero o falso. Una acción es aplicable sólo si una condición previa es verdadera en el estado actual, y la ejecución de esta acción cambia el estado actual mediante la modificación de los valores de verdad de algunos fluentes.

Sin embargo, los agentes de planificación generalmente no tienen una visión completa del mundo. Incluso si su conocimiento es incompleto, es decir, un número de fluentes es desconocido, se deben tomar decisiones, ejecutar acciones, y razonar sobre la base de su información (incompleta) que está al alcance. Por ejemplo, imagine un robot en frente de una puerta. Se desconoce si la puerta está abierta, el robot debe decidir empujar hacia atrás. Alternativamente, se puede decidir poner un sensor que proporcione el estado de la puerta con el fin de obtener una información completa. Sin embargo, esto requiere que una acción de detección apropiada esté disponible y, muy importante, de hecho ejecutable (es decir, el sensor no se rompa). Así, incluso en la presencia de sensores, algunos fluentes pueden permanecer desconocidos y dejar a un agente en un estado de información incompleta.

El lenguaje K adopta una visión trivalente de fluentes en los que sus valores pueden ser verdaderos, falsos, o desconocidos. El lenguaje es muy flexible, y es capaz de modelar las transiciones entre los estados del mundo (es decir, los estados de conocimiento completo) y de razonar sobre ellos como un caso particular, como veremos. Se permite el uso explícito de la negación predeterminada, explotando la potencia de los answer sets para tratar con un conocimiento incompleto. Sin embargo, a diferencia de los idiomas de acción que permitan estados incompletos como AK [67], K no adopta una visión de razonar sobre los posibles casos para determinar las transiciones de estado del conocimiento. DLVK proporciona un potente sistema de planificación declarativa, que está listo para usar para los experimentos[19].

3.9.1 Descripción

Las principales características del lenguaje K son:

Background Knowledge

El dominio de la planificación tiene un background que está representado por un programa lógico, que se requiere para admitir un answer set único, que es polinomialmente computable. Las reglas y los hechos de este programa definen los predicados que no están sujetos a cambios, es decir, representan el conocimiento estático. Un ejemplo en blocksworld es `block(B)`, que define la propiedad (inalterable) de que B es un bloque.

Declaraciones

Los rangos de los argumentos de los fluentes y las acciones se escriben, al afirmar que ciertos predicados deben contenerlos. Por ejemplo, especifica los tipos de los argumentos de la acción `move`.

`move(B, L)` requires `block(B)`, `location(L)`.

Los literales después de la palabra clave `requires` (el `block(B)` y `location(L)`) deben ser literales positivos del conocimiento de background estático mencionado anteriormente.

Reglas de Causalidad

La construcción principal de K son las reglas de causalidad. Estas son sintácticamente similares a las reglas del lenguaje C [28, 43, 45] y tienen la forma:

`caused f if B after A.`

Intuitivamente, la regla dice "Si B es cierto en el estado actual y se sabe que A es cierto en el estado anterior, entonces F se sabe que es cierto en el estado actual". Tanto la parte del `if` como la parte del `after` pueden estar vacías (lo que significa que es verdad).

La negación

La negación por defecto (débil) "not" se puede utilizar en las reglas de la parte posterior del `if` y el `after`. Permite el modelado natural de propiedades inerciales, propiedades predeterminadas y se ocupan del conocimiento incompleto en general, lo cual es similar a la programación lógica con la semántica de answer set. Por otra parte, la negación fuerte (escrito en programas como `"-"`) es soportada también. Con el fin de apoyar la representación problema práctico, K proporciona varias macros que son "ejecutados" a través de la negación débil, como, por ejemplo,

`inertial on(X, Y).`

que informalmente afirma que $on(X, Y)$ se concluye mantener en el estado actual, si $on(X, Y)$ se mantuvo en el estado anterior y $-on(X, Y)$ no se especificó o,

default $-on(X, Y)$.

que establece que $-on(X, Y)$ se mantiene a menos que $on(X, Y)$ haya sido implicado por alguna regla de causalidad.

Ejecutabilidad de las acciones

A fin de ser elegibles para la ejecución, cualquier acción debe satisfacer algunas precondiciones en un determinado estado del conocimiento, que puede ser definido con el uso de declaraciones de ejecutabilidad. Por ejemplo,

executable $move(X, Y)$ **if not** $occupied(X)$, **not** $occupied(Y)$, $X \neq Y$.

dice que el bloque X puede ser movido en la ubicación Y si ambos X e Y no son conocidos como ocupados y estos son diferentes. En este ejemplo, se persigue una estrategia de modelado valiente: Se asume que es suficiente con no saber que un bloque está ocupado para ser capaz de moverlo o mover algo encima de él.

Aquí, $X \neq Y$ (desigualdad) significa $\text{not}(X = Y)$, donde "=" (igualdad) es un predicado incorporado que está tácitamente presente en el conocimiento de fondo.

En general, las múltiples declaraciones de ejecutabilidad sobre la misma acción están permitidas. Si el cuerpo de la regla está vacío, significa que la acción siempre califica para la ejecución, a condición de que las restricciones de tipo en X y Y son respetadas. Por otra parte, la ejecución de una acción A bajo la condición B también puede ser bloqueada, con la declaración

nonexecutable A **if** B.

En caso de conflictos, **nonexecutable** A sobrescribe **executable** A.

Restricciones de integridad

En general, una regla de causalidad expresa una restricción de estado que debe cumplirse en todos estados. Es muy común que las restricciones de integridad del estado para los estados (posiblemente refiriéndose a los respectivos estados precedentes), i.e. conjunciones de literales, es decir, que estos no pueden ser satisfechos simultáneamente. Para facilitar la representación adecuada de las restricciones de integridad, K proporciona una declaración

forbidden B after A

como un acceso directo para `caused false if B after A`. Intuitivamente, esto descarta cualquier estado en el que (se sabe) B es verdadera, si (se sabe) A es cierto en el estado anterior.

Restricciones del estado inicial

K permite declarar reglas de causalidad con una parte `after` vacía que debe aplicarse únicamente al estado inicial. Tales reglas, que representan limitaciones en el estado inicial, deben estar precedidas por la palabra clave `"initially"`. Por ejemplo,

`Initially: forbidden block(B) not supported (B).`

requiere que el fluyente `supported` sea cierto para cada bloque en el estado inicial; la restricción es irrelevante para todos los estados subsiguientes. Las restricciones en el estado inicial resulta rentable porque reduce el cálculo computacional: si se nos garantiza que las acciones preservan alguna propiedad p, entonces es suficiente con comprobar la validez de p sólo en el estado inicial para asegurar que se mantiene en todos los estados.

Ejecución en paralelo de Acciones

De forma predeterminada, la ejecución simultánea de las acciones está permitida en K. Esto puede ser prohibido por las reglas adecuadas, sin embargo, para mayor comodidad del usuario, una declaración

`noConcurrency.`

se proporciona como un acceso directo que obliga a la ejecución de a lo más una acción a la vez.

Manipulación de conocimiento completo e incompleto

K también permite representar transiciones entre los estados posibles del mundo (que pueden ser vistos como estados de conocimiento completo). Primero de todo, es fácil reforzar que el conocimiento de algún fluyente f es completo, utilizando una regla

`forbidden not f not -f.`

Por otra parte, podemos "totalizar" el conocimiento de un fluyente mediante la declaración

`total f.`

que significa que, a menos que un valor de verdad para F puede ser derivado, los casos en los que f resp. -f es verdad serán considerados. En otras palabras, cada estado será "totalizado" agregando f o -f, si ninguno de ellos es verdad.

Metas y Planes

Una meta es un conjunto de literales, el plan es una secuencia de (en general, un conjunto de) las acciones cuya ejecución va desde un estado inicial a un estado donde todos los literales de la meta son verdaderos. En K, el objetivo es seguido por un signo de interrogación y por el número de pasos permitidos en un plan. Por ejemplo,

$\text{on}(c\ b) \quad \text{on}(b\ a) \ ? \ (3)$

solicita un plan de longitud 3 para el objetivo de la figura 1.

Con esto concluye la exposición del lenguaje de planificación K. Destacamos en este punto que el sistema de planificación DLV_k contiene el comando `securePlan`.

por el cual podemos pedir al sistema que calcule sólo planes seguros (a menudo llamados planes conformistas o planes a prueba de fallos en la literatura [30, 66]). Informalmente, un plan es seguro, si es aplicable a partir de cualquier estado legal inicial, es decir, todas sus acciones son ejecutables, independientemente de cómo evolucionan los estados y siempre hacen cumplir el objetivo.

3.9.2 Ejemplos

El problema de disparar de Yale

Un ejemplo que se trata con conocimiento incompleto es una variante del problema de disparar famoso de Yale (ver [33]). Suponemos aquí que el agente tiene una pistola y no sabe si se carga inicialmente. Esto puede ser modelado como sigue:

<i>fluents :</i>	<i>alive. loaded.</i>
<i>actions :</i>	<i>load. shoot.</i>
<i>always :</i>	<i>executable shoot if loaded.</i> <i>executable load if not loaded.</i> <i>caused - alive after shoot.</i> <i>caused - loaded after shoot.</i> <i>caused loaded after load.</i>
<i>initially :</i>	<i>total loaded.</i> <i>alive.</i>
<i>goal :</i>	<i>-alive ? (1)</i>

La declaracion de total conduce a dos posibles estados iniciales legales: S1: {loaded, alive} y S2={-loaded, alive}. Con S1 shoot es ejecutable, mientras que no es con s2. Al ejecutarse shoot se establece el objetivo, por lo que el problema de la planificación tiene un plan optimista
({shoot})

que no es seguro debido a s2.

CAPITULO 4: J2ME Y SERVLETS

4.1 J2ME

Sólo de forma muy simplista puede considerarse que J2ME y J2EE son versiones reducida y ampliada respectivamente, de J2SE: en realidad, cada una de las ediciones enfoca ámbitos de aplicación radicalmente distintos e independientes entre sí. Después de todos, las necesidades computacionales y APIs de programación requeridas por, por ejemplo, un juego gráfico ejecutándose sobre una PDA no tienen mucho que ver con los de un servidor distribuido de aplicaciones basado en EJB.

En lo que sigue presentamos con más detalle los componentes que integran la plataforma J2ME, la taxonomía propuesta por Sun de entornos de ejecución basados en las capacidades gráficas y computacionales de los dispositivos y la funcionalidad para la que están diseñados, la integración en J2ME de otras tecnologías Java relacionadas y la relevancia de la iniciativa J2ME para el desarrollo de aplicaciones y servicios embebidos en dispositivos móviles 2,5 y 3G [20].

4.1.1 Componentes de J2ME

Al contrario que en otras tecnologías orientadas a PCs y ordenadores convencionales, en J2ME el espectro de dispositivos considerados varía enormemente en cuanto a capacidad computacional, memoria y capacidades gráficas. Ante la imposibilidad de establecer una arquitectura común que se adecue a esta variedad de entornos hardware, J2ME define una serie de componentes (building blocks) a partir de los cuales se construye una implantación concreta para un dispositivo determinado. Estos componentes se agrupan en los siguientes tipos [22, 23]:

- Máquina virtual
- Configuración
- Perfil
- Paquetes opcionales

Un entorno de ejecución determinado de J2ME se compone entonces de una selección de máquina virtual, configuración y perfil, y posiblemente otros

paquetes opcionales. En la figura 2.6 se muestran las relaciones entre los distintos componentes de un entorno de ejecución de J2ME.

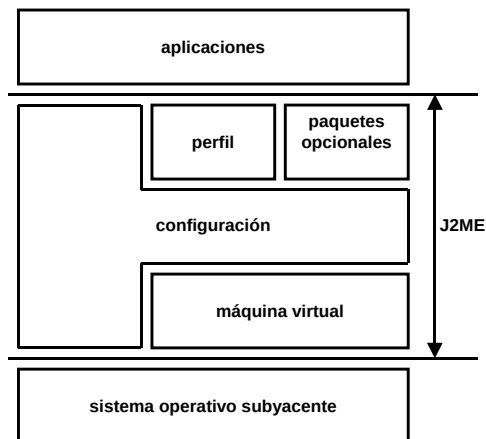


Figura 2.6 Relaciones entre componentes

4.1.2 Máquina virtual

Una máquina virtual de Java (JVM, *Java Virtual Machine*) es un programa encargado de traducir código intermedio (*bytecode*) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para el lenguaje Java[22]. De esta forma, la JVM proporciona al programa Java independencia de la plataforma con respecto al hardware y al sistema operativo subyacente.

Las implementaciones tradicionales de JVMs son en general muy pesadas en cuanto a memoria ocupada (*footprint*) y requerimientos computacionales. J2ME define varias JVMs de referencia adecuadas al ámbito de los dispositivos electrónicos, que en algunos casos se apartan de la especificación estándar relajando algunas exigencias con el fin de obtener implementaciones más ligeras.

4.1.3 Configuración

Una configuración es un conjunto de APIs básicas de Java que definen un entorno generalizado de ejecución [22]. Una configuración no está orientada a ningún campo específico de aplicación, sino que más bien trata de agrupar los dispositivos en cuanto a sus capacidades computacionales y entorno genérico de operación (por ejemplo, si tienen conectividad o no). En particular, las

librerías de interfaz gráfica nunca pertenecen a la definición de una configuración J2ME.

La configuración define la plataforma mínima necesaria para un grupo de dispositivos que tienen similar memoria y capacidades de procesamiento. Se compone de una máquina virtual, unas características del lenguaje Java y un conjunto mínimo de clases que soporta ese grupo de dispositivos.

La configuración tomada para la realización de nuestro proyecto es:

- Configuración del dispositivo: CLDC-1.0

4.1.4 Perfil

Un perfil es un conjunto de APIs orientadas a un ámbito de aplicación determinado, para una configuración J2ME dada. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proveen (por ejemplo, teléfonos móviles, electrodomésticos) y el tipo de aplicaciones que típicamente correrán sobre ellos. La librería de interfaz gráfica es generalmente una componente muy importante de la definición de un perfil [22]. Las interfaces gráficas difieren mucho entre distintos perfiles (displays textuales en teléfonos móviles, interfaces basadas en pen para PDAs, etc.), y en algunos perfiles ni siquiera existen (sistemas embebidos).

Un perfil siempre se define a partir de una configuración dada. Así, cabe pensar en un perfil como un conjunto de APIs que dotan a una configuración J2ME de funcionalidad específica para una familia de dispositivos concreta y para un tipo de aplicaciones determinado.

Con objeto de ofrecer un completo entorno de ejecución específico para cada categoría de dispositivo, las configuraciones se deben combinar con un conjunto de APIs de alto nivel (perfiles), que definen el modelo de ciclo de vida de la aplicación, el interfaz de usuario y el acceso a las propiedades específicas del dispositivo. Algunos de los perfiles existentes son (varios todavía en la etapa de definición):

- **Mobile Information Device Profile (MIDP).** Diseñado para teléfonos móviles y PDAs, incluye la interfaz de usuario, conectividad de red, almacenamiento local de datos y gestión de aplicaciones. Combinado con CLDC, MIDP aporta un entorno de ejecución para Java que minimiza los consumos de memoria y de procesador.
- **Foundation Profile (FP).** Este perfil es el de nivel más bajo asociado a CDC, ya que los perfiles CDC pueden ser vistos como capas que se añaden a para proveer funcionalidades a los diferentes dispositivos. FP suministra una implementación de CDC con capacidades de acceso a red que se utiliza para aplicaciones embebidas en alto grado y sin interfaz de usuario.

- **Personal Profile (PP)** es un perfil diseñado expresamente para dispositivos con un interfaz gráfico completo y con posibilidad de ejecución de applets. Incluye además las bibliotecas del Abstract Windows Toolkit (AWT). Añade, por tanto, un interfaz de usuario básico a FP.
- **Personal Basis Profile (PBP)** es un subconjunto de PP que suministra un entorno para dispositivos que se puedan conectar a una red y que dispongan de un interfaz un poco más desarrollado que aquellos que posean los dispositivos donde PP va dirigido.
- **PDA Profile (PDAP)** es similar al MIDP pero diseñado para PDAs que tengan mejores pantallas y más memoria de los teléfonos móviles.
- **Game Profile (GP)** ofrece la plataforma para escribir juegos en dispositivos CDC.

Actualmente el perfil más utilizado es MIDP, que será el utilizado para el desarrollo de este proyecto, y en concreto usaremos el Perfil del Dispositivo MIDP-2.0.

Las aplicaciones desarrolladas para un determinado perfil serán portables a cualquier dispositivo que soporte ese perfil. Cabe destacar también que un mismo dispositivo puede soportar varios perfiles y que sobre una configuración también pueden residir diversos perfiles.

4.1.5 Arquitectura J2ME

Como se ha explicado anteriormente, el programa de estandarización ha definido diversos entornos de ejecución de J2ME identificados por una selección particular de máquina virtual, configuración y perfil J2ME. Es de suponer que esta arquitectura evolucionará en dos sentidos: refinamiento de los entornos de ejecución definidos (nuevas versiones de los componentes) y definición de nuevos entornos de ejecución.

En el siguiente gráfico podemos ver un esquema modular con las diferentes configuraciones, perfiles y máquinas virtuales:

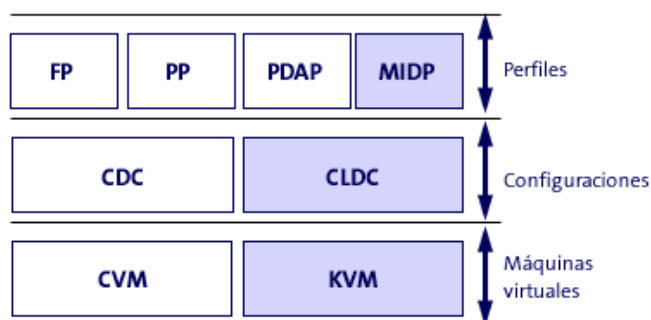


Figura 2.7 Arquitectura J2ME

En la figura 2.7 se muestra el estado actual de la arquitectura J2ME (figura 2.8 ejemplos de Sun). Solamente se han incluido en el diagrama los componentes de J2ME que están estandarizados o muy próximos a ser estandarizados y que tienen especial relevancia para el mercado de los dispositivos para comunicaciones móviles.

J2ME presenta dos configuraciones: CLDC y CDC. La primera se dedica a dispositivos con estrictas limitaciones de memoria, capacidad de cálculo, consumo y conectividad de red. Por otro lado, CDC se encarga de dispositivos con más potencia. Parte de CLDC es un subconjunto de CDC, por lo que la portabilidad de aplicaciones se puede conseguir cuando nos movemos de un entorno más restringido a otro más rico. De la misma manera, y siguiendo en el hilo de la portabilidad, una aplicación en J2ME podrá ejecutarse en J2SE normalmente, salvo que se utilicen las bibliotecas específicas de J2ME.

4.1.6 MIDlet

A las aplicaciones MIDP se denominan "MIDlet", el cual es un programa en lenguaje de programación Java para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java MicroEdition (Java ME). Generalmente son juegos y aplicaciones que corren en un teléfono móvil y está desarrollada bajo la especificación MIDP (perfil para información de dispositivo móvil) [26].

El ciclo de desarrollo de un MIDlet es el siguiente:

- Editar
- Compilar
- Pre-verificar MIDlet
- Ejecución en el emulador
- Ejecución en el dispositivo

Una vez realizados los MIDlets tienen que ser distribuidos en dos archivos especiales. Son los archivos JAR (Java Archive) y los archivos JAD (Java Application Descriptor). Un archivo JAR es un archivo comprimido (en formato ZIP) que contiene las clases (.class) que ha generado la compilación de nuestro programa. Además puede contener los recursos necesarios para el MIDlet como sonidos, gráficos, etc... Finalmente, contiene un archivo con extensión .mf., es lo que se llama un archivo de manifiesto. Este archivo contiene información sobre las clases contenidas en el archivo JAR.

El segundo archivo necesario para la distribución de MIDlets son los archivos JAD. El archivo JAD contiene información necesaria para la instalación de los MIDlets contenidos en el archivo JAR. Un archivo puede contener más de un MIDlet. Cuando ocurre esto, hablamos de un MIDlet suite (véase figura 2.8). Podemos editar los parámetros contenidos en el archivo JAD mediante el botón Settings de KToolBar. Aquí podemos editar información del MIDlet como el nombre, la versión o el autor del MIDlet (o de los MIDlets).

Sólo nos resta transferir los archivos JAR y JAD al dispositivo móvil con soporte J2ME. Hay varias formas de hacerlo dependiendo de la marca y modelo del dispositivo.

Si el teléfono tiene soporte de infrarrojos o bluetooth, y la computadora tiene puerto IrDA o bluetooth, se podrá transferir fácilmente el archivo sin necesidad de cable alguno. Si no, se tendrá que recurrir a un cable de datos (consulta el manual del teléfono).

Otra posibilidad es poner los archivos JAR y JAD en un servidor wap o un espacio web y descargarlo desde tu móvil. Para ello es necesario que el dispositivo tenga un navegador wap o web y soporte GPRS para una descarga fiable.



Figura 2.8 Ejemplos de aplicaciones MIDlets de Sun

Para que una aplicación MIDP trabaje en un dispositivo móvil, se requiere un dispositivo que implemente JavaME y MIDP (versiones 1.0 y posteriores) para ser utilizada [26]. Como otros programas desarrollados en Java, tienen la característica, "Escribir una vez, ejecutar en cualquier parte" ("Write once, run anywhere"). Para escribir se puede obtener Sun Java Wireless Toolkit o NetBeans con la extensión Mobility Pack. Para la distribución son necesarios dos archivos, archivo .JAR conteniendo el bytecode del programa y un archivo .JAD que describe los contenidos del archivo .JAR. Un MIDlet tiene que cumplir los siguientes requisitos para poder funcionar en un dispositivo móvil:

- La clase principal necesita ser una subclase de `javax.microedition.midlet.MIDlet`.

- El MIDlet necesita ser empacado dentro de un archivo .JAR.
- El archivo .JAR necesita ser pre-verificado usando un pre-verificador.
- En algunos casos, el archivo .JAR necesita ser firmado digitalmente por un proveedor de dispositivos móviles.

En la Figura 2.9 podemos observar el diagrama de estados el cual muestra el comportamiento de un Midlet en su ejecución.

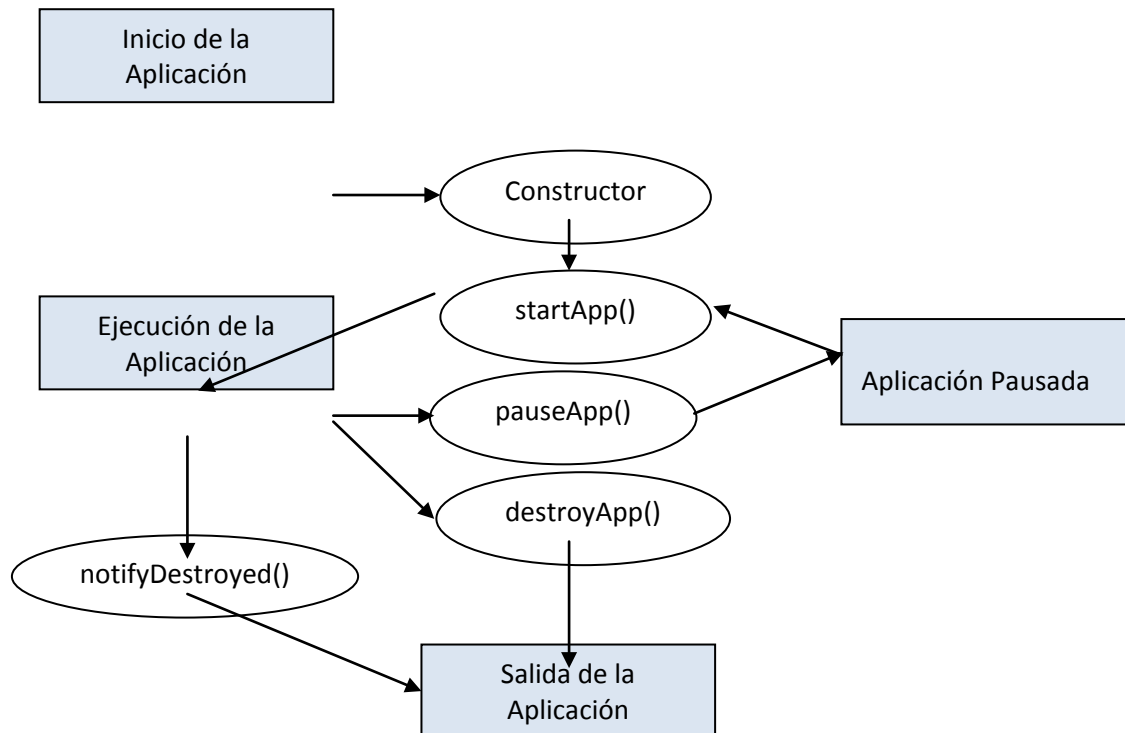


Figura 2.9. Diagrama de Estados de un Midlet

4.2 GPRS

Nuestra aplicación móvil utiliza el servicio GPRS para realizar la descarga de los Objetos de Aprendizaje Móviles, dicho servicio deberá estar configurado el dispositivo móvil que contiene nuestra aplicación.

El sistema GSM es el sistema de comunicación de móviles digital de 2ª generación basado en células de radio. Apareció para dar respuestas a los problemas de los sistemas analógicos.

Fue diseñado para la transmisión de voz por lo que se basa en la conmutación de circuitos, aspecto del que se diferencia del sistema GPRS. Al realizar la transmisión mediante conmutación de circuitos los recursos quedan ocupados durante toda la comunicación y la tarificación es por tiempo. Más adelante veremos como estas limitaciones hacen ineficiente la transmisión de datos con GSM y como GPRS lo soluciona.

Nuestra aplicación realiza la descarga de los Objetos de Aprendizaje Móviles vía GPRS el cual es una nueva tecnología que comparte el rango de frecuencias de la red GSM utilizando una transmisión de datos por medio de 'paquetes'. La conmutación de paquetes es un procedimiento más adecuado para transmitir datos, hasta ahora los datos se habían transmitido mediante conmutación de circuitos, procedimiento más adecuado para la transmisión de voz.

Ventajas del GPRS para el usuario.

Las ventajas que obtiene el usuario con el sistema GPRS son consecuencia directa de las características vistas en el punto anterior.

- Característica de "Always connected": un usuario GPRS puede estar conectado todo el tiempo que desee, puesto que no hace uso de recursos de red (y por tanto no paga) mientras no esté recibiendo ni transmitiendo datos.
- Tarificación por volumen de datos transferidos, en lugar de por tiempo (Por ejemplo: una compañía mexicana cobra \$ 0.04 pesos por Kilobyte descargado).
- Costo nulo de establecimiento de conexión a la red GPRS, frente a los quantum de conexiones existentes actualmente en GSM.
- Mayor velocidad de transmisión. En GSM sólo se puede tener un canal asignado (un "timeslot"), sin embargo, en GPRS, se pueden tener varios canales asignados, tanto en el sentido de transmisión del móvil a la estación base como de la estación base al móvil. La velocidad de transmisión aumentará con el número de canales asignados. Además, GPRS permite el uso de esquemas de codificación de datos que permiten una velocidad de transferencia de datos mayor que en GSM.
- Posibilidad de realizar/recibir llamadas de voz mientras se está conectado o utilizando cualquiera de los servicios disponibles con esta tecnología.
- Modo de transmisión asimétrico, más adaptado al tipo de tráfico de navegación html o wml (un terminal GPRS 4+1 (4 slots downlink y 1 uplink) tendrá cuatro veces mayor capacidad de transmisión de bajada que de subida).

4.2.1 Conexión a redes

Uno de los aspectos más interesantes de los MIDlets es su acceso a Internet a través de una conexión inalámbrica. Las clases e interfaces usadas en MIDP para acceso a red son muy similares a las usadas en J2SE y J2EE.

El CLDC delega las funciones de red en el GCF (Generic Connection Framework). El propósito del GCF es proporcionar una abstracción de los servicios de red. En lugar de forzar a todos los dispositivos a soportar todos los protocolos de red, el GCF describe un marco de trabajo en red del que el dispositivo seleccionará los protocolos y servicios que es capaz de soportar. En realidad, no es el dispositivo el que selecciona las capacidades de red que soportará, sino que esto lo hará el perfil del dispositivo. La responsabilidad del GCF es describir las capacidades de red disponibles en todos los dispositivos así como un sistema extensible en el que los diferentes perfiles de dispositivo pueden soportar un subconjunto de dichas capacidades.

El GCF describe una clase fundamental denominada *Connector*, que será usada para establecer todas las conexiones de red. Los tipos específicos de conexiones son modelados por las interfaces del GCF que son obtenidos a través de la clase *Connector*. Todas estas clases e interfaces están dentro del paquete *javax.microedition.io*. Tal y como vimos anteriormente, las interfaces son las siguientes [23]:

- **Connection** – Una conexión básica que sólo puede ser abierta y cerrada.
- **ContentConnection** – Un flujo (stream) de conexión que proporciona acceso a datos web.
- **DatagramConnection** – Una conexión para manejar comunicaciones orientadas a paquetes.
- **InputConnection** – Una conexión de entrada para las comunicaciones del dispositivo.
- **OutputConnection** – Una conexión de salida para las comunicaciones del dispositivo.
- **StreamConnection** – Una conexión en ambas direcciones para las comunicaciones del dispositivo.
- **StreamConnectionNotifier** – Una conexión especial para notificaciones, que es usada para esperar que se establezca una conexión.

La descripción de las capacidades específicas en cada caso, la tenemos en los perfiles. Por ejemplo, el API de MIDP construye sobre estas interfaces el

soporte para el trabajo en red de los MIDlets. El API de MIDP añade la interfaz `HttpConnection`, que proporciona un componente nuevo en el GCF para conexiones HTTP. Estas conexiones permiten a los MIDlets conectarse con páginas web. La especificación MIDP indica que las conexiones HTTP son el único tipo de conexiones obligatorio en las implementaciones de MIDP.

Siempre usaremos la clase `Connector` para establecer conexiones, independientemente del tipo de conexión. Todos los métodos de la clase `Connector` son estáticos. El más importante de ellos es el método `open()`. Las tres versiones de este método son:

`static Connection open (String name)`

`static Connection open (String name, int mode)`

`static Connection open (String name, int mode, boolean timeouts)`

El primer parámetro de estos métodos es la cadena de conexión. Este es el parámetro más importante porque determina el tipo de conexión que se realizará. La cadena de conexión tendrá el siguiente formato:

`Protocolo:Destino[;Parámetros]`

La primera parte define el protocolo de red, como `http` o `ftp`. El destino es típicamente el nombre de la dirección de red. El último parámetro es una lista de parámetros asociados a la conexión.

Algunos ejemplos de diferentes tipos de cadenas de conexión [25]:

- HTTP – `http://www.usal.es/`
- Socket – `socket://www.usal.es:80`
- Datagram – `datagram://:9000`
- File – `file:/datos.txt`
- Port – `comm:0;baudrate=9600`

La segunda y la tercera versión del método `open()` esperan un segundo parámetro denominado modo (`mode`), que describe el modo de conexión. Este

modo indica si la conexión es abierta para leer, escribir o para ambas cosas. Las siguientes constantes, definidas en la clase Conector, pueden ser usadas para describir el tipo de conexión:

- READ
- WRITE
- READ_WRITE

La última versión del método open() acepta un tercer parámetro, que es un flag que indica si el código que ha llamado el método es capaz de controlar una excepción por tiempo excedido (timeout). Esta excepción es lanzada si el periodo de timeout para establecer la conexión falla. Los detalles de este periodo y cómo es gestionado, es labor del protocolo específico, por lo que no tenemos control sobre el mismo.

El método open() devuelve un objeto de tipo Connection, que es la interface básica para el resto de interfaces de conexión. Para usar undeterminado tipo de interface de conexión, debemos convertir la interfaz Connection que devuelve el método open() al tipo apropiado:

```
StreamConnection conn =  
(StreamConnection)Connector.open(http://www.usal.es/);
```

4.2.2 Entrada/Salida desde el MIDlet

La clase Connector y las interfaces asociadas a esta son usadas para obtener una conexión a la red. Una vez que la conexión está establecida, debemos usar las clases de entrada/salida para leer y escribir datos a través de la conexión. Estas clases son aportadas por el paquete java.io:

- ByteArrayInputStream – Un flujo (stream) de entrada que se gestiona internamente como una matriz de bytes.
- ByteArrayOutputStream – Un flujo (stream) de salida que se gestiona internamente como una matriz de bytes.
- DataInputStream - Un flujo (stream) desde el cual se leen datos como tipos primitivos.
- DataOutputStream – Escribe datos en tipos primitivos en un flujo (stream) en formato binario.
- InputStream – La clase base para todos los flujos (streams) de entrada
- InputStreamReader – Un flujo (stream) desde el que se puede nleer caracteres de texto.

- `OutputStream` – La clase base para todos los flujos (streams) de salida
- `OutputStreamWriter` – Un flujo (stream) en el que se pueden escribir caracteres de texto.
- `PrintStream` – Un flujo (stream) de escritura que facilita el “envío” de datos en forma de tipos primitivos.
- `Reader` – Una clase abstracta para leer flujos (streams) de lectura
- `Writer` – Una clase abstracta para leer flujos (streams) de escritura

Las dos clases más importantes de esta lista son `InputStream` y `OutputStream`, que proporcionan la funcionalidad básica para recibir y enviar datos.

4.2.3 La clase `InputStream`

La clase `InputStream` es una clase abstracta que sirve como base para otras clases de streams de entrada en MIDP. Esta clase define un interfaz básico para leer bytes en modo de stream. El uso normal será crear un objeto de tipo `InputStream` desde una conexión y entonces recoger información a través del método `read()`. Si no hay información disponible en este momento, el stream de entrada usa una técnica como conocida como bloqueo (bloquing) para esperar hasta que haya datos disponibles.

La clase `InputStream` define los siguientes métodos:

- `read()`
- `read(byte b[])`
- `read(byte b[], int off, int len)`
- `skip(long n)`
- `available()`
- `mark(int readlimit)`
- `reset()`
- `markSupported()`
- `close()`

La clase OutputStream

La clase OutputStream es el homólogo de salida de la clase InputStream y sirve como base para todas las clases de streams de salida de MIDP. La clase OutputStream define el protocolo básico para escribir datos a través de un stream de salida. Normalmente crearemos un objeto de tipo OutputStream en una conexión y entonces llamaremos al método write [23, 24]. La clase OutputStream usa la técnica de bloqueo de forma similar a lo visto para la clase InputStream, es decir, se bloquea hasta que los datos son escritos en el stream. Durante el bloqueo la clase OutputStream no permite que más datos sean escritos.

Los métodos de la clase son:

- write(int b)
- write(byte b[])
- write(byte b[], int off, int len)
- flush()
- close()

4.2.4 Haciendo conexión

El primer paso en el desarrollo de cualquier tipo de comunicación de red en un MIDlet involucra el establecimiento de la conexión, esto se realiza a través de static open() el cual es método de la clase Connector [24]. El siguiente ejemplo muestra como abrir una conexión HTTP de una página web:

```
String url = "http://localhost:8080/mi_servlet/OAmt?por="+por+  
"&tipo="+tipo+"&busqueda="+enviar;  
HttpConnection c = (HttpConnection)Connector.open(url);
```

La conexión lanza un objeto HttpConnection para que se pueda realizar la comunicación conexión por un Outputstream o Inputstream. El siguiente paso es obtener un Outputstream o Inputstream sobre la conexión, el cual es necesario actualmente para ejecutar cualquier I/O.

4.2.5 Obtener un Input Stream

El obtener un Stream de una conexión es un poco avanzado. Para el propósito de lo desarrollado en este proyecto de tesis, demostraré como obtener un InputStream, el cual podrá ser usado para leer la página web asociada a la

conexión HTTP. El código necesario para obtener un `InputStream` de una conexión a través del cual se puedan leer datos, es el siguiente:

```
InputStream in = null;  
in = conn.openInputStream();
```

El método `openInputStream()` es todo lo necesario para establecer un `InputStream`. Lo siguiente a este punto es obtener el `Output Stream` para que se pueda iniciar la lectura de los datos de la conexión HTTP.

4.2.6 Leyendo desde el Stream

La clave para la lectura de datos de un objeto `InputStream` es el uso del método `read()`. La versión básica del método `read()` lee un byte de datos desde el `InputStream` y regresa a este como un entero. Las llamadas subsecuentes al método `read()` resulta en bytes adicionales de datos iniciando la lectura desde el `Stream`, continuando hasta el final `Stream`, el cual resulta en el retorno de el método `read()`. El siguiente es un ejemplo de cómo el código dentro de un ciclo (loop) lee líneas de datos de un `InputStream` e imprime la lectura:

```
<  
StringBuffer data = new StringBuffer();  
int ch;  
while ((ch = in.read()) != -1) {  
if (ch != '\n') {  
// Lee la línea caracter por caracter  
data.append((char)ch);  
}  
else {  
// Imprime la línea como cadena de datos  
System.out.println(data.toString());  
// Limpia la cadena para la siguiente línea  
data = new StringBuffer();  
}  
}
```

En éste código la variable *data* es un *string buffer* que es usado para retener por un momento una línea de datos. El método *read()* es llamada para leer caracteres del Stream. Dentro del ciclo *while*, el actual caracter es comparado con *'\n'* (nueva línea) para ver si el final de la línea ha sido alcanzado. Si es así, ya sabemos que tenemos acumulada una línea de texto completa, y si no hay error se imprime y se limpia el *string buffer*. [22,24]

Este código representa la premisa básica de cómo podemos leer los datos a través de un Input Stream de una conexión HTTP.

4.3 Servlets o aplicación Web

Cuando se habla de aplicaciones para Web comúnmente escuchamos hablar de JAVA, y con ello sus aplicaciones más conocidas, los Applets, que son programas que se pueden cargar a través de una red y que se ejecutan de igual forma en cualquier plataforma, todo ello gracias a las potentes características de JAVA. Hasta hace poco, JAVA se utilizaba básicamente para dotar a las páginas WEB de una mayor interactividad mediante los Applets, y por tanto solo actuaba sobre el lado cliente. Pero el lado servidor también puede beneficiarse de todas las ventajas que ofrece JAVA, gracias a los *Servlets*.

Los *Servlets* se diferencian de los Applets básicamente en que se ejecutan en el servidor y en que no presentan ningún tipo de interfaz gráfica, puesto que se encargan de hacer el trabajo oculto.

Los servlets son programas atienden peticiones de un cliente teniendo al servidor como el encargado, pero escritos en Java y con la ventaja de explotar todas las bondades de java. Por ejemplo, un servlet puede ser responsable de tomar los datos de un formulario HTML y enviarlos a una base de datos para actualización de la misma.

4.2.1 Concepto de Servlet

La palabra servlet deriva de otra anterior, applet, que se refería a pequeños programas que se ejecutan en el contexto de un navegador web. Por contraposición, un servlet es un programa que se ejecuta en un servidor.

El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

4.2.2 Ventajas de los Servlets

Las ventajas de los Servlets ante los CGI son:

Desempeño: Los servlets son más rápidos que los CGI debido a que utilizan *threads* en lugar de *procesos*.

Una de las quejas más comunes de los CGI/Perl es el desperdicio de los recursos del sistema. En un ambiente típico CGI, cada petición crea un nuevo proceso el cual carga el interprete de Perl. El intérprete de Perl carga el script CGI, lo compila, y lo ejecuta. Además, si la aplicación se comunica con la BD, una nueva conexión será necesaria para ser hecha por cada CGI.

Los Servlets inician un nuevo thread, *hilo*, (más que un nuevo proceso) con cada petición. Cada servlet es cargado una vez y usado más y más. Nota que a diferencia de los CGI, los servlets requieren que se cuente con una Máquina Virtual de Java (JVM) corriendo sobre el servidor todo el tiempo. Para sitios ocupados, ésta permite a los servlets usar mucho menos los recursos del sistema e incrementar el desempeño.

Portabilidad: Los servlets son tan portables como cualquier otra aplicación de Java.

Muchos sitios están buscando una solución de portabilidad. Tal vez desearían para su desarrollo usar Linux o Windows NT mientras su servidor de producción corra bajo Solaris. Ellos pueden querer vender su aplicación Web para muchos clientes como para plataformas sea posible. Hay versiones de Perl para muchas variantes de Unix como para Windows 95 y NT. Mientras que Perl para Unix no es ciento por ciento compatible con el Perl para NT, los CGIs escritos en Perl pueden ser escritos para trabajar sobre ambos. El llevar CGIs a Windows NT no es difícil de hacer, pero nunca funcionarán sin realizar al menos una modificación. Por otro lado, los CGI pueden contener comandos de Unix que no trabajarán sobre Windows NT.

La portabilidad con los servlets de Java es más simple. Java fue diseñado para ser portable a través de todas las plataformas, permitiendo que las aplicaciones sean movidas fácilmente de un sistema operativo a otro.

Seguridad: La seguridad puede ser un problema cuando se desarrollan CGIs. El área de mas preocupación es el proceso de estrada del usuario. Este podría ser desde formas o desde datos dentro de un URL. Muchos CGIs escritos en Perl son vulnerables a ataques donde la finalidad del usuario es trucar el CGI ejecutando algún comando sobre el servidor. Los Servlets no están en riesgo de correr comando de shell no planeados. Los lenguajes compilados como Java (o C) proveen mejor seguridad que los lenguajes que interpretan scripts.

Los Servlets son archivos de clases compilados mientras que un CGI/Perl es manipulado en su forma de código fuente. Dependiendo quien tenga acceso al servidor Web, se puede elegir entre instalar o no el código fuente.

Desarrollo: El desarrollo con Perl es simple. Lo único que se necesita es un editor y mucha documentación de Perl.. Mientras que a muchos desarrolladores les agrada este ambiente, hay una creciente proliferación de herramientas IDE (Integrated Development Environment) para desarrollo. En ese campo, Java por encima de Perl. Hay IDEs disponibles desde Sun, Microsoft, Symantec y muchas otras, proporcionando simples entornos GUI. Las IDEs pueden proveer útiles herramientas fáciles de aprender para nuevos desarrolladores.

Compartiendo el trabajo entre los desarrolladores es además es más fácil con Java, o puede asignarse en paquetes individuales. Cuando se trabaja con una aplicación CGI/Perl donde dos o mas desarrolladores estuvieron trabajando simultáneamente, la aplicación normalmente es manejada de un desarrollador a otro. El desarrollo de aplicaciones Java es mucho mejor para proyectos largos.

4.3.1 Arquitectura de los Servlets

El principal componente de la Servlet API es la interfaz Servlet. Todos los servlets implementan esta interfaz directamente, por medio de la extensión de la clase que la implementa, **HttpServlet**. Esta interfaz está provista de métodos que manipulan a los servlets y la comunicación con sus clientes. Ésta clase contiene una serie de métodos predefinidos, que son invocados en respuesta a eventos:

1. **Inicialización.** Se carga en memoria una instancia del servlet. La instanciación se produce la primera vez que el servlet es cargado en memoria por parte del servidor de aplicaciones. Como consecuencia de este evento se ejecuta el siguiente método:

```
public void init(ServletConfig config) throws ServletException
```

2. **Petición:** el cliente realiza la petición al servlet. Dentro de la versión 1.1 y en función del método de invocación usado en HTML (DO o GET) se ejecutará uno de los siguientes:

```
public void doGet(HttpServletRequest petition, HttpServletResponse  
respuesta) throws ServletException, IOException
```

```
public void doPost(HttpServletRequest petition, HttpServletResponse  
respuesta) throws ServletException, IOException
```

El objeto 'petition' es enviado por la JVM del servidor de aplicaciones y representa (encapsula) información de la petición HTTP. Más adelante

veremos para que puede servir el objeto 'respuesta', por de pronto nos sirve decir que en este objeto se encapsulan los servicios para generar la salida. También tiene la posibilidad de usar:

```
public void service(HttpServletRequest petition, HttpServletResponse respuesta) throws ServletException, IOException
```

3. Destrucción del servlet: la contraimagen de *init()*, el servlet va a ser destruido (normalmente por la detención del servidor). No se invoca cuando se termina la petición, sino cuando se va a descargar el servlet de memoria:

```
public void destroy()
```

Cuando un servlet es llamado desde un cliente, este recibe dos objetos: **ServletRequest** y **ServletResponse**. La interfaz ServletRequest se encarga la comunicación desde el cliente al servidor, mientras que la interfaz ServletResponse atiende la comunicación desde servlet al cliente.

La interfaz ServletRequest permite al servlet acceder a información como, los nombres de parámetros pasados por el cliente, el protocolo usado por el cliente, y los nombres de los host remotos que hacen la solicitud y el servidor que la recibe. Est interfaz permite a los servlets el acceso a métodos que permiten manejar la presentación de la respuesta como salida en el navegador, a través de los cuales consiguen los datos desde el cliente que usa protocolos como HTTP POST , etc..

La interfaz ServletResponse proporciona al servlet los métodos para contestarle al cliente. Permite al servlet configurar la forma de salida de los datos para el cliente, ServletOutputStream que permite enviar la replica de datos como respuesta. La subclases de ServletResponse le dan más capacidad al servlet para responder.

Por ejemplo:

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
public class ObjetosAprendizaje extends HttpServlet{
```

```
    PrintWriter out;
```

```
    public void init(){ } // motor del servlet para cargar la petición
```

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException{
```

```
try{
```

```
    String tipo=req.getParameter("formato ");
```

```
    String busqueda=req.getParameter("busqueda");
```

```
    res.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
```

```
    out.println("Busqueda:" + busqueda + "\n Formato: "+formato);
```

```
    out.close();
```

```
} catch(Exception ee){out.write(ee.toString());}
```

```
Public void doPost(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException{
```

```
try{
```

```
    doGet(request, response);
```

```
}catch (Exception e) { e.printStackTrace(); }
```

```
}
```

```
Public void destroy(){} //destruimos y liberamos los recursos abiertos
```

```
}
```

3.1 Estructura básica de un Servlet

Tanto doGet como doPost reciben como parámetros objetos de tipo HttpServletRequest y HttpServletResponse.

De la clase `HttpServletRequest` nos interesa el método `getParameter(String nombreParametro)`, que devuelve el valor del parámetro de Búsqueda y Formato. Por ejemplo, si visitáramos la URL

`http://localhost:8080/mi_servlet/OAm?formato=MP3&busqueda=REDES`

(vemos que es una petición GET, porque los parámetros se pasan en la URL) el servlet “ObjetosAprendizaje” (*Figura 3.1*) podría llamar a `getParameter("formato")` para obtener el Formato o `getParameter("Busqueda")` para obtener la Búsqueda.

Las clases e interfaces descritas conforman a un servlet básico. Pero existen métodos adicionales que provee la API con la capacidad para controlar sesiones o múltiples conexiones, entre muchas más aplicaciones

El ciclo de vida de un Servlet se divide en los siguientes puntos:

1. El cliente solicita una petición a un servidor vía URL.
2. El servidor recibe la petición.
 1. Si es la primera, se utiliza el motor de Servlets para cargarlo y se llama al método `init()`.
 2. Si ya está iniciado, cualquier petición se convierte en un nuevo hilo. Un Servlet puede manejar múltiples peticiones de clientes.
3. Se llama al método `service()` para procesar la petición devolviendo el resultado al cliente.
4. Cuando se apaga el motor de un Servlet se llama al método `destroy()`, que lo destruye y libera los recursos abiertos.

4.3.2 Ejemplos de aplicación de los servlets

Algunas de las aplicaciones de servlets incluyen:

Procesamiento de datos enviados con HTTPS a partir de una forma HTML, incluyendo la orden de compra o datos de una tarjeta de crédito. Un servlet como este podría ser parte de una orden de entrada al sistema de procesamiento de datos, haciendo la respectiva actualización en la base de datos, y quizá hasta un sistema de pago en línea.

Interacción múltiple entre personas. Un servlet puede manipular múltiples peticiones al mismo tiempo; podrían sincronizar dichas solicitudes de sistemas que dan soporte, por ejemplo conferencias en línea.

Redireccionamiento de peticiones. Los servlets pueden reenviar las peticiones a otros servidores y servlets. Esto permite que al ser usados se balancee la

carga entre varios servidores que atiendan al misma tarea, según el tipo de proceso que deban atender.

Dado que pueden manejar múltiples peticiones en forma concurrente, es posible implementar aplicaciones de colaboración, como por ejemplo una aplicación de videoconferencia.

CAPITULO 5: PLANEACIÓN NUTRICIONAL

5 Introducción

Casi todos los países están experimentando una epidemia de obesidad. En los países desarrollados, la obesidad no sólo es común en la juventud sino que se está volviendo cada vez más común entre los adultos jóvenes y los niños. El hecho es que una dieta pobre puede tener un impacto perjudicial en la salud. Hay una necesidad de proporcionar un asesoramiento dietético seguro, basado en la evidencia y de fácil acceso para todos, especialmente para los niños.

Por lo tanto, se presenta una nueva aplicación móvil que nos permite obtener un plan personalizado y automático para apoyar a las personas a mejorar su salud en general.

5.2 Arquitectura

En la figura 5.1, se muestra la arquitectura propuesta para el sistema alimentario. El módulo para obtener un plan personalizado y automático para apoyar a las personas a mejorar su salud en general se implementan como servicios web utilizando lenguajes estándar. Toma de datos desde un dispositivo móvil, y esta cuestión se envía al servicio web que devuelve un plan específico obtenidos mediante sistemas de DLV^K.

La arquitectura utilizada consta de los siguientes módulos:

- Mobile Planning Nourishing (mPN): es la interfaz del dispositivo móvil para el usuario, que es responsable de comunicar a través de GPRS a los servicios web.

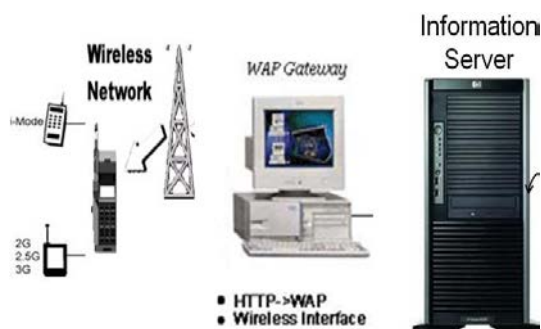


Figura 5.1. Arquitectura general

- Servicio Web del Sistema de Planeación K: es el servicio web codificado en java, servido por apache tomcat, que se dedica a satisfacer todas las peticiones del mPN a la página y envía todas las solicitudes al motor de integración del conocimiento para ser resueltas.

- Motor de Integración del Conocimiento: es el agente encargado de encontrar el mejor plan de dieta. Se encarga de recibir los datos personales y definir el rango de valores de la dieta.

En caso de que se necesite una re- planificación se encarga de enviar el nuevo estado inicial para ser procesado por dlV-k.

- Repositorio de comidas: Es la base de datos en donde DLV^k busca los alimentos que más se acercan a los valores propuestos.

5.3 Aplicación

De una manera más general, el usuario en su dispositivo móvil de su entrada / ella su perfil información, el sistema proporciona un sistema de comunicaciones móviles con el servicio Web mediante una conexión GPRS (para la economía y la accesibilidad de los usuarios en caso de México 4 centavos por kilobyte), en el servicio web del perfil del usuario se utiliza para calcular los valores de una dieta de referencia que se envían al motor de la integración del conocimiento. Este motor encuentra el plan más adecuado usar la alimentación desde el depósito de los alimentos. El plan se devuelve al usuario por la conexión previamente establecido para el dispositivo móvil en el que se muestra el plan.



Figura 5.2 Registro de datos personales

A continuación, se muestra (en la figura 5.2) la interfaz de la aplicación móvil desarrollada para recopilar toda la información necesaria antes de calcular cualquier plan (edad, peso, estatura y actividad física) los cuales son enviados al Servicio Web del Sistema de Planeación K.

Aquellos son necesarios para calcular el IMC (índice de masa corporal se define como la masa corporal del individuo dividido por el cuadrado de su altura) y comprobar si este está entre el rango normal de acuerdo con la Tabla 5,1 [25], y estimar el gasto energético basal (BEE), utilizando las ecuaciones de Harris-Benedict [11]. The B.E.E se multiplica por el valor de la actividad física (1 a 2) para calcular los requerimientos de calorías por día (kcal / día) como en el ejemplo (en la figura 5.3)

Tabla 5.1 BMI normal para cada edad

Edad	Hombre	Mujer
5	13-16.7	12.7-17
6	13-17	12.7-17.3
7	13.2-17.4	12.7-17.7
8	13.3-17.9	12.9-18.3
9	13.5-18.4	13.1-19
10	13.7-19.1	13.5-19.8
11	14.1-19.9	13.9-20.7
12	14.5-20.8	14.4-21.7
13	14.9-21.7	14.9-22.7
14	15.5-22.6	15.4-23.5
15	16-23.5	15.9-24.1
16	16.5-24.2	16.2-24.5
17	16.9-24.9	16.4-24.8
18	17.3-25.4	16.4-25
19	17.6-25.4	16.5-25
20-60	18.5-25	18.5-25



Figura 5.3 BMI

Ahora, podemos calcular el plan para alcanzar el requisito de calorías (carbohidratos 55%, proteínas 15% y 30% de grasas), dividido en tres comidas: desayuno, almuerzo y cena. Estos están separados para alcanzar el objetivo de volver a planificar una o dos si plan es cambiado manualmente por el usuario o si solicito uno nuevo. (Ver figura 5.4)

Para cada comida, se puede definir el dominio de la siguiente manera:

fluents:

eaten(T) requires meal(T).

%Count calories

idr(Z,V,Y,W) requires #int(Z), #int(V), #int(Y), #int(W).

% Initial set calories,proteins,fat,carbs

idrE(E,P,I,C) requires #int(E),#int(P),#int(I),#int(C).

idrMax(X,R,B,A) requires #int(X),#int(R),#int(B),#int(A).

end.

actions:

eat(F,T) requires meal(T),item(F,T,E,P,I,C).



Figura 5.4 Interfaz principal y la función de planificación.

En nuestro caso, la única acción que el usuario puede hacer es comer o tener una comida como la siguiente:

1. Primero la sopa
2. El plato principal
3. Una bebida al final.

El objetivo es optimizar la cantidad de carbohidratos (g), proteínas (g) y grasas (g), que deben estar entre los valores de $Idre$ (suficiente) y $idrMax$ (máximo permitido), y cerca del requerimiento calórico. Esta gama se establece en un 80% para $Idre$ y 110% para $idrMax$ de los valores ideales calculados con anterioridad.

Las siguientes reglas de las declaraciones de las acciones y fluentes (siempre) describen las transiciones y las limitaciones de los estados iniciales del dominio (por razones de espacio no presentamos todos los códigos). Por último, en el goal se define el objetivo a alcanzar y la duración del plan.

always:

executable eat(F,T) if not eaten(T),item(F,T,E,P,I,C).

nonexecutable eat(F,2) if not eaten(1).

nonexecutable eat(F,3) if not eaten(2).

forbidden idr(Z,V,Y,W), idrMax(X,R,B,A), $Z > X$, $V > R$, $Y > B$, $W > A$.

caused idr(Z,V,Y,W) after eat(F,T), item(F,T,E,P,I,R), idr(A,B,C,D), $+(A,E,Z)$, $+(P,B,V)$, $+(C,I,Y)$, $+(R,D,W)$.

caused eaten(T) after eat(F,T).

caused end after idr(Z,V,Y,W), idrE(E,P,I,C), $Z > E$, $V > P$, $Y > I$, $W > C$.

inertial eaten(T).

inertial idrE(E,P,I,C).

inertial idrMax(X,R,B,A).

noConcurrency.

% las condiciones iniciales del usuario se establecen automáticamente por el Servicio Web del Sistema de Planeación K, pero continuando con el ejemplo se tienen estos hechos para el desayuno:

initially:

idr(0, 0, 0, 0).

idrE(417,13,12,51).

idrMax(500,17,15,63).

% Las condiciones adicionales se establecerán aquí si el usuario decide comer alimentos distintos de los sugeridos para el programa (véase el gráfico 5.5) entonces el usuario puede modificar esta propuesta.

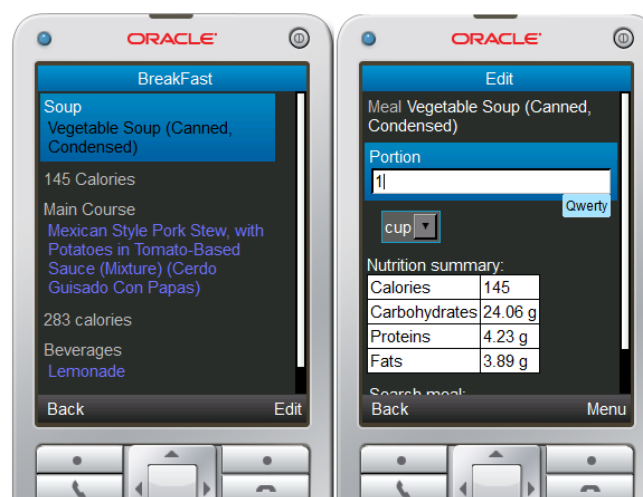


Figura 5.5 Interfaz para modificar la dieta

```

% foods

item(soup1,381,29.25,19.13,22.02).

item(soup2,102,3.05,2.93,17.93).

...

item(mainCourse1,443,9.74,32.02,31.88).

item(mainCourse2,438,13.5,31.37,27.35).

...

item(drink2,136,0.26,0.07,35.18).

item(drink3,2,0.28,0.05,0.09).

...

food(F):- item(F,_,_,_,_).

meal(1).

meal(2).

meal(3).

goal: end?(4)

```

En el **goal**, se define el objetivo a ser alcanzado y la duración del plan.

5.2.1 Ejecutándolo en DLV^K

Supongamos que el background knowledge, la planificación del programa y el dominio inicial se separa en archivos `dieta.dl`, `dieta.plan` y `initial.plan`, respectivamente. La ejecución del comando ejecutado por el Motor de Integración del Conocimiento:

```
# dlv dieta.dl dieta.plan initial.plan -FP -N=1000
```

Calcula el resultado para el desayuno:

```
PLAN: eat(soup7,1); eat(mainCourse7,2); eat(drink5,3); (no action)
```

Para la comida y la cena respectivamente:

PLAN: eat(soup5,1); eat(mainCourse2,2); eat(drink9,3); (no action)

PLAN: eat(soup10,1); eat(mainCourse8,2); eat(drink2,3); (no action)

Este plan es el más adecuado y cercano a los valores ideales propuestos (como se puede ver en la Figura 5.6). La opción-N de línea de comandos = 1000 le dice a dlw que ejecute solo las operaciones que no sobrepasen un máximo valor de 1000. La Opción-FP se utiliza para invocar el front-end de planificación. Alternativamente, se pueden utilizar las opciones-FPopt y FPsec, que explícitamente hacen cumplir la planificación optimista y segura, respectivamente.

Para la planificación de problemas con un único estado inicial de un dominio determinista, estas dos opciones no devuelven resultados diferentes, pues tanto el plan seguro como el determinista coincidirán.

Esta solución es clara, sencilla y todas las reglas de IR y CR tienen que satisfacer los requisitos de seguridad para los literales de tipo negados por defecto, es decir, cada variable se produce en un tipo de defecto negado literal y tiene que ocurrir en al menos un tipo no negado literal o dinámico literal.

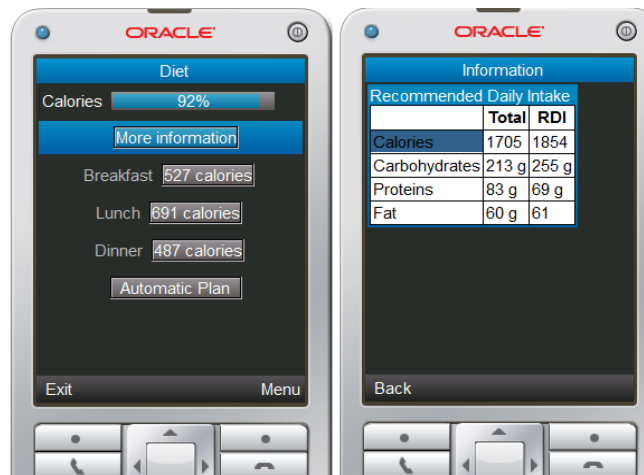


Figura 5.6 Resumen de la dieta

CONCLUSIONES Y TRABAJO A FUTURO

A lo largo de esta tesis, hemos presentado una serie de nuevas tecnologías de la información y comunicación que están revolucionando muchas de las áreas que cotidianamente hemos cultivado. De manera particular, en el presente trabajo nos hemos dado a la tarea de desarrollar un proyecto en el área de planeación e incorporando las tecnologías móviles.

Hemos logrado integrar un conjunto de herramientas que nos permiten desarrollar y llevar el conocimiento a los usuarios de teléfonos móviles de una manera sencilla. La tecnología J2ME nos ofrece un ambiente de desarrollo para dispositivos móviles, permitiéndonos desarrollar la herramienta de planificación de dietas.

Además, hemos presentado una aplicación novedosa que incorpora el paradigma ASP para la planificación de un problema cotidiano como es el de llevar una dieta saludable.

Se ha demostrado que el lenguaje k es muy expresivo en términos de planeación, permitiendo incluso planear problemas difíciles con la alternativa de tener precondiciones de las acciones. Con la implementación actual de DLVK se proporciona un sistema que permite una amplia variedad de dominios que pueden ser codificado y resueltos.

Con la aplicación desarrollada para ayudar a la cuestión nutricional, pudimos crear una herramienta que permite a todos tener acceso a un plan de alimentación personalizado sin la necesidad de tener un dietista personal. Sin embargo, existen algunas limitaciones o posibilidades de mejora que sólo investigaciones adicionales resolverán.

Algunas de las cosas que se mejorarán a futuro:

- Incrementar las comidas a 5
- Añadir la opción de comer aperitivos e incluirlos en la planificación si se desean.
- Incrementar el repositorio de comidas
- Migrar la aplicación a plataformas más comerciales como Android o IOS

BIBLIOGRAFÍA

- [1] NWANA, H. Software Agents: An Overview. Knowledge Engineering Review. Cambridge University Press. v.3, p.1-40. 1996.
- [2] Stuart Russell and Peter Norving. "Inteligencia Artificial: Un enfoque Moderno", Prentice Hall, 2002.
- [3] Gelfond M. and Lifschitz V. The Stable Model Semantics for Logic Programming, 5th Conference on Logic Programming, R. Kowalski and K. Bowen, Mit Press. 1988
- [4] Leone N. The DLV Project: A disjunctive Datalog System
"<http://www.dbai.tuwien.ac.at/proj/dlv/>"
- [5] Kowalski Robert, Logic for Problem Solving. Artificial Intelligence series. The computer science library, 1990.
- [6] Katsuno H. and Mendelzon A. Propositional knowledge base revision and minimal change. Artificial Intelligence 52:263-294, 1991
- [7] JENNINGS, N.; WOOLDRIDGE, M. Agent Technology - Foundations, Applications, and Markets. Springer-UNICOM. 1998.
- [8] MAES, P. Modeling Adaptive Autonomous Agents. Artificial Life Journal, v.1, n 1&2, pp.135-162. MIT Press, 1994.
- [9] S. Franklin et A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

- [10] Berney, B. Software Agents: A review. Manchester Metropolitan University, 1996.
- [11] Rao, A.S., Georgeff. M.P. (1995). BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agents Systems (ICMAS-95).
- [12] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson, Learning and Executing Generalized Robot Plans. Artificial Intelligence, 3 (1972) 251-288.
- [13] Julio Cesar de Mello e Souza. El Hombre que Calculaba. Ed. Piedra Santa
- [14] T. Eiter and M. Fink and G. Sabattini and H. Thompits, Considerations on Updates of Logic Programs, Proceedings of the seventh European Workshop on Logic in Artificial Intelligence (JELIA 00), 2000, M.O. Aciego and I.P. de Guzman and G. Brewka and L. M. Pereira, Springer LNAI, Vol. 1, 1919.
- [15] Zacarias F. Belief Revision and Updates in CommonSense Reasoning. Tesis de doctorado, UDLAP 2004.
- [16] J.J. Alferes and J.A. Leite and L.M. Pereira and H. Przymusinska and T.C.Przymusiski, Dynamic logic programming, Procs, of 6th the International Conference on Principles of Knowledge Representation and reasoning(KR 98), Anthony Cohn and Lenhart Schubert and Stuart Shapiro, Morgan Kaufmann Publishers
- [17] David Pearce. From here to There: Stable Negation in Logic Programming. What is Negation. 1999, D. Gabbay, H. Wansing. Kluwer Academic Publishers, Dordrecht.
- [18] DLV-Complex: The Project: S. Cozza, F. Calimeri, G. Lanni and N. Leone <https://www.mat.unical.it/dlv-complex>
- [19] DLV_k <http://www.dbai.tuwien.ac.at/proj/dlv/K/>

[20] Felipe Vidueira Estévez, Introducción a J2ME. Departamento de Ciencias de la Computación de la Universidad de Alcalá.

<http://www.mobigame.uah.es/antiguos/2003/cd/ii-tm/ij2medocumento.pdf>

[21] MC. Juan Carlos Olivares Rojas, J2ME, Department of computer and system, Instituto Tecnológico de Morelia.

<http://www.slidefinder.net/J/J2ME/25013509>

[22] Miguel Montesinos, Alberto Romeu, Francisco J. Peñarrubia, José Manuel.

Aplicación de Turismo sobre teléfonos móviles. Valencia.

[23] Sergio Gálvez Rojas y Lucas Ortega Díaz. Java a tope: J2ME (Java 2 Micro

Edition). Universidad de Málaga, España.

[24] Sing Li and Jonathan Knudsen. Beginning J2ME: From Novice to Professional,

(3rd Ed.). Apress, 2005.

[25] Instituto Tecnológico Autónomo de México-ITAM

<ftp://biblioteca.itam.mx/pub/investigadores/gigola/DiploLog-III/RUT33.PPT>

[26] <http://www.java2s.com/Code/Java/J2ME/Downloadandviewapngfile.htm>

[27] Michael Morrison, "Wireless Java with J2ME in 21Days", Edit: SAMS, International Standard Book Number: 0-672-32142-4, 201 West 103rd St., Indianapolis, Indiana, 46290 USA, 2007

[28] Alan Kavanagh, John Beckmeyer, GPRS Networks. Osborne Publishing, 2002