Facultad de Matemática y Computación
Universidad de la Habana

Temas de Simulación

Autores
Dr. Luciano García Garrido
Lic. Luis Martí Orosa
Lic. Luis Pérez Sánchez

Índice general

Capítulo 1: Sistemas, Modelos y Simulación

Todo problema tiene un dominio en el cual se presenta y exige la construcción de un sistema, es decir, la selección y/o definición de entidades (mediante propiedades), relaciones estructurales y relaciones funcionales (entre estas entidades), las que se consideran involucradas y relevantes a la solución del problema.

Ejemplo: sistema planetario.

Entidades los planetas (propiedades).
Relaciones orden (distancia creciente) con respecto al Sol.
Procesos movimientos de rotación y traslación.

Los sistemas se convierten en objeto de investigación para resolver el problema y se construyen y reconstruyen en varios niveles de abstracción según un proceso denominado modelación que
suministra a diferentes niveles de abstracción representaciones estructurales y funcionales a las que se denomina modelos del sistema.

El objetivo de crear modelos de un sistema tiene como fin esencialmente observar y/o modificar y/o controlar en condiciones ideales su conducta o dinámica.

Ejemplos,

- Modelos de la astronomía primitiva (nómadas y agricultores): Cielo con estrellas fijas en el cual se distinguían los 7 cuerpos celestiales móviles (planetas) visibles sin telescopio: Sol, Luna, Mercurio, Venus, Martes, Júpiter, Saturno.

- Sistema Ptolemaico, Ptolomeo de Alejandría (140 NE), Modelo en el cual la tierra es el centro del universo.

- Sistema Copernicano, Primer esquema de nuestro modelo actual.

- Sistema Newtoniano, Muestra mayor precisión.

- Teoría de la relatividad, Exactitud en los cálculos, se puede predecir el movimiento de los astros.

La simulación es el proceso mediante el cual los procesos (dinámica) de un sistema son observados y/o modificados y controlados mediante modelos apropiados a tales fines.

La dinámica de un sistema se considera simulada sólo en la medida en que esta dinámica, mediante su observación controlada, puede ser modificada con vistas a verificar principios o leyes y/o a determinar su forma más satisfactoria de realización. Siendo la computadora es el instrumento ideal para tales fines la simulación es esencialmente computacional.

1.1. Concepto de Sistema

Dominio de un problema = estructuras + procesos, tal que, Estructuras: entidades (objetos, componentes) + relaciones (nexos, interacciones) entre las entidades y Procesos: (operaciones, acciones, procedimientos) que realiza una estructura.

Un sistema esta constituido por estructuras y procesos de un dominio de un problema las cuales se obtienen por abstracción a partir del dominio orientada por el problema.

Diferentes tipos de sistemas:

- sistemas naturales: constituidos por estructuras y funciones que el hombre logra distinguir en la realidad. Ejemplos: partículas, átomos, moléculas, seres humanos, árboles, hormigueros, manadas.

- sistemas socio-políticos: productos del desarrollo histórico-social del hombre. Ejemplos: comunidad patriarcal, tribu, país, sociedad científica.

- Sistemas tecnológicos: construidos por el hombre. Ejemplos: máquinas, fábricas, complejos industriales, sistemas de comunicación, hospitales.

Sistemas simbólicos y formales: construidos por el ser humano para intercambiar, procesar y almacenar información (expresiones de un lenguaje con significado asociado) y/o describir y manipular abstractamente modelos de sistemas.

Ejemplos:

- Lenguajes naturales: español, inglés.

- Lenguajes y teorías científicas: física, medicina, sistema de ecuaciones diferenciales.

- Lenguajes y teorías formales: lógica de primer orden, lenguajes de estructura de frase.

- Lenguajes de programación y programas: C, Java, Prolog, sistemas operativos, sistema de razonamiento automatizado.

El objeto de la investigación científica son los sistemas. Por ejemplo:

Biología investiga los sistemas vivientes (organismos). La biología destaca claramente las dos dimensiones (estructura-proceso) de todo sistema:

- Anatomía: investigación de las estructuras de las plantas, animales y otros entidades biológicas. áreas: (macro) anatomía, histología (estructura de los tejidos), citología (estructura de las células).

- Fisiología: investigación de los procesos biológicos en los organismos.

Computación Diseña y analiza la estructuras y procesos de sistemas computacionales.

- Estructuras: lenguajes libres de contexto, redes de computadoras.

- Procesos: reconocimientos de expresiones de un lenguaje, interacción de recursos en un sistema operativo.

En general se hace una gran distinción en el estudio de los sistemas teniendo en cuenta la introducción o no del tiempo en dicho estudio y se denomina en general sistema dinámico a todo aquel sistema en el que el tiempo es un factor indispensable en la solución del problema bajo consideración.

1.1.1. Sistemas Dinámicos

Sistemas que se estudian considerando el tiempo como una dimensión necesaria para describir tanto su estructura como sus procesos.

Sistema dinámico estacionario: la estructura del sistema no cambia y los procesos ocurren en el tiempo pero no se considera la dependencia del tiempo.

Sistema dinámico no-estacionario: la estructura del sistema cambia con el tiempo y los procesos del sistema ocurren en y dependen de el tiempo.

A partir de ahora salvo que hagamos la distinción necesaria cuando hablemos de un sistema, la referencia es a un sistema dinámico.

1.1.2. Sistema y Medio

Un sistema existe (funciona, se desarrolla, interactúa) en un medio (ambiente). Medio es un sistema en el cual actúa, funciona, se desarrolla, con el cual interactúa en general un

sistema bajo estudio. Qué sistema es medio para un sistema bajo investigación depende del problema que se trata de resolver.

Un sistema se considera cerrado si no se le asocia un medio, de lo contrario se le considera abierto.

Salvo que se haga la especificación correspondiente consideraremos en lo que sigue sólo sistemas abiertos.

La relación de un sistema con su medio tiene lugar a través de sus entradas y salidas. Por sus entradas un sistema recibe información (sustancia, energía) de su medio. Por sus salidas el sistema responde al medio después de haber procesado la información (sustancia, energía) recibida. Como entrada también el sistema puede recibir información de su respuestas al medio (retroalimentación, feedback).

Un sistema experimenta cambios o transformaciones de acuerdo con su interacción con el medio.

Lo esencial de estos cambios o transformaciones para el sistema es recogido en el concepto de estado del sistema. En general la acción del medio sobre un sistema (entrada) y el estado de este último son parámetros necesarios para predecir su comportamiento (estado) futuro y/o su salida o respuesta al medio. Luego, la dinámica de un sistema se describe en términos de una función de cambio de estado y una función de emisión de salida.

1.1.3. Sistemas deterministas, no-deterministas y aleatorios

Una distinción esencial entre los sistemas tiene lugar a partir de la información o conocimiento que tengamos de los mismos para su estudio. Tanto la estructura como los procesos que tiene lugar en un sistema nos puede resultar conocidos, desconocidos o parcialmente conocidos (sistemas con información completa o incompleta); nuestra capacidad de observación de su estructura y sus procesos puede ser limitada y sujeta a error (sistemas sujetos a error); los procesos asociados pueden exhibir cierta incertidumbre (sistemas con incertidumbre).

En los sistemas **deterministas (no-deterministas)** en general conocemos la estructura del sistema y las funciones de cambio de estado y de salida son funciones deterministas (no-deterministas).

Son sistemas con información completa y sin incertidumbre. En estos casos el comportamiento del sistema se describe por algoritmos o procedimientos efectivos asociados a dichas funciones.

Un problema con ciertos sistemas deterministas es el grado de complejidad estructural y funcional que exhiben. En tales casos enfrentarlos con algoritmos o procedimientos efectivos resulta impracticable. En ocasiones es posible aproximarnos a sus soluciones con enfoques que reduzcan el problema a casos tratables, prácticos y relevantes.

En ocasiones la aproximación aleatoria es una alternativa para enfrentar la complejidad estructural y funcional del sistema determinista (con información completa y sin incertidumbre) asociado a la solución de un problema.

Sistemas con información incompleta e incierta: La estructura y la función del sistema son inciertas o parcialmente conocidas, la observación de su comportamiento puede estar sujeta a error o ser incierta.

En estos casos podemos describirlos por formalismos probabilistas, difusos o de "computación suave" (softcomputing).

1.2. Concepto de Modelo

Un sistema se define en términos de un problema que deseamos resolver. La solución de un problema particular siempre entraña una abstracción de estructuras y procesos que son relevantes al problema, es decir, indispensable para el planteamiento y el hallazgo de una solución al problema.

Los sistemas naturales, sociales y tecnológicos tienen realización física. Como tales son dominios donde definimos sistemas en función de los problemas que deseamos resolver en dichos dominios.

Diferentes problemas sobre un mismo dominio da lugar a la concepción de diferentes sistemas sobre el mismo.

Ejemplo: un meteorólogo y un organizador de defensa civil tienen diferentes problemas que resolver con respecto a un huracán y conciben por la tanto diferentes sistemas sobre tal fenómeno natural:

- el primero observando sus propiedades y comportamiento y tratando de establecer modelos de su trayectoria futura,
- el segundo determinando qué modelos de contingencia y protección aplicar en qué zonas del posible paso del huracán.

Los sistemas creados para solucionar un problema deben ser observables y/o controlables y/o modificables, o sea dado un sistema creado para resolver un problema se desea **Observarlo** y/o **Controlarlo** y/o **Modificarlo**.

Observarlo: Para conocer mejor su estructura y sus procesos. Objetivo: Predecir su comportamiento.

Controlarlo: Asociarle un subsistema que mantenga sus procesos bajo parámetros deseables. Objetivo: Controlar su comportamiento.

Modificarlo: Cambiar su estructura y sus procesos buscando optimizarlas en algún sentido. Objetivo: hacer más económico y eficiente su comportamiento. Las dimensiones mencionadas se implican e interactúan en el proceso de investigación.

Los problemas de modificación y control son los problemas que esencialmente aborda la simulación de sistemas De acuerdo con la definición inicialmente dada de simulación. Se simula un sistema en la medida en que además de observarlo, podemos modificarlo y controlarlo según los objetivos descritos.

Un modelo se utiliza para poder observar, controlar y modificar un sistema. El sistema debe tener una representación mental y/o física para un investigador. Lo único posible es representar el sistema bajo investigación con otro sistema (mejor) conocido: Un modelo es un sistema que representa a otro sistema. El proceso que lleva a la creación del modelo de un sistema lo denominaremos modelación.

Dada la indispensabilidad de la representabilidad de un sistema desde el inicio de su creación comienza el proceso de modelación, luego la selección inicial por abstracción de entidades, relaciones y procesos que se consideran inicialmente relevantes para resolver el problema bajo investigación constituye el primer modelo de un sistema al cual denominaremos sistema real.

La complejidad de la mayoría de los sistemas reales impide que este modelo sea siempre observable, mucho menos y a veces imposible controlable y modificable.

La construcción de un modelo siempre incluye la construcción de lo que denominaremos especificación o modelo conceptual de un sistema: El modelo conceptual es el modelo con el cual hacemos nuestra primera aproximación a un sistema real. El modelo conceptual es una descripción (especificación) de la estructura y los procesos del sistema real, orientada por nuestros propósitos de investigación.

La descripción se realiza en lenguaje natural o en algún lenguaje científico, luego se trata ya de utilizar un sistema simbólico para modelar un sistema real. Siempre es objeto de reformulaciones (refinamiento).

Tipos de modelos:

- Sistema real (modelo conceptual, inicial).

- A escala (transformación métrica).

- Analógico (homologías con otros sistemas).
- Matemático (sistemas de ecuaciones, etc.).
- Computacional (modelos computacionales).

Tipo de Modelo	Realización
Sistema Real	Cohete en Vuelo
A escala	Proyectil en vuelo
Analógico	Pelota en vuelo
Matemático	Ecuaciones diferenciales
Computacional	Programa

1.2.1. Modelación Computacional

La representación (el modelo) abstracta (o) más manipulable de un sistema es la que ofrece un programa. Todos los otros tipos de modelos tienen una realización y por ende una manipulación efectiva mediante un programa. Así, un modelo matemático de un sistema encuentra su manipulación más adecuada tanto cuantitativa como analítica en un programa de computadora.

Se extiende su manipulación cuando el programa en ejecución despliega una dinámica más próxima a la del sistema bajo estudio. Se pueden realizar cambios en determinados parámetros que alteran la dinámica del sistema. Se puede tener una visualización de la estructura, la función y la trayectoria del sistema, aún de sistemas abstractos y virtuales.

1.3. Simulación

El proceso de simulación puede verse como la ejecución, observación y cambio de un modelo determinado.

La simulación computacional es la basada en modelos computacionales, es decir es la realizada en la computadora.

La computadora brinda la posibilidad de representar diversos entornos, por ejemplo:

- Entorno físico adecuado para el lanzamiento de un cohete.
- Polígono militar.

- Terreno de Football.

La versatilidad de la computadora permite que esta pueda comportarse como cualquier máquina existente e incluso como cualquier sistema existente. Mientras más complejo es el sistema, menos tratable se convierte para la computadora, es por esto que es necesario utilizar modelos que simplifiquen el sistema y que solo tome del mismo las entidades, relaciones, etc, que realmente son relevantes para su análisis.

La simulación computacional permite observar el comportamiento de sistemas que en la realidad son extremadamente costosos, por ejemplo el vuelo de un avión, el desarrollo de una batalla, incluso el estudio de proteínas. Además permiten el estudio de la interacción humana con ciertos sistemas cuya simulación no computacional podría costarle la vida a la persona involucrada.

Capítulo 2: Generación de Variables Aleatorias

2.1. Algoritmo de la transformada inversa

Considérese una variable aleatoria continua con distribución F. Un método general para la generación de tal variable (llamado el método de la transformada inversa), está basado en la siguiente proposición.

Proposición 2.1.1.
Sea U una variable aleatoria que distribuye uniforme (0, 1). Para cualquier función de distribución continua F, la variable aleatoria X definida como:

$$X = F^{-1}(U)$$

tiene distribución F.

Demostración. Sea F_x la función de distribución de $X = F^{-1}(U)$. Entonces

$F_x(x) = P\{X \le x\}$
$= P\{F^{-1}(U) \le x\}$ sustituyendo X
$= P\{F(F^{-1}(U)) \le F(x)\}$ por ser F una función monótona creciente
$= P\{U \le F(x)\}$
$= F(x)$ por ser U uniforme (0,1)

□

2.2. Distribución Uniforme (a, b)

Las variables aleatoria con distribución uniforme (a, b) son utilizadas para simular errores aleatorios.

La función de distribución sería:

$$F(x) = \begin{cases} \dfrac{1}{b-a} \rightarrow a \le x \le b \\ 0 \rightarrow eoc. \end{cases}$$

si se aplica la transformada inversa puede verse que si X = a + (b − a)*U entonces X distribuye uniforme (a, b).

2.3. Distribución exponencial con frecuencia λ

Las variables aleatorias con distribución exponencial son utilizadas para simular, por ejemplo, el tiempo que transcurre entre el arribo de una persona y la siguiente a una tienda, también se puede utilizar para simular tiempo de atención.

Sea $F(x) = \lambda e^{-\lambda x}$ la función de distribución de una variable aleatoria que distribuye exponencial con frecuencia λ (con esperanza 1/λ), entonces aplicando la transformada inversa X = -(1/λ)log(U).

2.4. Distribución Poisson con parámetro λ

Supóngase que se tiene un evento E el cual tiene una probabilidad p de ocurrir y una probabilidad q = 1 − p de no ocurrir y que se realizan n experimentos para ver si E ocurre o no. Entonces la variable aleatoria que denota la cantidad de veces que ocurre E se dice que distribuye Binomial ($F(x) = C_x^n p^x q^{n-x}$). La distribución Poisson surge a partir de la aplicación de la ley de los grandes números a la distribución binomial, obteniéndose la función de distribución $P(X \le x) = \frac{e^{-\alpha}\alpha^x}{x!}$, en la cual α es a esperanza o sea la cantidad de veces que se quiere que ocurre E.

Como puede verse una variable aleatoria que distribuya Poisson es discreta, y el método de la transformada inversa se aplica sólo a variables aleatoria continuas.

Se le llamará tiempo entre eventos (que ocurren) sucesivos a la cantidad de experimentos fallidos que se realizan entre un evento y su sucesor.

Un proceso Poisson de parámetro α ocurre cuando los tiempos entre eventos sucesivos independientes distribuyen exponencial con frecuencia λ = α/n, donde a n también se le conoce como la cantidad total de unidades de tiempo.

Si se asume que solo se tiene una unidad de tiempo (n = 1) entonces λ = α y si se toma a X_i como el tiempo que transcurre entre la ocurrencia del evento i − 1 y la ocurrencia del evento i, entonces el n-ésimo evento ocurrirá en el tiempo $\sum_{i=1}^{n} X_i$ y por lo tanto la

cantidad de eventos que ocurren en una unidad de tiempo puede ser expresado como:

$$N = \max\left\{ n : \sum_{i=1}^{n} X_i \le 1 \right\}.$$

Como cada X_i es una variable aleatoria de distribuye exponencial con frecuencia λ, entonces de la siguiente forma se puede generar una variable aleatoria que distribuya Poisson con parámetro λ:

$$\begin{aligned} N &= \max\left\{ n : \sum_{i=1}^{n} -\frac{1}{\lambda}\log(U_i) \le 1 \right\} \\ &= \max\left\{ n : \sum_{i=1}^{n} \log(U_i) \ge -\lambda \right\} \\ &= \max\left\{ n : \log(U_1 * \ldots * U_n) \ge -\lambda \right\} \\ &= \max\left\{ n : (U_1 * \ldots * U_n) \ge e^{-\lambda} \right\} \end{aligned}$$

Por lo tanto una variable aleatoria Poisson N con parámetro λ puede ser generada, a partir de la generación sucesiva de variables uniformes (0, 1) hasta que su producto sea menor que $e^{-\lambda}$ y haciendo a N la cantidad de variables uniformes generadas menos 1. Es decir,

$N = \min\left\{ n : U_1 * \ldots * U_n < e^{-\lambda} \right\} - 1$.

2.4.1. Generación de un Proceso Poisson

Supóngase que se quieren generar las primeras n unidades de tiempo de un Proceso Poisson con parámetro λ*n. Para hacer esto se necesita volver a utilizar el hecho de que los tiempos entre eventos sucesivos distribuyen exponencial con frecuencia λ. Por lo tanto una forma de generar el proceso es generar sus tiempos intermedios. Si se generan n variable aleatorias uniformes $U_1, \ldots, U_n$ y $X_i = -(1/\lambda)\log(U_i)$, entonces X_i es el tiempo que transcurre entre el evento i − 1 y el evento i del proceso Poisson.

Si se desean generar las primeras T unidades de tiempo se puede realizar los siguientes pasos:

PASO 1: t = 0, I = 0.
PASO 2: Genera la variable aleatoria uniforme U.
PASO 3: t = t −(1/λ)log(U).
PASO 4: Si t > T entonces ir al PASO 7.
PASO 5: I = I + 1, S(I) = t.
PASO 6: Ir al PASO 2.
PASO 7: fin.

El valor final de I en el algoritmo anterior representa el número de eventos que ocurrieron en T unidades de tiempo y los valores S(1) . . . S(I) son los tiempos de ocurrencia de cada eventos en orden creciente.

2.4.2. Generación de un Proceso Poisson no homogéneo

Una variación muy importante de los Procesos Poisson es el Proceso de Poisson no homogéneo, el cual relaja la suposición de los procesos Poisson de que los tiempos intereventos distribuyen con un mismo valor de frecuencia (λ).

Usualmente es muy difícil obtener resultados analíticos para un modelo matemático que tenga un Proceso de Poisson no homogéneo y como resulta estos procesos no son tan aplicados como se debiera.

Supóngase que se desean simular las primeras T unidades de tiempo de un Proceso Poisson no homogéneo con función de intensidad $\lambda(t)$ (véase como λ es una función que depende del tiempo).

Si se toma un valor λ igual al máximo de λ (t) ($\lambda(t) \leq \lambda$, $\forall(t \leq T)$), entonces se puede reutilizar el algoritmo propuesto en la sección anterior.

Para poder reutilizar este algoritmo se le debe hacer un variación la cual se basa en lo siguiente: Si un evento de un Proceso Poisson con parámetro λ/n que ocurre en el tiempo t es contado con probabilidad $\lambda(t)/\lambda$, entonces el proceso de contar estos eventos es un Proceso Poisson no homogéneo con función de intensidad $\lambda(t)$. Entonces el algoritmo quedaría como sigue:

PASO 1: t = 0, I = 0.
PASO 2: Genera la variable aleatoria uniforme U.
PASO 3: $t = t - (1/\lambda)\log(U)$.
PASO 4: Si t > T entonces ir al PASO 8.
PASO 5: Genera la variable aleatoria uniforme U'.
PASO 6: Si $U' \leq \lambda(t)/\lambda$ entonces I = I + 1, S(I) = t.
PASO 7: Ir al PASO 2.
PASO 8: fin.

2.5. Ejercicios

Implementar todos los métodos descritos durante este capítulo.

Capítulo 3: Simulación basada en eventos discretos

La simulación de un modelo probabilístico requiere generar los mecanismos estocásticos del modelo y entonces observar el comportamiento del modelo en el tiempo.

Dependiendo de las razones para la simulación, existen algunas cantidades de interés que se querrán determinar. Sin embargo, dado que la evolución del modelo en el tiempo requiere de estructuras lógicas complejas de sus elementos, no es siempre sencillo cómo observar esta evolución para determinar las cantidades de interés. Un marco general, construido a partir de la idea de eventos discretos, se ha desarrollado para ayudar a observar un modelo durante el tiempo y determinar las cantidades de interés. La simulación basada en este marco se le denomina simulación basada en eventos discretos.

3.1. Eventos discretos

Los elementos principales de la simulación basada en eventos discretos son las variables y los eventos. Para hacer la simulación continuamente se observan ciertas variables. En general hay tres tipos de variables que son usualmente utilizadas, *la variable de tiempo, las variables contadoras y las variables de estado del sistema*.

Variables:

1. Variable de tiempo t Se refiere a la cantidad de tiempo de simulación que ha transcurrido.

2. Variable contadoras Estas variables cuentan la cantidad de veces que un evento ha ocurrido hasta el tiempo t.

3. Variables de estado del sistema (SS) Estas describen el estado del sistema hasta el tiempo t.

Cuando un evento ocurre los valores de las variables anteriores cambian o se actualizan y a partir de ellas se colecta cualquier dato de salida. Para determinar cuando el próximo

evento ocurrirá se mantiene una lista de eventos en la cual se almacena el evento que ocurrirá más próximo en el futuro.

3.2. Un sistema de atención de un solo servidor

Considere una estación de servicio en la cual los clientes arriban en correspondencia con un Proceso Poisson no homogéneo. Solamente hay un servidor y una vez que llega un cliente es atendido por el servidor, si está vacío sino espera su turno en una cola. Cuando el servidor termina de atender un cliente comienza inmediatamente a atender al cliente que lleva más tiempo esperando en la cola. El tiempo que demora el servidor en atender un cliente es una variable aleatoria que distribuye G.

Existe un tiempo fijo T a partir del cual no se le permite entrar a más ningún cliente al sistema, sin embargo todos los clientes que ya estaban dentro deberán ser atendidos.

Supóngase que se quiere simular el sistema anterior para determinar (a) la cantidad promedio de tiempo que un cliente pasa en el sistema y (b) el tiempo promedio pasado T en el que el último cliente deja el sistema.

Para realizar la simulación anterior se tomarán las variables siguientes:

1. Variable de tiempo t
2. Variables contadoras
    a. N_A: la cantidad de arribos (en el tiempo t)
    b. N_D: la cantidad de partidas (en el tiempo t)
3. Variables de estado del sistema n: el número de clientes que está en el sistema.

Entonces el momento natural en el cual se deben variar las variables anteriores es cuando ocurre o una llegada o una partida de un cliente, tómese entonces a estos como los eventos. Hay dos tipos de eventos de llegada y de partida. La lista de eventos contiene, entonces, el momento de la próxima llegada y el momento de la próxima partida EL = t_A, t_D.

Las variable de salida serán A(i) tiempo de llegada del cliente i-ésimo, D(i) tiempo de partida del cliente i-ésimo y T_p el tiempo pasado T en el que el último cliente deja el sistema.

Inicialización

$t = N_A = N_D = 0$.
SS = 0.
Generar T_0 y hacer $t_A = T_0$, $t_D = 1$.

Caso 1 $t_A <= t_D \wedge t_A <= T$

$t = t_A$ (se mueve el tiempo de la simulación a t_A).

$N_A = N_A + 1$ (llega un nuevo cliente).
$n = n + 1$ (hay un cliente más en el sistema).
Generar T_t y hacer $tA = t + Tt$ (tiempo del próximo arribo).
Si $n = 1$ entonces generar Y hacer $t_D = t + Y$ (tiempo de la próxima partida).
$A(N_A) = t$.

Caso 2 $t_D < t_A \wedge t_D <= T$
$t = t_D$.
$n = n - 1$.
$N_D = N_D + 1$.
Si $n = 0$ entonces $t_D = 1$ sino generar Y y hacer $t_D = t + Y$.
$D(N_D) = t$.

Caso 3 $\min(t_A, t_D) > T \wedge n > 0$
$t = t_D$.
$n = n - 1$.
$N_D = N_D + 1$.
Si $n > 0$ entonces generar Y y hacer $t_D = t + Y$.
$D(N_D) = t$.

Caso 4 $\min(t_A, t_D) > T \wedge n = 0$
$T_p = \max(t - T, 0)$.

3.3. Un sistema de atención con dos servidores en serie

Considere un sistema igual al anterior solo que hay dos servidores en serie, es decir, cuando un cliente termina en el servidor 1 pasa al servidor 2 (o a su cola si está ocupado).

Variable de Tiempo
t.

Variables contadoras
N_A y N_D.

Variables de estado del sistema
(n_1, n_2): n_1 cantidad de clientes en el servidor 1 y n_2 cantidad de clientes en el servidor 2.

Variables de salida
$A_1(n)$: tiempo de arribo del cliente n.
$A_2(n)$: tiempo de arribo del cliente n al servidor 2.
D(n): tiempo de partida del cliente n.

Lista de eventos

t_A tiempo de la próxima llegada.
t_1 tiempo de la próxima partida del servidor 1.
t_2 tiempo de la próxima partida del servidor 2.

Inicialización

$t = N_A = N_D = 0$.
$SS = (0, 0)$.
Genera T_0, $t_A = T_0$ y $t_1 = t_2 = \infty$.

Caso 1 $t_A = \min(t_A, t_1, t_2)$ ^ $t_A < T$

$t = t_A$.
$N_A = N_A + 1$.
$n = n + 1$.
Genera T_t y $t_A = t + T_t$.
Si $n_1 = 1$ entonces Genera Y_1 y $t_1 = t + Y_1$.
$A_1(N_A) = t$.

Caso 2 $t_1 < t_A$ ^ $t_1 <= t_2$ ^ $t_1 < T$

$t = t_1$.
$n_1 = n_1 - 1$.
$n_2 = n_2 + 1$.
Si $n_1 = 0$ entonces $t_1 = 1$ sino genera Y_1 y $t_1 = t + Y_1$.
Si $n_2 = 1$ entonces genera Y_2 y $t_2 = t + Y_2$.
$A_2(N_A - n_1) = t$.

Caso 3 $t_2 < t_A$ ^ $t_2 < t_1$ ^ $t_2 < T$

$t = t_2$.
$N_D = N_D + 1$.
$n_2 = n_2 - 1$.
Si $n_2 = 0$ entonces $t_2 = 1$ sino genera Y_2 y $t_2 = t + Y_2$.
$D(N_D) = t$.

3.4. Un sistema de atención con dos servidores en paralelo

Similar al sistema anterior solo que los servidores están en paralelo, es decir que cuando llega un cliente este es atendido por el servidor que se encuentre vacío y si los dos están llenos entonces espera en la cola.

Variable de Tiempo

t.

Variables contadoras

N_A: cantidad de llegadas.
C_j : cantidad clientes atendidos por el servidor j, con j = 1, 2.

Variables de estado del sistema

$(n, i_1, i_2, \ldots, in)$ si hay n clientes en el sistema, i_1 está en el servidor 1, i_2 está en el servidor 2, i_3 es el primero de la cola, i_4 el próximo y así sucesivamente.

Nótese que SS = (0) cuando el sistema está vacío y SS(1, j, 0) o SS(1, 0, j) cuando hay un solo cliente, j, y está siendo atendido por el servidor 1 o el servidor 2 respectivamente.

Variables de salida

A(n): tiempo de arribo del cliente n.
D(n): tiempo de partida del cliente n.

Lista de eventos

(t_A, t_1, t_2): tal que t_A es el tiempo del próximo arribo, ti es el tiempo de partida del cliente que está siendo atendido en el servidor i, con i = 1, 2. Si no hay cliente en el servidor i entonces ti = 1.

Inicialización

$t = N_A = C_1 = C_2 = 0$.
SS = (0).
Genera T_0 y $t_A = T_0$, $t_1 = t_2 = \infty$.

Caso 1 $SS = (n, i_1, i_2, \ldots, i_n)$ ^ $t_A = \min(t_A, t_1, t_2)$

$t = t_A$.
$N_A = N_A + 1$.
Genera T_t y $t_A = t + T_t$.
$A(N_A) = t$.
Si SS = (0) entonces $SS = (1, N_A, 0)$, genera Y_1 y $t_1 = t + Y_1$.
Si SS = (1, j, 0) entonces $SS = (2, j, N_A)$, genera Y_2 y $t_2 = t + Y_2$.
Si SS = (1, 0, j) entonces $SS = (2, N_A, j)$, genera Y_1 y $t_1 = t + Y_1$.
Si n > 1 entonces $SS = (n + 1, i_1, i_2, \ldots, i_n, N_A)$.

Caso 2 $SS = (n, i_1, i_2, \ldots, i_n)$ ^ $t_1 < t_A$ ^ $t_1 <= t_2$

$t = t_1$.
$C_1 = C_1 + 1$.
$D(i_1) = t$.
Si n = 1 entonces SS = (0) y $t_1 = 1$.
Si n = 2 entonces $SS = (1, 0, i_2)$ y $t_1 = 1$.
Si n > 2 entonces $SS = (n - 1, i_3, i_2, \ldots, i_n)$, genera Y_1 y $t_1 = t + Y_1$.

Caso 3 $SS = (n, i_1, i_2, \ldots, i_n)$ ^ $t_2 < t_A$ ^ $t_2 < t_1$

Este caso se deja como ejercicio.

3.5. Un modelo de inventario

Un almacén guarda un tipo de producto y lo vende a un precio de r por unidad. Los clientes llegan al almacén con una distribución Poisson y la cantidad demandada por cliente distribuye G. Para mantener el inventario se tiene (s, S) tal que si $x < s$ (x nivel de inventario) entonces se ordena una cantidad $S - x$ de unidades de producto ($s < S$). El costo de ordenar y unidades es c(y) y toma L unidades de tiempo en satisfacerse una orden. Hay un costo de almacenamiento de h por unidad de producto, por unidad de tiempo.

Variable de Tiempo

t.

Variables contadoras

C: costo total de las órdenes.
H: costo total de almacenaje.
R: ganancia.

Variables de estado del sistema

(x, y): donde x es la cantidad de inventario, y es la cantidad solicitada.

Variables de salida

P perdida por solicitudes de clientes que no pudieron ser satisfechas por falta de inventario.

Lista de eventos

(t_0, t_1): donde t_0 es el tiempo de arribo del próximo cliente y t_1 tiempo de satisfacción de la próxima orden de compra, si no hay ninguna orden de compra entonces $t_1 = \infty$.

Inicialización

$t = C = H = R = 0$.
x = cantidad inicial en almacén.
$y = 0$.
Genera T_0 y $t_0 = T_0$.
$t_1 = 1$.

Caso 1 $t_0 < t_1 \wedge t_0 < T$

$H = H + (t_0 - t)*x*h$.
$t = t_0$.
Genera Y con distribución G y $w = \min(x, D)$.
$R = R + w*r$.
$P = P + (D - w)*r$.
$x = x - w$.
Si $x < s \wedge y = 0$ entonces $y = S - x$ y $t_1 = t + L$.
Genera T_t y $t_0 = T_t$.

Caso 2 $t_1 <= t_0 \wedge t_1 < T$

$H = H + (t_1 - t)*x*h.$

$t = t_1.$

$C = C + c(y).$

$x = x + y.$

$y = 0.$

$t_1 = 1.$

La ganancia por unidad de tiempo puede calcularse como $(R-C-H)/T$.

3.6. Ejercicios

1. Completar el ejemplo de la sección 3.4.

2. Generalizar para k servidores en serie.

3. Generalizar para k servidores en paralelo.

4. Generalizar el modelo de inventarios para k productos.

Capítulo 4: Análisis estadístico de la Simulación

Una simulación es usualmente utilizada para determinar el valor de alguna cantidad θ conectada a un modelo estocástico específico. La simulación del sistema resulta en el dato de salida X, una variable aleatoria cuyo valor esperado es θ. Una segunda simulación independiente a la anterior brinda una nueva variable aleatoria independiente con esperanza θ. Este proceso continúa hasta que se tengan k corridas y k variables independientes $X_1, \ldots, X_k$ las cuales están idénticamente distribuidas con esperanza θ. El promedio de estos k valores, $\overline{X} = \sum_{i=1}^{k} \frac{X_i}{k}$, es entonces utilizado como estimador de θ.

En este capítulo se considera el problema de decidir cuándo se detiene la simulación, es decir decidir el valor apropiado de k. Para ayudar en esta decisión, se considerará apropiado tener una buen estimador de θ.

4.1. La esperanza y la varianza de la muestra

Supóngase que $X_1, \ldots, X_n$ son variables aleatorias independientes con la misma función de distribución. Sean θ y σ^2 la esperanza y la varianza, tal que $\theta = E[X_i]$ y $\sigma^2 = Var(X_i)$. El valor, $\overline{X} = \sum_{i=1}^{n} \frac{X_i}{n}$, el cual es el promedio de los n valores de datos, es llamado *esperanza de la muestra*.

Cuando la esperanza de la población θ es desconocida, la esperanza de la muestra es usualmente utilizada para estimarla, ya que,

$$\begin{aligned} E\left[\overline{X}\right] &= E\left[\sum_{i=1}^{n} \frac{X_i}{n}\right] \\ &= \sum_{i=1}^{n} \frac{E[X_i]}{n} \\ &= \frac{n\theta}{n} = \theta \end{aligned}$$

esto demuestra que X es un estimador insesgado de θ. Se dice que un estimador de un parámetro es insesgado si su valor esperado es igual al del parámetro.

Para determinar la calidad de X como estimador de la esperanza de la población θ, se considera la esperanza de su error cuadrático, es decir el valor esperado del cuadrado de la diferencia entre X y θ. Ahora,

$$\begin{aligned} E\left[\left(\overline{X}-\theta\right)^2\right] &= Var\left(\overline{X}\right) && (\textit{dado que } E\left[\overline{X}=\theta\right]) \\ &= Var\left(\frac{1}{n}\sum_{i=1}^{n}X_i\right) \\ &= \frac{1}{n^2}\sum_{i=1}^{n}Var\left(X_i\right) && (\textit{por independencia}) \\ &= \frac{\sigma^2}{n} && (\textit{dado que } Var\left(X_i\right)=\sigma^2) \end{aligned}$$

Entonces, $\overline{X}$ es una variable aleatoria con esperanza θ y varianza σ^2. Se dice que $\overline{X}$ es un buen estimador de θ en la medida que $\sigma/\sqrt{n}$ es pequeño.

La dificultad de utilizar directamente σ^2/n como un indicador de cuan bien la esperanza de la muestra estima la esperanza de la población, es que normalmente σ^2 no se conoce. Entonces también hay que estimarlo. Como,

$$\sigma^2 = E\left[\left(X-\theta\right)^2\right]$$

es el promedio del cuadrado de la diferencia entre los valores de los datos y su desconocida esperanza, puede parecer que como $\overline{X}$ se utiliza como estimador de θ, el estimador natural de σ^2 es $\sum_{i=1}^{n}\frac{\left(X_i-\overline{X}\right)^2}{n}$; pero para hacer el estimador insesgado hay que dividir por n − 1 en vez de por n.

Definición 4.1.1. Se define como varianza de la muestra a $S^2=\frac{\sum_{i=1}^{n}\left(X_i-\overline{X}\right)^2}{n-1}$.

Utilizando la identidad algebraica

$$\sum_{i=1}^{n}\left(X_i-\overline{X}\right)^2=\sum_{i=1}^{n}X_i^2-n\overline{X}^2 \qquad (4.3)$$

se puede demostrar la siguiente proposición.

Proposición 4.1.2. $E\left[S^2\right]=\sigma^2$.

Demostración. Utilizando la identidad (4.3) se puede ver que

$$\begin{aligned}(n-1)E\left[S^2\right] &= E\left[\sum_{i=1}^{n} X_i^2\right] - nE\left[\overline{X}^2\right] \\ &= nE\left[X_i^2\right] - nE\left[\overline{X}^2\right] \qquad (4.5)\end{aligned}$$

donde la última igualdad se cumple ya que X_i tienen la misma distribución. Retomando que para cualquier variable aleatoria Y , Var(Y) = E[Y^2] − (E[Y]^2) o equivalentemente E[Y^2] = Var(Y) + (E[Y])^2, se obtiene que,

$$\begin{aligned}E\left[X_i^2\right] &= Var\left(X_i\right) + \left(E\left[X_i\right]\right)^2 \\ &= \sigma^2 + \theta^2\end{aligned}$$

y

$$\begin{aligned}E\left[\overline{X}^2\right] &= Var\left(\overline{X}\right) + \left(E\left[\overline{X}\right]\right)^2 \\ &= \frac{\sigma^2}{n} + \theta^2 \qquad \text{por (4.1) y (4.2)}\end{aligned}$$

Por lo tanto por (4.5) se obtiene que,

$$(n-1)E\left[S^2\right] = n\left(\sigma^2 + \theta^2\right) - n\left(\frac{\sigma^2}{n} + \theta^2\right) = (n-1)\theta^2$$

lo cual prueba la proposición.

□

Se utiliza entonces S^2 como estimador de la varianza σ^2 y $S = \sqrt{S^2}$, denominada la desviación estándar de la muestra como estimador de θ.

Supóngase ahora que se tiene la opción de continuamente generar valores X_i ¿Si el objetivo es estimar θ, cuándo se debe parar de generar nuevos valores? La respuesta a esta pregunta es que primero se debe seleccionar un valor adecuado de la desviación estándar del estimador, d es la desviación estándar del estimador X. Se debe continuar la generación de nuevos datos, hasta que $\frac{S}{\sqrt{n}} < d$. El siguiente algoritmo se utiliza para determinar cuando para la simulación,

1. Seleccionar un valor adecuado para la desviación estándar del estimador d.
2. Generar al menos 30 conjuntos de valores (30 simulaciones).
3. Continuar generando valores , parando cuando se hayan generado k conjuntos de valores y $\frac{S}{\sqrt{k}} < d$, donde S es la desviación estándar de la muestra basada en esos k conjuntos de valores.

4. Estimar θ como $\overline{X} = \sum_{i=1}^{k} \frac{X_i}{k}$.

4.2. Ejercicios

1. Aplicar el análisis estadístico de la simulación a las simulaciones estudiadas en el capítulo anterior.

2. Para estimar θ, se generan 20 valores independientes con esperanza θ. Si los valores obtenidos fueron: 102, 112, 131, 107, 114, 95, 133, 145, 139, 117, 93, 111, 124, 122, 136, 141, 119, 122, 151, 143 ¿ Cuántos valores adicionales se deben generar para obtener un estimador de θ con una desviación estándar de 0.5?.

Capítulo 5: Simulación con sistemas difusos

5.1. Lógica difusa

La palabra ‘‘difusa’’: Difuso (lat. Diffusu Pp. Irreg. De difundir

2 adj. Ancho,dilatado

3 Excesivamente dilatado, superabundante en palabras: es un orador dif. y un escritor conciso.

4 Que es poco concreto, claro o limitado.

Contr. 3 Sobrio, conciso, ceñido

Conjuntos precisos (crisp sets): Conjunto con función de pertenencia o membresía asociada “determinista”: dada una entidad de un dominio la función determina de manera efectiva sí o no dicha entidad pertenece a (es elemento de) de un (sub) conjunto del dominio.

Ejemplo: Conjunto de los números pares: P = 0, 2, 4, . . . (representación extensional de P), función de pertenencia par(X) si X = 2N y N es un número natural, P = {X| X = 2N y N es un número natural} (representación intencional de P).

La función de membresía o pertenencia se denomina clásicamente en matemática función característica. Un conjunto que tiene asociada una función característica se denomina un conjunto bien definido y en lógica un conjunto recursivo.

La función característica es un predicado en una o más variables construido en el lenguaje de alguna teoría basada en una lógica bivalente (true (1) y false (0)).

En la lógica bivalente rigen la ley del tercero excluido y el principio de no-contradicción: un enunciado A es verdadero o falso pero no ambos.

5.1.1. Conjuntos imprecisos o difusos, definición intuitiva

(El conjunto de) los jóvenes.
(El conjunto de) los viejos

Los anteriores conjuntos no tienen asociada una función característica (o al menos no se ha podido definir todavía ninguna). Si embargo los utilizamos como tales para nuestros razonamientos ordinarios: La pertenencia de una persona dada a uno de estos conjuntos es considerada al menos implícitamente como una pertenencia de grado.

El grado de pertenencia se hace mas manifiesto cuando hablamos de "los más viejos", "los menos jóvenes", etc.

La lógica difusa trata de definir formalmente esta noción de grado de pertenencia.
También razonamos con enunciados difusos o imprecisos:

El agua esta caliente.
No me están atendiendo bien.
Rosa no es muy joven.

En todos los casos conjuntos y enunciados difusos no son tales sólo debido a la apreciación subjetiva del que habla (la cual por supuesto está presente y será importante en los desarrollos posteriores) si no a la imposibilidad de encontrar límites precisos, "duros", que demarquen cuando el agua no está caliente, en qué oportunidad decido que no me atienden bien, cuando considero que una persona no es muy joven. En resumen ¿Cómo darnos cuenta o nos ponemos de acuerdo de las transiciones?

Tales enunciados no son ni verdaderos ni falsos de manera absoluta y no se cumple para ellos el principio del tercero excluido, ni el principio de no-contradicción.

La lógica difusa construye teorías formales que permitan la operatoria y el razonamiento con clases (conjuntos) y enunciados vagos o imprecisos.

Como formalismo es una abstracción que trata de captar operaciones y reglas de inferencia lógicas que puedan servir para resolver computacionalmente problemas en los cuales debemos manipular clases y enunciados imprecisos.

De modo general la lógica difusa trata de captar y formalizar la noción de grado de pertenencia de un conjunto difuso y de grado de verdad de una proposición difusa.

5.1.2. Revisión de la lógica proposicional bivalente

Variables proposicionales: p, q, r, ...

Operaciones proposicionales: ¬ (negación), ∨ (disyunción), ^ (conjunción), => (implicación), <=> (bicondicional o equivalencia lógica).

Signos auxiliares: [,].

Definición de Fórmulas:

1. Toda variable proposicional es una fórmula.
2. Si A es una fórmula, entonces ¬A es una fórmula.
3. Si A y B son fórmulas, entonces [A ∨ B], [A ^ B], [A => B], [A <=> B].

Lógica proposicional bivalente (LPB)

En LPB, las variables proposicionales toman los valores de verdad: 1 (verdadero), 0 (falso).

Definición de las operaciones proposicionales:

A	B	¬A	A ∨ B	A ^ B	A => B	A <=> B
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	1	1	1	1	1

Una interpretación de una fórmula A es una función que asigna un valor de verdad a cada una de las variables proposicionales que ocurren en A.

Si A tiene n variables proposicionales distintas, entonces A tiene 2^n interpretaciones.

Una interpretación de una fórmula A determina un valor de verdad para A. El método de las tablas de verdad sirve para calcular los valores de una fórmula correspondientes a sus diversas interpretaciones.

Una fórmula A de la lógica proposicional es

- Cumplible, si es verdadera por alguna interpretación.
- Una tautología, si es verdadera para todas sus interpretaciones.
- Incumplible o una contradicción, si no es cumplible.

5.1.3. Lógicas n-valentes (n <= 2)

Lógica trivalente.

En una lógica trivalente las proposiciones pueden tomar, por ejemplo, los valores (1 (verdadero), 1/2 (indeterminado), 0 (falso).

Las operaciones lógicas se define por funciones de verdad adecuadas:

A	B	¬A	A ∨ B	A ^ B	A => B	A <=> B
0	0	1	0	0	1	1
0	1	1	1	0	1	0

1	0	0	1	0	0	0
1	1	1	1	1	1	1
0	½					
½	0					
0	½					
½	1					
½	½					

La tabla anterior ha sido rellenada sólo con los valores correspondientes a la funciones de verdad de la lógica bivalente. Note que las definiciones extensionales dadas de ¬, ∨ y ^ pueden expresarse intencionalmente de la manera siguiente:

A ∨ B = max(A,B) (valor de verdad máximo entre A y B).
A ^ B = min(A,B) (valor de verdad mínimo entre A y B).
¬A = 1 − A (1 menos el valor de A).

Las anteriores definiciones de ¬, ∨ y ^ no son las únicas posibles.

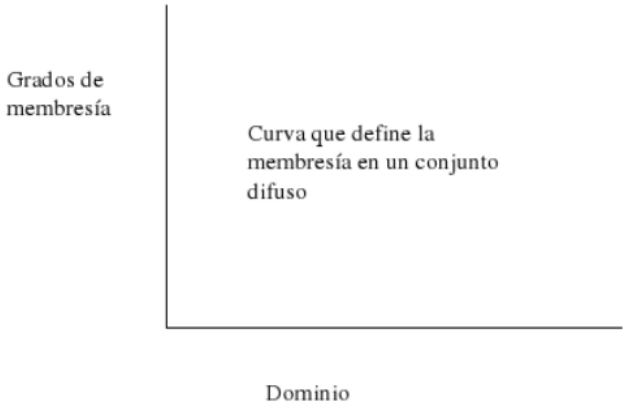
La función de verdad de la implicación puede ser diversamente definida. Esta es la definición más común dada por el lógico polaco Lukasiewicz: $A =>_L B = \min(1, 1 - A + B)$.

5.1.4. Lógicas difusas

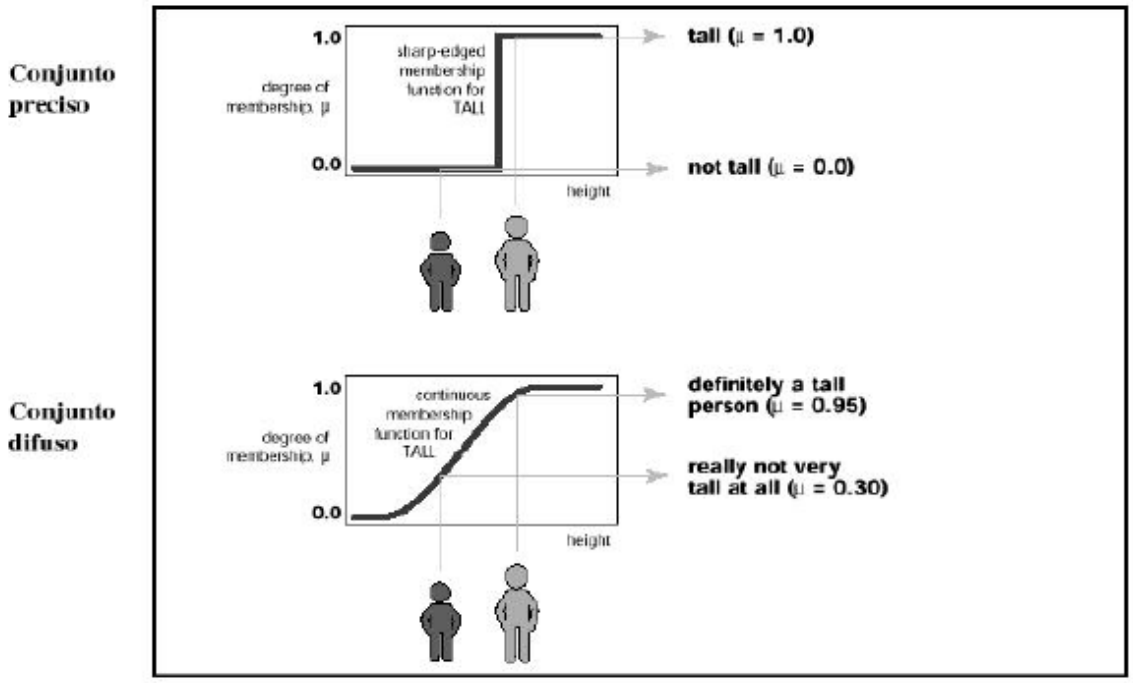
En una lógica difusa las proposiciones toman valores en el intervalo unidad [0, 1] o en uno de sus subintervalos {0, 1}, {0, 1/2, 1}, etc. que satisfagan ciertos requisitos (más adelante). Los valores en el intervalo mencionado constituyen los grados de pertenencia o membresía.

Dado un dominio X, un conjunto difuso A sobre X queda definido por una función de membresía $\mu_A(x)$ que establece para cada elemento de X un grado de pertenencia a A. El conjunto de los pares formados por cada elemento x de X y su grado de pertenencia a A, $\mu_A(x)$, define el conjunto difuso A: $A = \{< x, \mu_A(x) > | x \backslash in X\}$.

La función de membresía puede representarse mediante una curva en el plano.

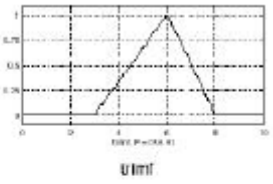


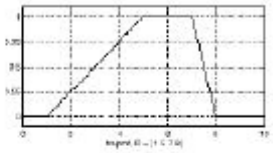
El conjuntos de las personas altas representado como,

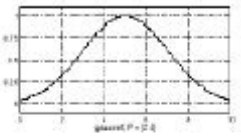


Repertorio de funciones de membresía para conjuntos difusos.

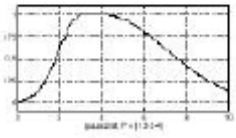
Triangular **Trapezoidal** **Gaussiana**

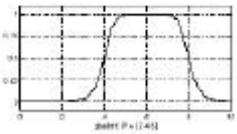






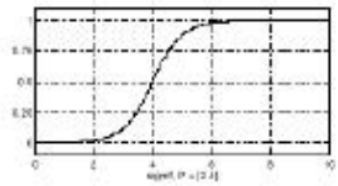
Gaussiana2 **Campana generalizada**

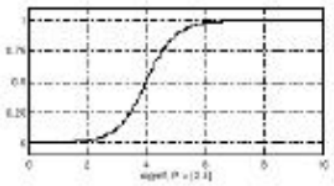




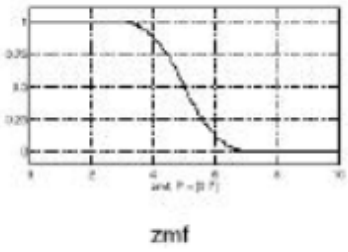
Funciones de membresía asimétricas

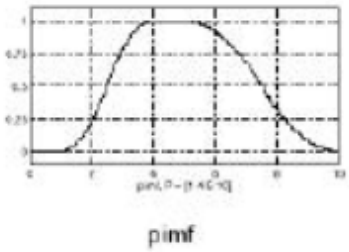
Sigmoidal **Diferencia de sigmoidales**

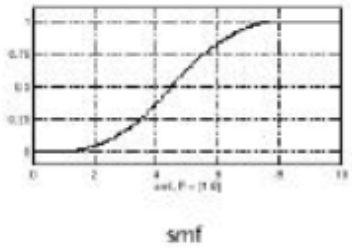




Curvas basadas en polinomios







Variables lingüísticas

El concepto de variable lingüística fue introducido y juega un papel fundamental en la lógica difusa.

- La temperatura se puede medir, por ejemplo, en grados centígrados. Consideremos el conjunto preciso de los valores numéricos de la temperatura en grados centígrados como el dominio o universo de definición de nuestros conjuntos.

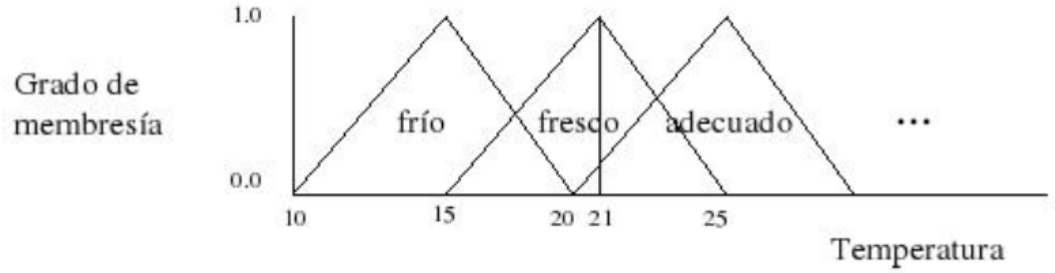
- En un sistema acondicionador de aire la temperatura del ambiente se controla cualificándola por ejemplo con los siguientes valores lingüísticos: frío, fresco, adecuado, cálido, caliente.

Pero frío, fresco, adecuado, cálido y caliente so propiedades difusas que según hemos vistos se define en la lógica difusa como conjuntos difusos.

Luego la temperatura se puede considerar una variable lingüística con un conjunto asociado de valores lingüísticos, siendo cada uno de estos valores un conjunto difuso definido por una correspondiente función de membresía.

De modo que para cada valor cuantitativo de temperatura la función de membresía de cada conjunto difuso determina su grado de pertenencia a dicho conjunto.

Una definición de la temperatura como variable lingüística con tres valores lingüísticos definidos como conjuntos difusos definidos a su vez triangularmente.



Se aprecia que para 21 ◦C $\mu_{fresco}(21) = 1$, $\mu_{frío}(21) = 0$, $\mu_{adecuado}(21) = 0{,}1$ . . . .

El proceso que realiza una función de membresía de proyectar un valor numérico en un valor numérico difuso se denomina *Difusificación* (fuzzyfication).

El proceso inverso de proyectar un número difuso en un número se denomina *Desdifusificación* (defuzzyfication).

5.2. Sistemas difusos

Como se ha estudiado, siendo **R** el conjunto de los números reales, la función de transición de estado ϕ que describe la dinámica de un sistema es $\phi : R_n \rightarrow R_p$ donde R_n

(R_p) es el conjunto de vectores que representan las entradas (salidas) del sistema. La modelación de la dinámica de un sistema consiste en hallar una definición para ϕ.

E. H. Mamdani desarrolló el concepto de sistemas difusos definiendo una aproximación a ϕ basada en su descripción mediante reglas que involucran conjuntos difusos.

Mamdani diseñó un sistema difuso para el control de una máquina de vapor, usando un modelo de sistema difuso que difiere sólo en detalles de los que actualmente se utilizan en la práctica. Su trabajo marca el inicio de la ingeniería difusa.
Los sistemas difusos son descritos por reglas de la forma: Si X es A, entonces Y es B.

Ejemplo: Reglas para el sistema de aire acondicionado como un sistema fuzzy.

R1: Si el aire está frío, entonces velocidad=parada.
R2: Si el aire está fresco, entonces velocidad= lenta.
R3: Si el aire está correcto, entonces velocidad=media.
R4: Si el aire está cálido, entonces velocidad=rápida.
R5: Si el aire está caliente, entonces velocidad=máxima.

Sistemas de Inferencia Difusa (SID)

Un SID determina como aplicar las reglas difusas para transformar las entradas en salidas de un sistema difuso. De acuerdo con el SID seleccionado se tienen diferentes tipos de sistemas difusos.

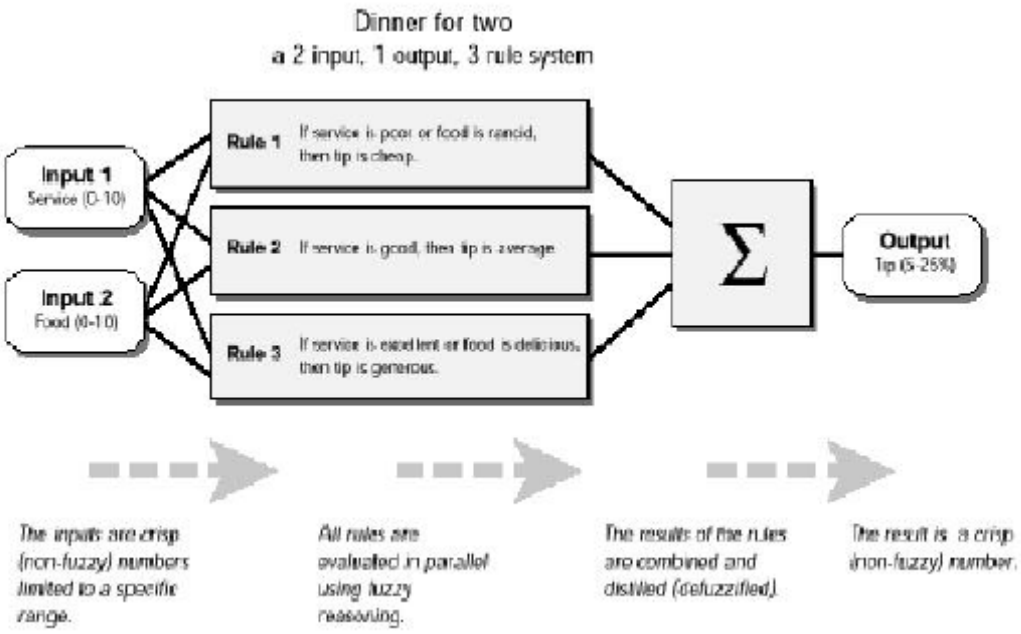
Estudiaremos un tipo particular de sistemas difusos: Sistemas difusos aditivos: modelo aditivo estándar (MAS).

Ejemplo: Sistema de determinación de propina.

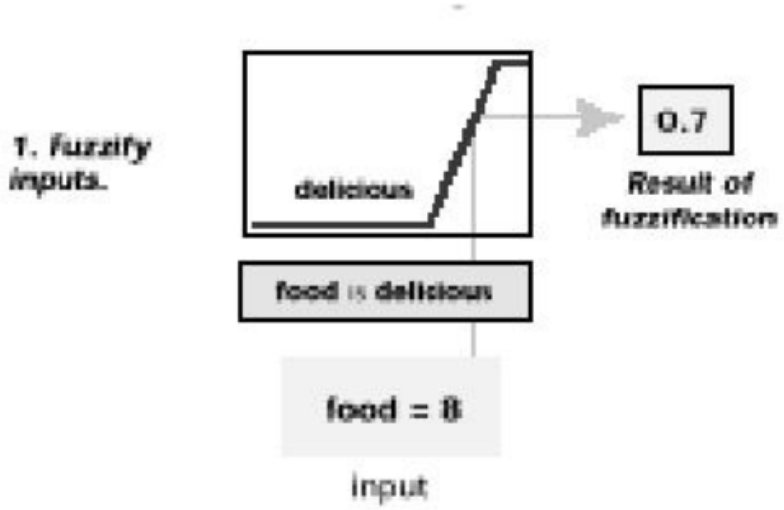
R1: Si el servicio es pobre o la comida es mala, entonces la propina es poca.
R2: Si el servicio es bueno, entonces la propina es promedio.
R3: Si el servicio es excelente o la comida es deliciosa, entonces la propina es generosa.

A continuación utilizando este ejemplo, desarrollaremos los pasos fundamentales de un SID. El ejemplo ha sido tomado del Manual del toolbox sobre lógica difusa del Mathlab, el cual puede consultarse para más información.

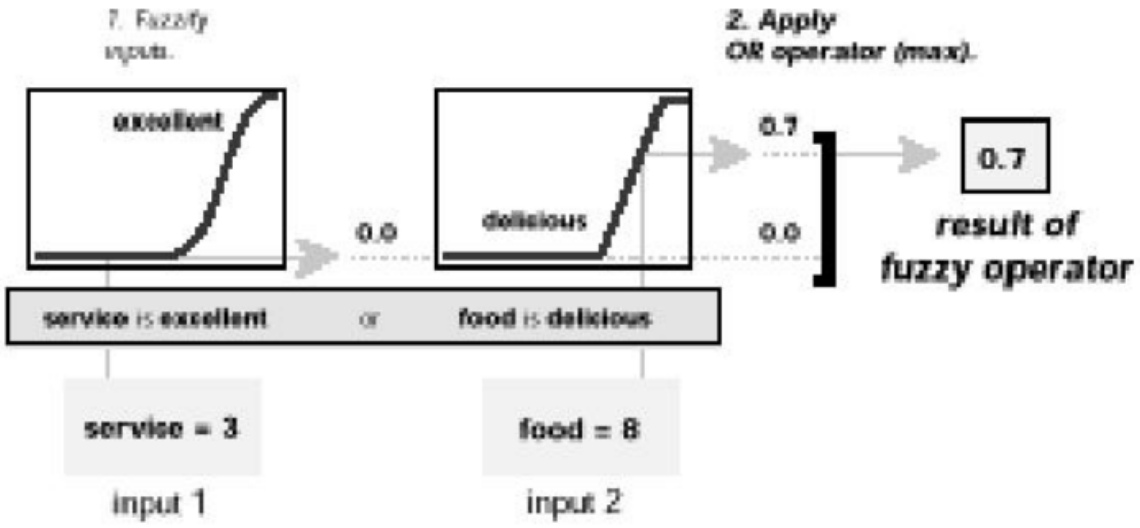
Esquema del sistema difuso para la determinación de la propina.



Paso #1: Difusificación de la entrada: tomar el valor (escalar) de la entrada y transformarlo en el grado de membresía en el conjunto difuso correspondiente.

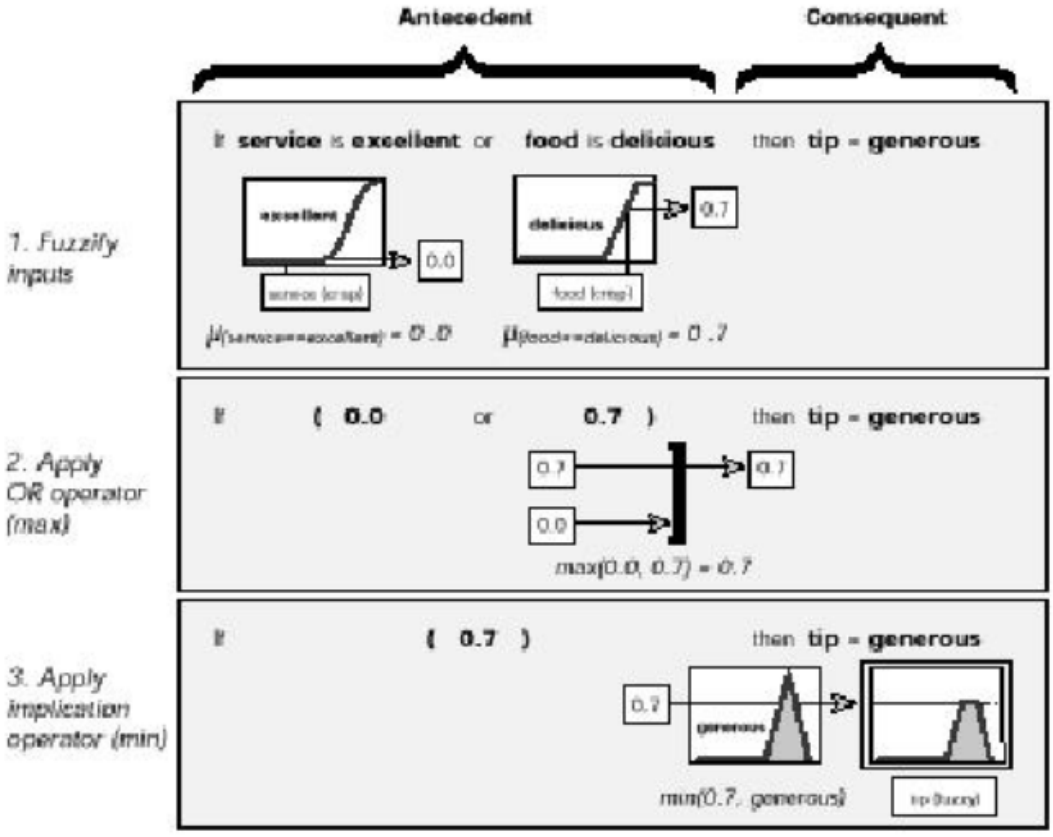


Paso #2: Operaciones lógicas en el antecedente de las reglas: aplicar las operaciones lógicas que ocurren en el antecedente de la regla.

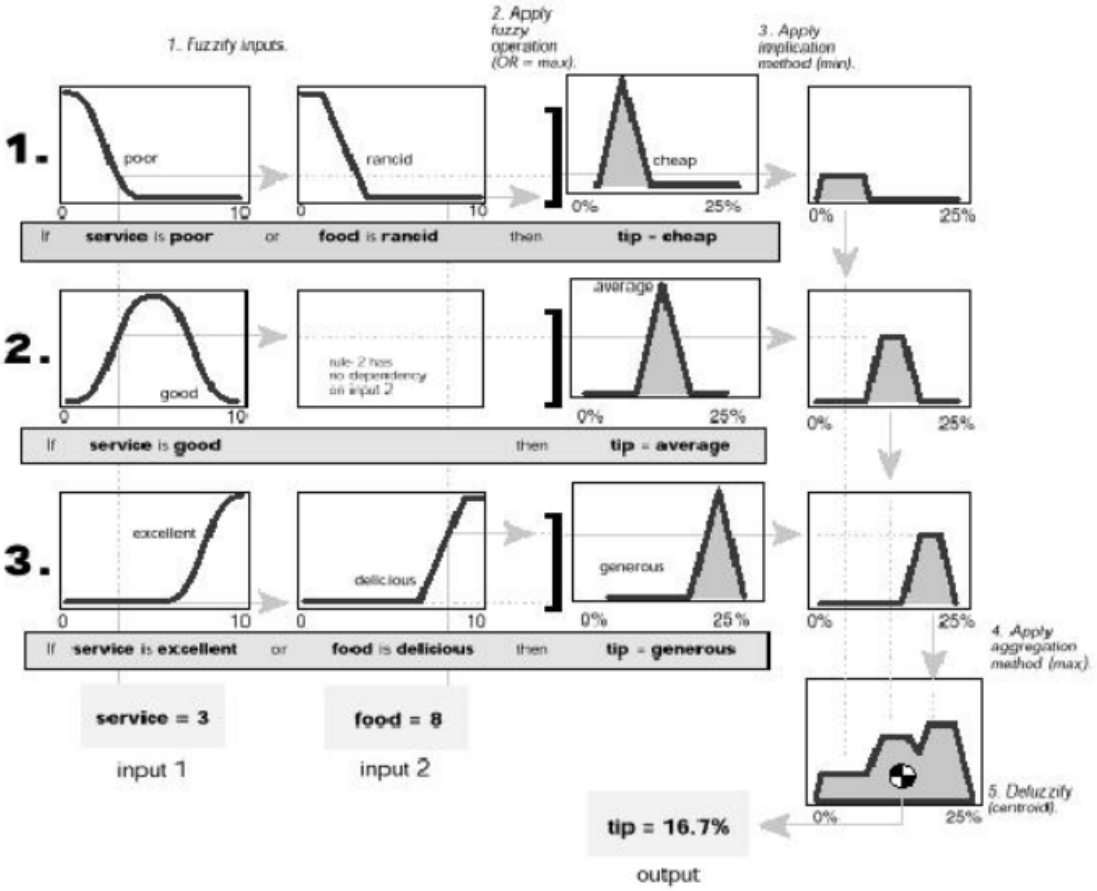


Puede haber otras operaciones y diferentes definiciones de las operaciones.

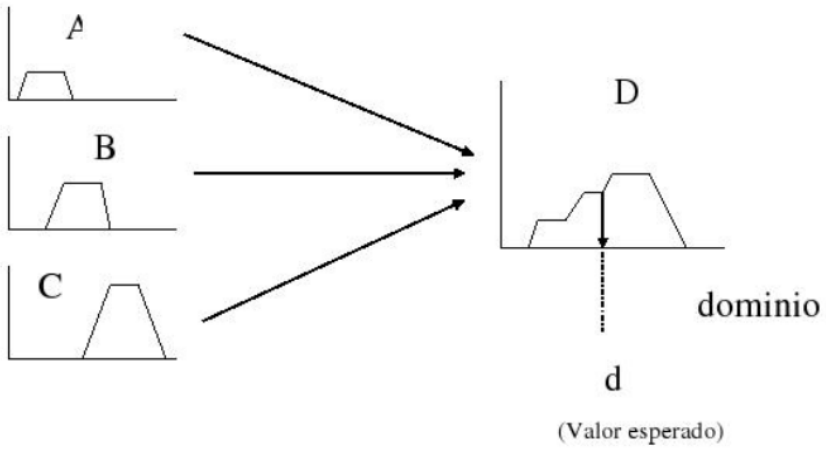
Paso #3: Obtención del consecuente de una regla a partir del antecedente.



Paso #4: Agregación de los consecuentes de todas las reglas.



Paso #5: Desdifusificación (defuzzification). El último paso del proceso inferencial consiste en determinar el valor (escalar) d del dominio más representativo de la región difusa compuesto por los conjuntos difusos de salida resultantes de aplicar las reglas.



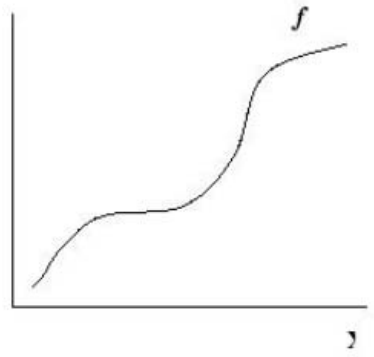
Técnica del centroide o centro de gravedad.

El centroide determina el punto "balance" de la región difusa calculando la media ponderada de dicha región. Sea A la región difusa, entonces $d \leftarrow \frac{\sum_i d_i \mu_A(d_i)}{\sum_i \mu_A(d_i)}$ con i = 0..n.

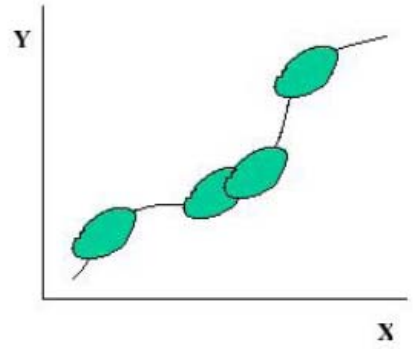
La del centroide es la técnica más difundida.

5.2.1. Aproximación de funciones con sistemas difusos

Un sistema difuso es capaz de aproximar, con cualquier nivel de precisión cualquier función continua, lineal o no lineal (Kosko, B. "Fuzzy System as Universal aproximators" IEEE Transaction on computers, 1933).

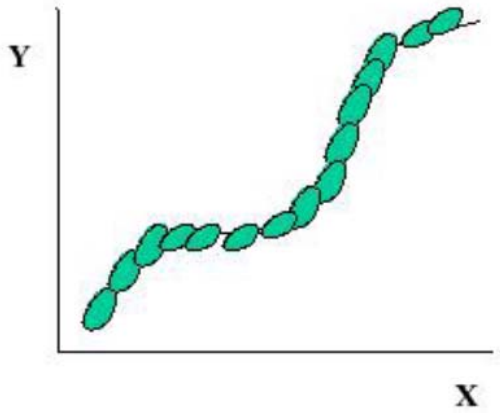


Curva de la función *f*



Parches determinados por cuatro reglas

Una mejor aproximación con un número mayor de reglas.



Las reglas que definen F se aproximan a la función f de un sistema cubriendo con parches (subconjuntos difusos del espacio producto X × Y) su curva: a mayor número de reglas mejor la aproximación.

Proceso de control simplificado de una turbina de vapor.

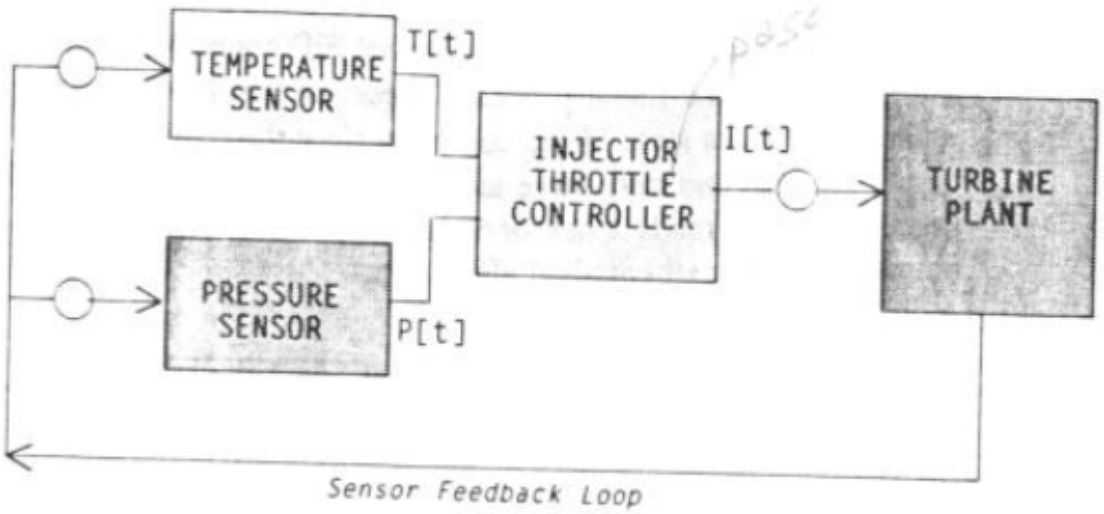


Figure 8.3 A Simplified Steam Turbine Control Process

Difusificación del dominio de valores de la variable temperatura

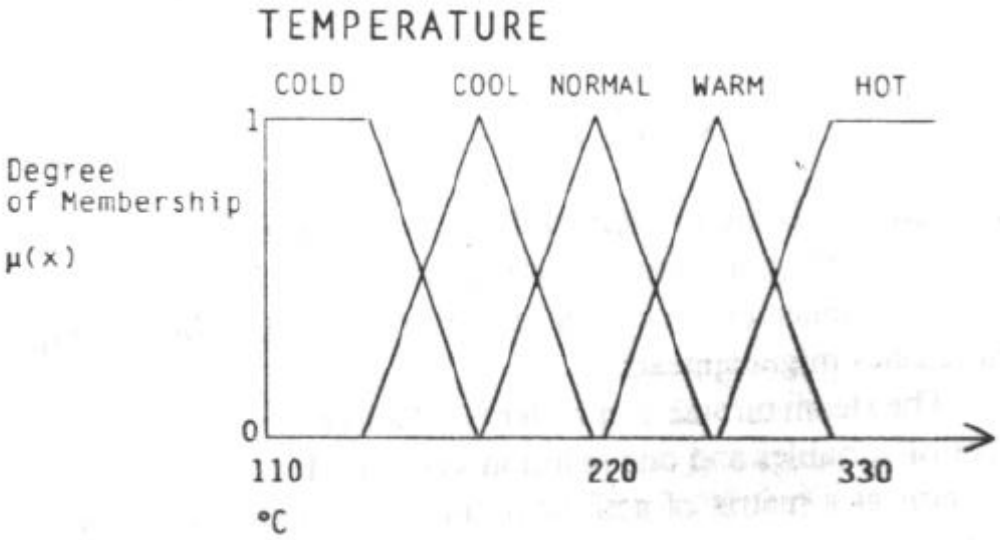


Figure 8.5 The Temperature Control Variable

Difusificación del dominio de valores de la variable presión

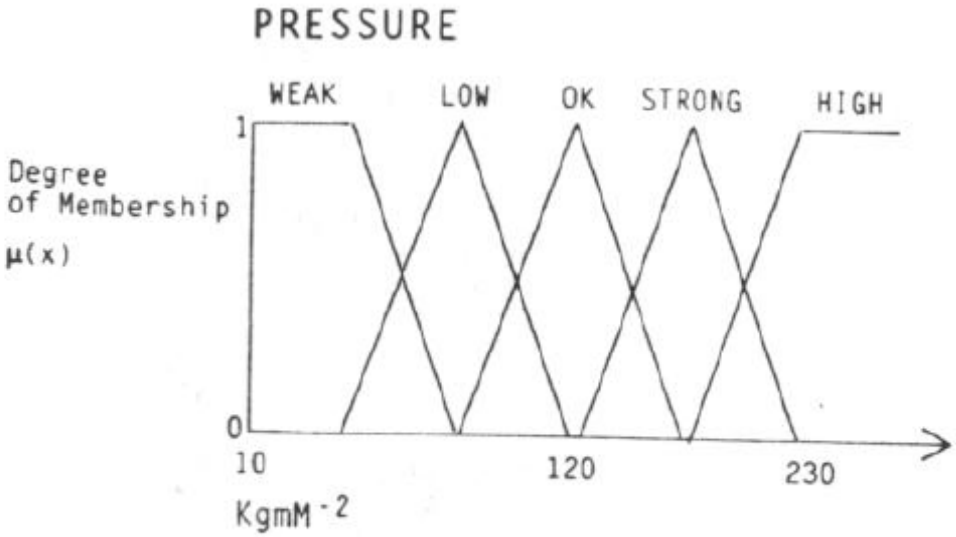


Figure 8.6 The Pressure Control Variable

Difusificación del dominio de valores de la variable apertura del inyector de vapor

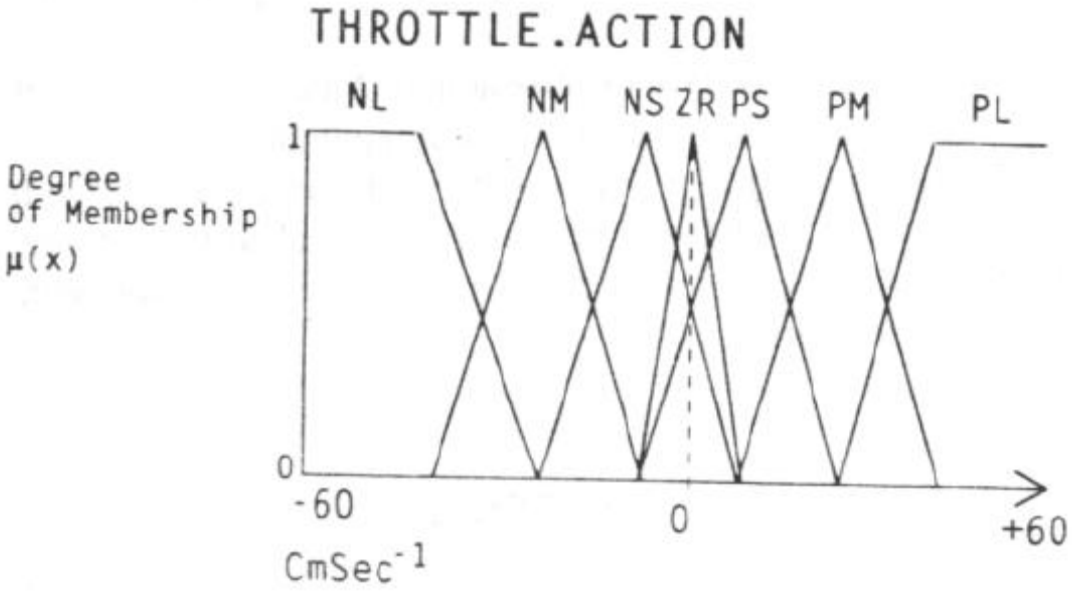


Figure 8.7 The Throttle.Action Solution Variable

Algunas reglas difusas para el control de la turbina de vapor

```
[R1] if temperature is COOL and pressure is WEAK
        then throttle action is PL

[R2] if temperate is COOL and pressure is LOW
        then throttle action is PM

[R3] if temperature is COOL and pressure is OK
        then throttle action is ZR

[R4] if temperature is COOL and pressure is STRONG
        then throttle action is NM
```

Figure 8.8 Sample Rules from the Steam Turbine Controller

Arreglo que contiene todas las reglas

PRESSURE / TEMP.	WEAK	LOW	OK	STRONG	HIGH
COLD	PL	PM	PS	NS	NM
COOL	PL	PM	ZR	NM	NM
NORMAL	PM	PS	ZR	NS	NM
WARM	PM	PS	NS	NM	NL
HOT	PS	PS	NM	NL	NL

Figure 8.9 FAM for the Steam Turbine Controller

Resultado de la aplicación de la primera regla

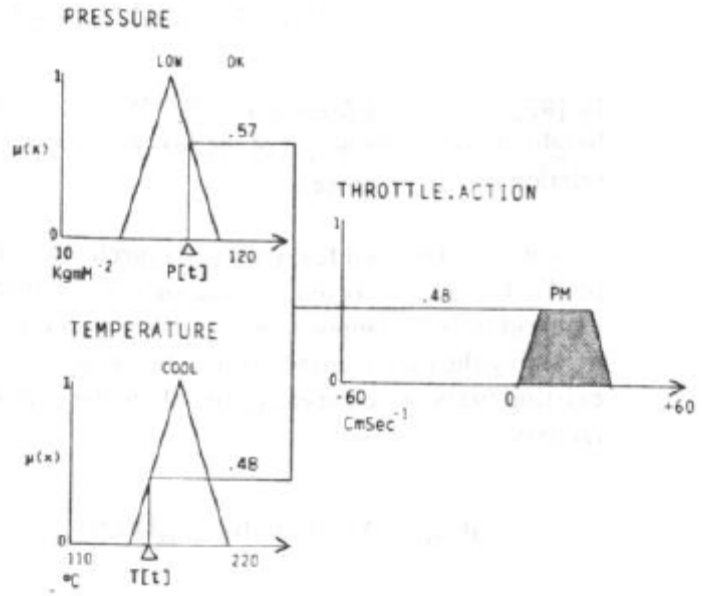


Figure 8.10 Throttle.Action After Applying the First Rule

Resultado de la aplicación de la segunda regla

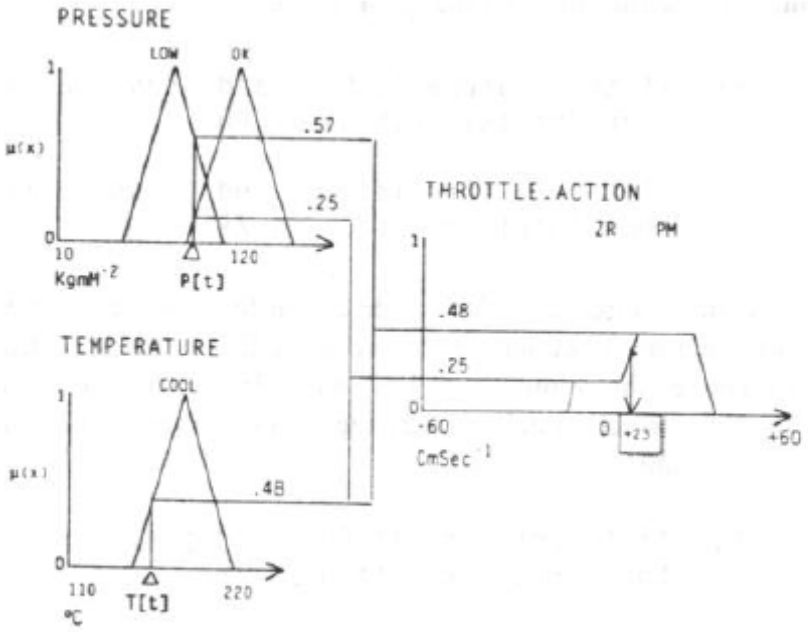


Figure 8.11 Throttle.Action After Applying the Second Rule

Formas alternas de desdifusificación

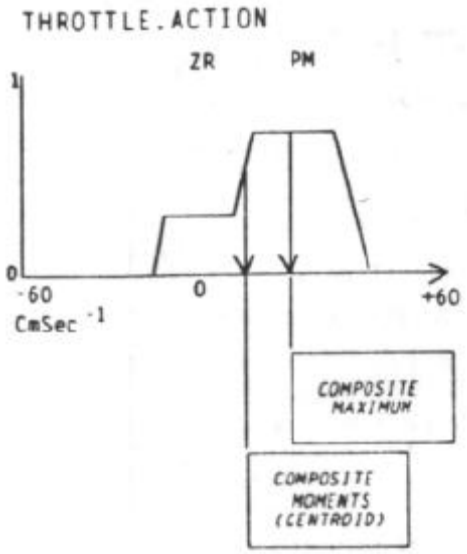


Figure 8.12 Alternative Ways of Interpreting the Output Fuzzy Region

5.2.2. La maldición de la dimensionalidad en los sistemas difusos: la explosión de reglas

Se necesitan muchas reglas para aproximar la mayoría de las funciones. Se puede apreciar que el número de reglas crece exponencialmente con el número de variables de entrada y de salida. Se necesitan todas las reglas para realizar la transformación de la entrada en la salida. Principio muy aceptable para cualquier método de modelación de sistemas: usar el menor número de variables y optimizar sus relaciones y dependencias con el menor número de reglas. Los sistemas difusos que funcionan en la práctica tiene pocas variables y pocas reglas.

5.3. Ejercicios

1. Aplique las definiciones de los operadores lógicos para la lógica trivalente en el caso de las tres operaciones mencionadas y llene sus respectivas columnas con los valores correspondientes.

   Demuestre que bajo las definiciones anteriores se cumplen las leyes de DeMorgan.

2. Construya la columna de valores correspondiente a A) B utilizando la definición dada por Lukasiewicz. Analice los valores obtenidos y diga qué aspectos relevantes de esta definición puede señalar.

3. A partir de la definición)$_L$ (implicación de Lukasiewicz) defina la operación de equivalencia lógica n-valente de Lukasiewicz.

4. Defina a partir de ,$_L$ el OR-exclusivo de Lukasiewicz.

5. Diseñe el sistema difuso para el control de una lavadora, si se conoce que posee dos sensores uno de PH y otro de peso (se sabe que el PH mientras más ácido más sucia la ropa). Se quiere que el sistema difuso determine la cantidad de agua, detergente y la temperatura del agua.

Capítulo 6: Agentes

6.1. Agentes inteligentes.

¿Qué es un agente?

Una forma obvia de comenzar debería ser presentando una definición del *término Agente*. Sorprendentemente no existe una definición universalmente aceptada, de hecho el debate y la controversia sobre este tema continúa. Gran parte de este problema esta dado por que varios atributos asociados con este término tienen más o menos importancia en dominios diferentes. Así pues, la habilidad de los agentes para aprender de sus experiencias anteriores es de vital importancia; sin embargo para otras aplicaciones, el aprendizaje no solo es insignificante sino que se resulta indeseable.

No obstante, algunos tipos de definiciones son importantes. A continuación presentaremos una definición tomada de [], en la cual se define el término agente como:

UN *AGENTE* ES UN SISTEMA COMPUTACIONAL SITUADO DENTRO DE UN *MEDIO-AMBIENTE*, DENTRO DEL CUAL ES CAPAZ DE REALIZAR ACCIONES AUTÓNOMAS EN CAMINADAS A LOGRAR SUS OBJETIVOS.

Hay varios puntos que se deben hacer notar en esta definición. Primero: Esta definición hace referencia a "agentes" y no a "agentes inteligentes". Segundo: esta definición no hace referencia a en que tipo de ambientes se pueden encontrar al agente. Tercero: no se brinda una definición para el término *automonía*. La autonomía es un concepto muy delicado como para definirlo de una forma precisa; pero en lo adelante lo usaremos para reflejar que un agente es capaz de actuar sin intervención de algún ser humano u otro sistema: es decir tiene completo control tanto sobre sus estados internos como sobre su comportamiento.

Figure 1.1 Un agente en su medio ambiente. El agente recibe estímulos de su medio y realiza acciones sobre este. Esta interacción es en muchas ocasiones un proceso continuo e infinito.

En la figura 1.1 se muestras una representación abstracta de un agente. En este diagrama se puede observar las acciones generadas por el agente para afectar el medio donde se encuentra.

En dominios que presentan una complejidad razonable, un agente no tiene un control completo sobre el ambiente, en el mejor de los casos tendrá un control parcial, sobre lo que puede influir. Desde el punto de vista del agente esto significa que la misma acción efectuada en circunstancias aparentemente iguales, pueden tener efectos completamente diferentes, y por lo tanto puede fallar en obtener el efecto deseado. Por esto, un agente situado en el más trivial de los ambientes debe estar preparado para la posibilidad de fallar. Podemos resumir esta situación formalmente que los ambientes son no-deterministas.

Normalmente, un agente tendrá disponible un repertorio de acciones. Este conjunto de acciones posible representa la capacidad ejecutiva del agente: su habilidad para modificar el medio-ambiente. Nótese que no todas las acciones se pueden realizar en todas las situaciones. Por ejemplo, la acción "*levantar mesa*" solo puede ser aplicada en situaciones donde el peso de la mesa sea lo suficientemente pequeño para que el agente "pueda" levantarla. Las acciones, por consiguiente, tienen *pre-condiciones* asociadas a ellas, las cuales definen las posibles situaciones en las cuales pueden ser aplicadas.

Ejemplos de agentes

Veamos a continuación algunos ejemplos de agentes (aunque todavía no son agentes inteligentes):

• Cualquier sistema de control puede ser visto como un agente. Por ejemplo, supongamos que ese sistema sea un termómetro.

El termómetro tiene un censor para determinar la temperatura. Este censor se encuentra completamente dentro del ambiente (una habitación, por ejemplo) y produce como salida una de dos señales: una que indica que la temperatura es demasiado baja y otra la cual indica que la temperatura esta bien. Las acciones con las que dispone el agente son: "*encender-calentador*" o "*apagar-calentador*". El efecto de la acción "*encender-calentador*" es la de subir la temperatura de la habitación para esto no puede ser garantizado: por ejemplo, si la puerta de la habitación esta abierta, los cambios que se realicen no tendrán efecto. La componente (extremadamente simple) de toma de decisión de una implementación del termómetro tiene las siguientes reglas:

demasiado frío → encender-calentador
temperatura OK → apagar-calentador

Los sistemas de control de ambientes más complejos tienen, por supuesto, una estructura de decisión mucho más rico. Por ejemplo, pruebas espaciales, vuelo de aviones no tripulados, sistema de control para rectores nucleares, etc.

• Los *daemons*, (como los procesos de background en UNIX), los cuales monitorean un ambiente de software y realizan acciones para modificarlo, pueden ser vistos como agentes. Un ejemplo es el programa de XWindows: *xbiff*. Este utilitario monitorea continuamente en espera de la llegada de un correo electrónico e indica, a través de un icono si hay o no mensajes nuevos.

A diferencia de muestro termómetro, biff "habita" un ambiente de software. Obtiene información sobre su medio ejecutando funciones de software (ejecutando programas del sistema como ls, por ejemplo) y las acciones que realiza son acciones de software (cambiar un icono en la pantalla o ejecutar un programa). La componente de toma de decisión es más simple que la del ejemplo del termómetro.

6.1.1. Características de los ambientes

El problema que enfrenta un agente es el de decidir cual de sus acciones puede ejecutar para satisfacer sus objetivos de la mejor forma. La complejidad del proceso de toma de decisiones es afectado por diferentes propiedades que presentan los ambientes. En *Reading in Agents*, Russell & Norving sugieren la siguiente clasificación para las propiedades de los ambientes:

• Accesible vs. inaccesible.
Un ambiente accesible es aquel en el cual el agente puede obtener información completa y actualizada sobre el estado del medio. Los ambientes más complejos (incluyendo el mundo real e Internet) son inaccesibles. Mientras más accesible sea un ambiente más fácil será construir agentes que operen en él.

• Determinista vs. no-determinista.

Un Ambiente determinista es aquel en el cual cualquier acción se puede garantizar que tiene un único efecto: no hay incertidumbre sobre el estado en que quedará el ambiente después de realizar una acción. El mundo real puede ser catalogado de no-determinista. Los ambientes deterministas no presentan un gran problema para los programadores de agentes.

• Episódico vs. no-episódico.
En un ambiente episódico, el rendimiento de un agente depende de un número de episodios discretos, no existe ningún vínculo entre el rendimiento de un agente en escenarios diferentes. Un sistema para el ordenamiento de correos es un ejemplo de un ambiente episódico. Estos ambientes son mas singles desde el punto de vista de los programadores debido a que el agente puede decidir que acción debe realizar basado solamente en el episodio actual: el agente no tiene que razonar sobre las consecuencias que tendrá esto en el futuro.

• Estático vs. Dinámico.
Un ambiente estático es aquel se mantiene inalterable a no ser que el agente realice una acción sobre él. Un ambiente dinámico es aquel en el que existen otros procesos operando sobre el y por consiguiente los cambios en él están fuera del control del agente. El mundo real es un ambiente altamente dinámico.

• Discreto vs. continuo.
Un ambiente es discreto si existe un número fijo y finito de acciones y en el *percepts*. El juego de ajedrez se desarrolla sobre un ambiente discreto, mientras que manejar un taxi se realiza sobre uno continuo.

La clase mas compleja de ambientes son aquellos que son: inaccesibles, no-deterministas, no-episódicos, dinámicos y continuos.

6.1.2. Agentes Inteligentes

No estamos acostumbrados a ver a un termómetro o a los *daemons* de UNIX como agentes y mucho menos como agentes inteligentes. Entonces, ¿Cuándo consideraremos que un agente es inteligente? Esta pregunta, al igual que la pregunta de: ¿Qué es inteligencia?, no es fácil de responder. Pero para el propósito de este curso podemos definirlo como:

UN AGENTE INTELIGENTE ES AQUEL QUE ES CAPAZ DE REALIZAR ACCIONES AUTÓNOMAS *FLEXIBLES* PARA LOGRAR SUS OBJETIVOS, EN DONDE *FLEXIBLE* SIGNIFICA TRES COSAS:

• REACTIVO: UN AGENTE INTELIGENTE DEBE SER CAPAZ DE PERCIBIR SU AMBIENTE Y RESPONDER DE UN MODO OPORTUNO A LOS CAMBIO QUE OCURREN PARA LOGRAR SUS OBJETIVOS;

• PRO-ACTIVO: UN AGENTE INTELIGENTE DEBE SER CAPAZ DE MOSTRAR UN COMPORTAMIENTO *GOAL-DIRECTED TOMANDO LA INICIATIVA* PARA LOGRAR SUS OBJETIVOS.

• SOCIABLE: UN AGENTE INTELIGENTE DEBE SER CAPAZ DE INTERACTUAR CON OTROS AGENTES (POSIBLEMENTE TAMBIÉN CON HUMANOS), PARA LOGRAR SUS OBJETIVOS.

Estas propiedades son más exigentes de los que aparecen a simple vista. Para ver porque veámoslas una a una.

Primeramente consideremos la pro-actividad. No es difícil construir un sistema que tenga un comportamiento goal-directed: de hecho, lo hacemos cada vez que escribimos una función en PASCAL, C++, o en JAVA. Cuando programamos una función de este tipo, la describimos en término de sus pre-condiciones y el efecto que tendrá (sus post-condiciones). El efecto del procedimiento son sus objetivos. Entonces, si se cumplen las pre-condiciones el procedimiento es invocado y esperamos que el procedimiento sea ejecutado correctamente hasta que termine, una vez terminado las post-condiciones serán verdaderas. El procedimiento es solo un plan para alcanzar el objetivo.

En estos modelos se asume que el ambiente no va a cambiar mientras el procedimiento se ejecuta. Si el ambiente cambia y las precondiciones se vuelven falsas durante el proceso, entonces el comportamiento del procedimiento se indefine o simplemente falla. También se asume que durante la ejecución del procedimiento el objetivo sigue siendo valido hasta que este termine, pero si el objetivo deja de ser válido no hay razón para seguir ejecutando el procedimiento.

En estos ambientes dinámicos, el agente debe ser *reactivo*, tal como lo describimos antes. Esto es, el debe ser sensitivo a los eventos que ocurran en el ambiente, donde estos eventos afectan los objetivos del agente o las suposiciones en las que se basa el proceso que el agente esta ejecutando para lograr sus objetivos.

Por otro lado no queremos que nuestro agente este constantemente reaccionando y por lo tanto nunca se enfoque en un objetivo el tiempo necesario para lograrlo

Como hemos visto, construir sistemas puramente pro-activo o puramente reactivo no es difícil. Pero, lo difícil es construir sistemas que consigan un balance efectivo entre el comportamiento reactivo y el de proactivo.

Finalmente hablemos de la *sociabilidad*. En sentido la sociabilidad es trivial: cada día millones de computadoras en todo el mundo intercambian información, tanto con humanos como con otras computadoras. Pero la habilidad de intercambiar bits no es sociabilidad. Consideremos que en el mundo humano, en comparación parte de nuestros objetivos pueden ser llevados a cabo sin la cooperación de otras personas, las cuales son también autónomas y tienen su propia agenda. Para lograr entonces nuestros objetivos debemos *negociar* y *cooperar* con otros, debemos entender y razonar sobre los objetivos de los otros y realizar algunas acciones que no teníamos programadas para lograr su cooperación y nuestros objetivos. Este tipo de sociabilidad es mucho más complejo y menos comprensible que la simple habilidad de intercambiar información.

Por el momento nos concentraremos con la toma de decisiones de agentes inteligentes individuales en ambientes que pueden ser dinámicos, impredecibles y no-deterministas pero que no contiene otros agentes.

6.1.3. Agentes vs. Objetos

Usualmente los programadores de objetos no encuentran ninguna novedad en la idea de agentes. Si consideramos las propiedades de los agentes y los objetos, quizás no nos sorprenda. Los objetos están definidos como entidades computacionales que encapsulan algún estado, son capaces de realizar acciones en ese estado y se comunican a través de paso de mensajes. Aunque existen similitudes obvias, existen grandes diferencias entre agentes y objetos.

Primero en que grado son autónomos los agentes y los objetos. Recordemos que la característica que define a la programación orientada a objetos es el principio de encapsulamiento: los objetos tiene control sobre sus estados internos. En lenguajes de programación como java podemos declarar variables y métodos privados, por lo que solo podrán ser accesibles desde el objeto. De esta forma se puede creer que tiene autonomía sobre sus estados. Pero los objetos no tienen control sobre su comportamiento. Si un método *m* se hace público a otros objetos, entonces estos podrán invocarlos cada vez que quieran (una vez que un objeto tiene un método público, entonces este no tiene control sobre si se debe o no ejecutar el procedimiento). Por supuesto, un objeto debe tener métodos disponibles a otros objetos, o sino no pudiéramos construir sistemas más allá de él.
Si construimos un sistema diseñamos los objetos que estarán en él y podemos asumir que esos objetos comparten un objetivo común. Pero en muchos sistemas multiagentes no se puede asumir que tengan objetivos comunes. Un agente *i* no ejecutará una acción (o método) *a* solo porque otro agente *j* lo quiera. Si *j* pide a *i* realizar una acción *a*, entonces *i* puede o no realizarla. El control con respecto a la decisión de ejecutar o no una acción es diferente en agentes y objetos. En el caso de objetos la decisión esta en el objeto que invoca el método y en el caso de agentes la decisión esta en el agente que recibe la petición. Esta distinción se puede resumir con la frase: los objetos lo hacen gratis, los agentes por dinero.
La segunda distinción importante entre agentes y objetos es con respecto a la noción del comportamiento autónomo flexible. El modelo estándar de objeto no tiene nada en absoluto que decir sobre como construir sistemas que integren este tipo de comportamiento. Podemos construir programas usando técnicas de programación orientada a objetos que integren esto tipo de comportamiento; pero este argumento no encierra el punto principal del problema, que el modelo de programación orientada a objetos no tiene nada que ver con este tipo de comportamiento.

La tercera diferencia es que cada agente se considera que tenga su propio hilo de control y en el modelo estándar de programación orientada a objeto hay un solo hilo de control en el sistema. Por supuesto, recientemente hay una gran cantidad de trabajos dedicados a la concurrencia en la programación orientada a objetos. Por ejemplo; JAVA, entre otros,

provee constructores built-in para la programación multi-hilos. Pero esto no capta la idea que tenemos de agentes como entidades autónomas.

• Los agentes tienen una noción más fuerte de autonomía que los objetos; en particular, un agente decide por si mismo si realiza o no una acción a pedido de otro agente.

• Los agentes tienen un comportamiento flexible (reactivo, pro-activo y sociable) y el modelo de objeto no tiene nada que decir sobre este tipo de comportamiento.

• Un sistema multiagente es inherentemente multi-hilos, en el que cada agente tiene al menos un hilo de control.

Note que no hay nada que impida que implementemos los agentes usando técnicas de programación orientadas a objetos.

6.1.4. Ejercicios

1. De otros ejemplos de agentes (no necesariamente inteligentes) que ud. conozca. Para cada uno defina:

   a) El ambiente que ocupa, los estados que ese ambiente puede tener y sus características.

   b) El repertorio de acciones con que cuenta el agente y cualquier precondición que tengan asociadas estas acciones.

   c) El objetivo del agente.

6.2. Arquitecturas abstractas para agentes.

Podemos formalizar fácilmente la representación abstracta de un agente presentada anteriormente. Pero antes, vamos a asumir que los estados del ambiente pueden ser caracterizados por un conjunto $E = \{ s_1, s_2, ...\}$de estados del ambiente. En cualquier instante dado, asumiremos que el ambiente se encuentra en uno de estos estados. La capacidad ejecutiva del agente la podemos representar como por un conjunto $Ac = \{ a_1, a_{2,...}\}$ de acciones. Entonces abstractamente podemos ver a un agente como una función

$$Ag: E^* \rightarrow Ac$$

la cual dada una secuencia de estados del ambiente da como resultado una acción. En lo adelante llamaremos a un agente modelado por una función de esta forma como agente estándar. De esta forma un agente decide que acción va a realizar sobre la base de la historia del sistema que el ha presenciado hasta el momento.

El comportamiento no-determinista de un ambiente puede ser modelado como una función de transformación:

$$t : E \times A \rightarrow \wp(S)$$

La cual toma el estado actual del ambiente e $\in E$ y una acción a $\in Ac$. (realizada por el agente) y devuelve un conjunto de estados del ambiente (*t(e, a)*), que pueden ser el resultado de aplicar *a* en el estado e. Si todos los conjuntos que forman parte de la imagen de *t* son simples, entonces el ambiente es determinista. Por lo que, un ambiente puede ser representado como un par: $<E, e_0, t>$, donde *E* es el conjunto de estados del ambiente, e_0 el estado inicial de *E*, *t* una función de transformación.

De la misma forma podemos representar la interacción del agente con el ambiente como una secuencia:

$$h: e_o \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} e_3 \xrightarrow{a_3} \ldots \xrightarrow{a_{n-1}} e_n \xrightarrow{a_n} \ldots$$

donde s_0 es el estado inicial del ambiente (o el estado en que se encontraba el ambiente cuando se empezó a ejecutar el agente), a_n es la n-ésima acción que el agente escogió para ejecutar y s_n es el n-ésimo estado que toma el ambiente (el cual es uno de los posibles resultados de $t(a_{n-1}, s_{n-1})$).

Si, $Ag: E^* \rightarrow Ac$, es un agente y $t: E \times Ac \rightarrow \wp(E)$, es un ambiente y s_0 es el estado inicial del ambiente, entonces la secuencia:

$$h: e_o \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} e_3 \xrightarrow{a_3} \ldots \xrightarrow{a_{n-1}} e_n \xrightarrow{a_n} \ldots$$

representa una posible *historia*(o *corrida*) del agente *Ag* en el ambiente si $<E, e_0, t>$ y solo si se cumple:

$$\forall n \in \mathbf{N}, \; a_u = Ag(e_0, e_1, \ldots, e_n) \text{ y } \forall n \in \mathbf{N}, n > 0, \; e_u \in t(e_{n-1}, a_{n-1})$$

El *comportamiento característico* de un agente *Ag*, en un ambiente *E*, es el conjunto de todas las historias que satisface esta propiedad. Denotaremos por *hist(Ag, E)* al conjunto de todas las historias del agente *Ag* en el ambiente *E*. Dos agentes Ag_1 y Ag_2 se dicen que tiene *conductas equivalentes* con respecto a un ambiente *E* si y solo si $hist(Ag_1, E) = hist(Ag_2, E)$ y diremos que tiene *conductas equivalentes* si y solo si tienen *conductas equivalentes* con respecto a todos los ambientes.

En general trabajaremos con agentes cuyas interacciones con el ambiente no terminan y en tales casos, consideraremos que sus historias son infinitas.

6.2.1 Agentes puramente reactivos.

Ciertos tipos de agentes deciden que hacer sin hacer referencia a su historia. Ellos basan su decisión enteramente en el presente, sin referencia a todo lo pasado. A este tipo de agentes los llamaremos agentes puramente reactivos, debido a que simplemente responden directamente al ambiente. Formalmente, el comportamiento de estos agentes puede ser representado con una función $Ag: E^* \rightarrow Ac$.

Se puede ver a simple vista que para cada agente puramente reactivo, existe un agente equivalente estándar; sin embargo al revés, no se cumple generalmente. El agente termómetro es un agente puramente reactivo. Asumamos, sin perdida de generalidad, que el ambiente donde se encuentra el termómetro, tiene dos estados: *temperatura-baja* y *temperatura-ok*. Entonces el agente termómetro se puede definir:

$$Ag(s) = \begin{cases} \text{apagar-calentador, si e = temperatura-ok} \\ \text{encender-calentador, e.o.c.} \end{cases}$$

6.2.2. Percepción

Ver los agentes a este nivel abstracto nos simplifica el análisis. Sin embargo, no nos ayuda a construirlos, ya que no nos brinda pista de como diseñar la función *Ag*. Por esta razón, empezaremos a refinar el modelo abstracto de agente, dividiéndolo en subsistemas de la misma forma en que lo hacemos en ingeniería de software.

Lo primero que haremos es dividir la función de decisión del agente en los subsistemas de percepción y acción (ver figura 1).

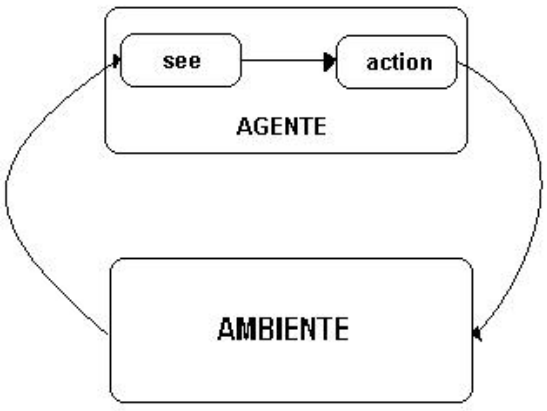


Figura 1. Subsistemas de Percepción y acción.

La idea es que la función *see* capta la habilidad de observar su ambiente, mientras que la función *action* representa el proceso de toma de decisión del agente. La función *see* podría ser una pieza de hardware en el caso de que el agente este situado en el mundo físico: una cámara de video o un sensor infrarrojo en un robot móvil, por ejemplo. Para

un agente software, el sensor podrían ser comandos el sistemas que obtiene información sobre el ambiente software tales como: *ls, finger* u otros por el estilo. Lo que da como salida esta función *see* es una percepción: una entrada perceptiva. Sea *P* un conjunto no vació de percepciones. Entonces la función *see* queda definida como:

$$see: E \rightarrow P$$

la cual mapea los estados del sistemas en percepciones, la función *Ag* queda entonces definida:

$$action: P^* \rightarrow A$$

La cual toma como entrada una secuencia de percepciones y da como salida una acción.

Esta simple definición nos permite explorar algunas propiedades de los agentes y la percepción. Supongamos que tenemos dos estados del ambiente (e_1 y e_2) y que $see(e_1) = see(e_2)$. Entones dos estados diferentes del ambiente son convertidos en la misma percepción, por lo que el agente recibirá la misma información perceptiva de diferentes estados del ambiente, por lo que e_1 y e_2 son indistinguibles para el agente. Por ejemplo, retornemos al ejemplo del termómetro. Sea *x* tal que representa la proposición "*temperatura-ok*" y sea *y* la proposición "*Juan es primer ministro*".

Si estos son los dos únicos hechos que nos interesan sobre nuestro ambiente, entonces el conjunto *E* tiene cuadro elementos:

$$E = \{\underbrace{\{\neg x, \neg y\}}_{e_1}, \underbrace{\{\neg x, y\}}_{e_2}, \underbrace{\{x, \neg y\}}_{e_3}, \underbrace{\{x, y\}}_{e_4}\}$$

Entonces el estado s_1, la temperatura no esta ok y Juan no es primer ministro; en s_2, la temperatura no es ok y Juan si es primer ministro. Ahora bien, el termómetro solo es sensible a la temperatura de la habitación y la temperatura no esta relacionada con si Juan es primer ministro o no. Entonces, la función *see* del termómetro tiene dos percepciones como resultado: p_1 y p_2, las cuales indican que la temperatura está baja o esta OK, respectivamente. La función *see* quedaría definida:

$$see(e) = \begin{cases} p_1 & si\ e = e_1 \text{ o } e = e_2 \\ p_2 & si\ e = e_3 \text{ o } e = e_4. \end{cases}$$

Dados dos estados del ambiente $e_1 \in E$ y $e_2 \in E$, diremos que $e_1 \equiv e_2$, si $see(e_1) = see(e_2)$. No es difícil notar que $\equiv$ es una relación de equivalencia, la cual particiona a *S* en estados mutuamente indistinguibles. Si $| \equiv | = | E |$, entonces el agente puede distinguir cada uno de los estados del ambiente: el agente tiene una percepción total del ambiente. Por otro

lado, si | ≡ | = 1, entonces el agente no tiene habilidad perceptiva: no puede distinguir ningún estado del ambiente. En este caso, para el agente, todos los estados del sistema son idénticos.

6.2.3. Agentes con estados

Hasta ahora hemos modelado las funciones de acción de los agentes de secuencias de estados del ambiente o percepción a acciones. Esto nos permite representar agentes cuya toma de decisión esta influenciada por la historia del ambiente. Sin embargo, es una representación algo intuitiva, por lo que ahora la reemplazaremos por un modelo equivalente pero más natural. La idea es consideremos a agentes que mantienen estados (ver figura #3). Estos agentes una estructura de dato interna, la cual es normalmente usada para guardar información sobre el estado del ambiente y la historia.

Sea *I* el conjunto de todos los estados internos del agente. El proceso de toma de decisión de los agentes de este tipo esta basado entonces en esta información. La función de percepción para agentes que mantienen estados que igual que antes:

$$see: E \rightarrow P$$

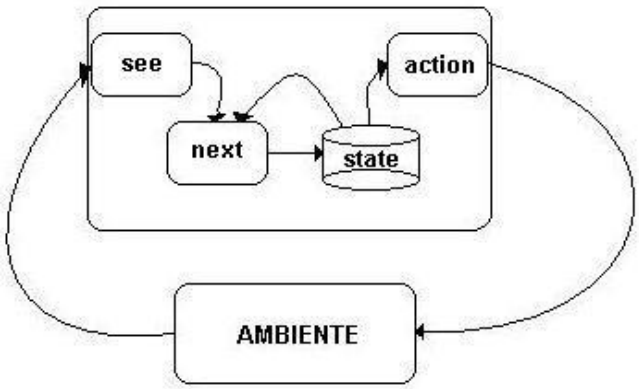


Figure 2. Agentes con estados.

La función de selección de acción queda ahora definida:

$$\boldsymbol{Ag: I \rightarrow Ac}$$

De estados internos a acciones. Se introduce una nueva función, *next*, la cual esta definida a partir de un estado interno y una percepción a un estado interno:

$$\boldsymbol{next: I \times P \rightarrow I}$$

El comportamiento de un agente con estados puede ser resumido de la manera siguiente. El agente empieza en un estado inicial i_0. Entonces observa el estado actual del ambiente *s* y genera una percepción *see(e)*. El estado interno del agente se actualiza a través de la función *next*, pasando a ser $next(i_0, see(e))$. La acción que realiza el agente es entonces $Ag(next(i_0, see(e)))$. Esta acción es ejecutada y el agente entra en otro ciclo.

Vale la pena observar que los agentes basados en estados definidos aquí, no son más poderosos que los agentes estándar que se definieron con anterioridad; de hecho son idénticos en su poder expresivo: cada agente basado en estados puede ser transformado a un agente estándar de *comportamiento equivalente*.

6.2.4. ¿Cómo entregar tareas a los agentes?

Normalmente construimos agentes para que realicen alguna tarea por nosotros. Para que el agente realice esa tarea, debemos de algún modo comunicarle al agente la tarea que queremos que realice. Una forma de especificar la tarea pudiera ser simplemente escribir un programa para que el agente lo ejecute.

Una ventaja obvia de este método es que no dejamos incertidumbre sobre lo que el agente debe hacer. Pero la desventaja más obvia que tiene es que tenemos que pensar, exactamente, como realizar la tarea nosotros mismos, y si se presenta alguna circunstancia imprevista, el agente que esta ejecutando la tarea no responder adecuadamente a este imprevisto. Usualmente nosotros queremos *decirle al agente que tiene que hacer sin decirle como lo tiene que hacer*. Una forma de lograr esto es definir las tareas de forma indirecta. Hay muchas maneras de hacer esto y una de ellas es asociar una *ganancia* a los estados del ambiente.

6.2.5. Funciones de utilidad.

Una utilidad es un valor numérico que representa cuan bueno es el estado: mientras más grande sea ese valor mejor será el estado. La tarea del agente es entonces lograr estados que maximicen la utilidad. De esta forma la tarea puede ser definida como una función:

$$u: E \rightarrow \Re$$

la cual asocia un valor real con cada estado del ambiente. Dada esta medida del rendimiento, podemos definir la utilidad total del agente de diferentes formas.

Una forma es definirlo por el peor estado que el agente haya logrado; y otra forma es tomando el promedio de las utilidades de todos los estados que el agente haya logrado. La medida depende del tipo de tarea que queremos que el agente realice.

Una de las principales desventajas que este enfoque es el que asignando una utilidad a estados locales, es difícil especificar una perspectiva a largo plazo cuando se asignan utilidades a estados individuales. Para resolver este problema se puede en vez de asignarle una utilidad a los estados sino a las corridas en si mismas.

$$u: R \rightarrow \Re$$

Si se está considerando que el agente debe operar independientemente por un período de tiempo largo, este enfoque parece ser más apropiado.

Veamos un ejemplo de una función de utilidad (Tileworld).

Aquí se tiene una matriz de bidimensional como ambiente, en la cual se encuentra agentes, tejas, huecos y obstáculos. Los agentes e pueden mover en cuadro direcciones: arriba, abajo, a la derecha y a la izquierda; si el agente se encuentra al lado de una teja este puede empujarla. Un obstáculo es un conjunto de celdas, por las cuales el agente no puede pasar. Los agentes deben rellenar los huecos con tejas y así ganan puntos. La meta de los agentes es rellanar tantos huecos como pueda. Este ambiente es dinámico, empieza en un estado generado aleatoriamente y cada determinado tiempo cambia por la aparición y desaparición de huecos. El rendimiento de un agente en una corrida en particular se define como:

$$u(r) = \frac{\textit{Número de huecos llenados en r}}{\textit{Número de huecos que aparecieron en r}}$$

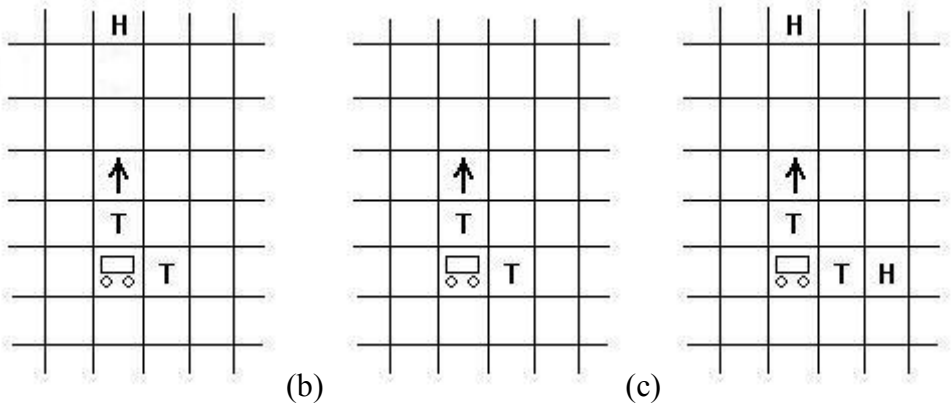


Figura 3. Tres escenas en el Tileworld; (a) el agente localiza un hueco adelante y empieza a empujar una teja hacia él; (b) el hueco desaparece antes que el agente pueda rellenarlo, el agente debe reorganizarse ante este cambio y modificar su comportamiento apropiadamente; (c) mientras el agente empuja una teja hacia el hueco del norte, aparece un hueco a su derecha, aquí es mejor empujar la teja de la derecha y después continuar hacia arriba.

A pesar de su simpleza, este ejemplo nos permite examinar varias capacidades importantes de un agente. Quizás la más importante de estas capacidades es la de

reaccionar a los cambios del ambiente y explotar las oportunidades cuando estas aparecen (figura 3).

6.2.6. Predicados para especificar tareas.

Un predicado para especificar tareas aquel donde la función de utilidad actúa como un *predicado* sobre las corridas. Formalmente, diremos que una función de utilidad $u: R \rightarrow \Re$ es un predicado si su conjunto de llegada es {0, 1}, es decir que a cada corrida le asigne 1 (true) o 0 (false). Diremos que una corrida $r \in R$ satisface la especificación u si $u(r)=1$, y no la satisface si $u(r)=0$. Usaremos ψ para denotar un predicado y escribiremos $\psi(r)$ para indicar que r satisface a ψ. En otras palabras $\psi(r)$ es verdadero si y solo si $u(r) = 1$.

Tareas de ambientes

Una *tarea de ambiente* es un par $<Env, \psi>$, donde *Env* es un ambiente y $\psi: R \rightarrow \{0, 1\}$ un predicado sobre las corridas. Sea *TE* el conjunto de todas las *tareas de ambientes*. De esta manera las tareas de ambientes especifica: las propiedades del sistema en el que el agente habita; y también, el criterio con el cual el agente será juzgado si triunfa o falla en sus tareas

Dada una tarea de ambiente $<Env, \psi>$, llamaresmos $R_\psi(Ag, Env)$ al conjunto de todas las corridas del agente *Ag* en el ambiente *Env* que satisfacen ψ. Formalmente

$$R_\psi(Ag, Env) = \{ r / r \in R(Ag, Env) \text{ y } \psi(r) \}$$

Y diremos que el agente tiene éxito en la tarea de ambiente $<Env, \psi>$ si:

$$R_\psi(Ag, Env) = R(Ag, Env)$$

En otras palabras: $\forall r \in R(Ag, Env)$ se tiene que $\psi(r)$. Note que esta definición de éxito es un poco pesimista. Una alternativa, optimista, seria definir el éxito de un agente, si logra cumplir ψ en al menos una corrida:

$$\exists r \in R(Ag, Env) \text{ tal que } \psi(r)$$

La noción de predicados para la especificación de tareas es mas bien una manera abstracta de describir tareas para los agentes. De hecho, es la generalización de ciertas formas muy comunes de tareas. Quizás, los dos tipos de tareas más comunes que podamos encontrar son: *tareas a cumplir y objetivos a mantener*.

(1) *tareas a cumplir*: de la forma 'lograr un estado φ'

(2) *objetivos a mantener:* de la forma 'mantener el estado ψ'

(1) se puede definir formalmente como:

Una tarea de ambiente $<Env, \psi>$ es una *tarea a cumplir* si y solo si existe un conjunto $G \subseteq E$ tal que para toda $r \in R(Ag, Env)$ el predicado $\psi(r)$ es verdadero si y solo si existe algún $e \in G$ tal que $e \in r$.

Un ejemplo de este tipo de tareas es suponer que el agente juega contra el ambiente. El agente y el ambiente comienzan en algún estado; el agente juega realizando una acción y el ambiente responde con algún estado, el agente entonces realiza otra acción y así sucesivamente. El agente gana si logra llevar al ambiente a uno de los estados de G.

(2) de la misma forma se puede definir formalmente como:

Una tarea de ambiente $<Env, \psi>$ es un objetivo a mantener si y solo si existe un conjunto $\beta \subseteq E$ tal que para toda $r \in R(Ag, Env)$ el predicado $\psi(r)$ es verdadero si y solo si para todo algún $e \in \beta$ tal que $e \notin r$.

En este caso diremos que β es el conjunto de fallos y usaremos $<Env, \beta>$ para denotar un *objetivo a mantener*, con estados fallos β y ambiente *Env*.

Nuevamente podemos suponer que el agente juega contra el ambiente. Esta vez el agente gana si logra evadir todos los estados de β. El ambiente en su rol de oponente trata que el agente logre un estado de β, el agente gana si logra evadir β.

Tareas más complejas pueden ser logradas combinando estos dos tipos de tareas. Una de estas combinaciones pudiera ser: "lograr algún estado de G mientras se evitan todos los estados de β. Combinaciones más complejas pueden ser definidas.

6.2.7. Ejercicios

1. Pruebe que:
   (a) Para cada agente puramente reactivo existe un agente estándar de comportamiento equivalente.

   (b) Existen agentes estándar para los cuales no existen agentes puramente reactivos.

2. Probar que los agentes con estados son equivalentes en poder de expresividad a los agentes estándar. Para cada agente con estado existe un agente estándar de comportamiento equivalente y viceversa.

3. Estudiamos 2 formas para especificar tareas utilizando funciones de utilidad: asociando utilidades a los estados ($u : E \rightarrow \Re$) ó a las corridas ($u : R \rightarrow \Re$). La segunda es estrictamente más expresiva que la primera. De un ejemplo de una función de utilidad sobre las corridas que no pueda ser definida por con una función de utilidad sobre los estados.

6.3. Agentes de Razonamiento deductivo.

6.3.1. Introducción.

Hasta ahora hemos considerado a los agentes abstractamente. Hemos hablado de agente que con o sin estados, sin considerar cuales podrían ser estos estados; de la misma forma hemos modelado la el proceso de toma de decisión de los agentes como una función *action* abstracta, pero no hemos discutido como puede ser implementada esta función. En lo adelante nos dedicaremos a rectificar esta omisión.

Empezaremos por considerar una clase de agente llamada *Agentes de Razonamiento deductivo*.

La forma tradicional de construir sistemas inteligentes artificiales, conocida como IA simbólica, sugiere que el comportamiento inteligente puede ser generado en un sistema puede dándole una representación simbólica a su ambiente y al comportamiento deseado y manipular sintácticamente esta representación. En esta conferencia nos enfocaremos en este modelo, en el cual esta representación simbólica son fórmulas lógicas y la manipulación sintáctica se corresponde a la deducción lógica.

Primeramente, veamos un ejemplo para introducir de manera informal la idea que hay detrás de este tipo de agentes. Supongamos que tenemos un robot cuyo trabajo es recorrer las oficinas de un edificio recogiendo la basura. Una forma de implementar el sistema de control del robot es darle una representación del ambiente (ver figura 1).

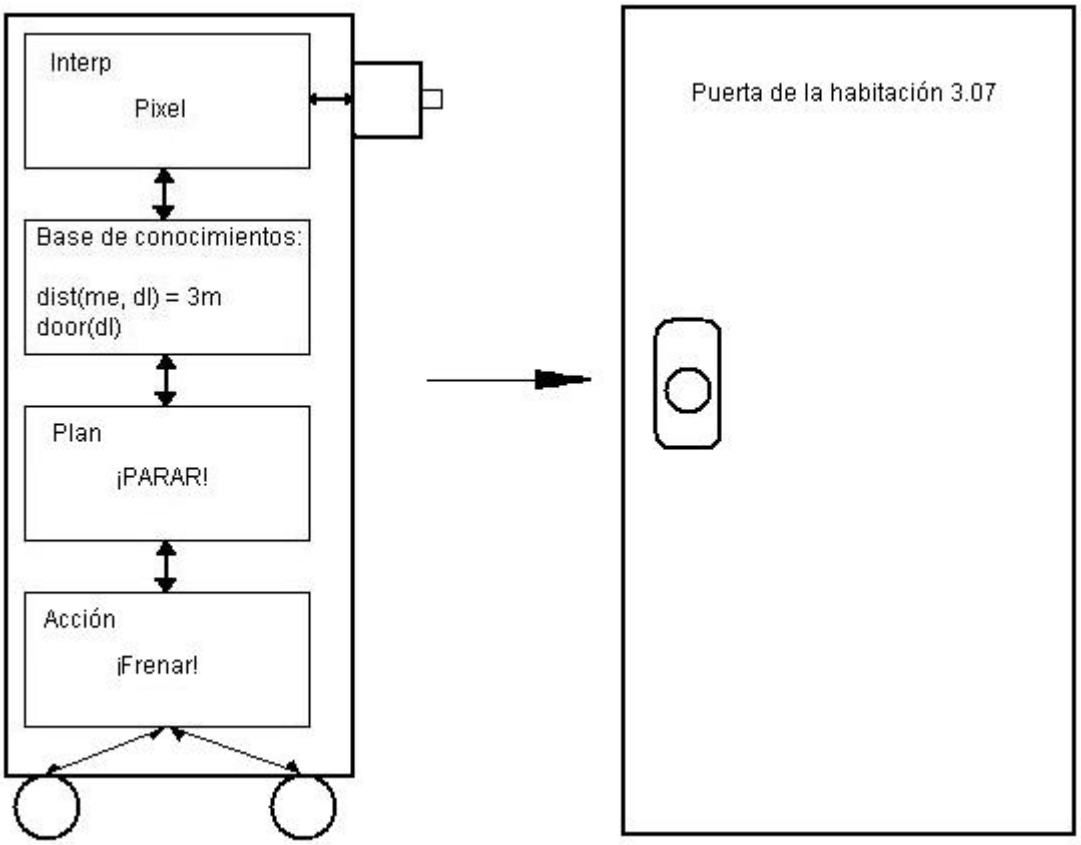


Figura 1. RALPH es un robot autónomo que opera en mundo de pasillos y puertas. Su censor de entrada es una cámara de video; un subsistema, llamado *Interp* traduce la señal de la cámara a una representación interna, basada en lógica de primer orden. La información sobre el mundo del agente esta dentro de una estructura de dato llamada "*base de conocimiento*".

Para construir este agente, debemos resolver dos problemas claves:

(1) El problema de la traducción. Como traducir el mundo real en una descripción simbólica adecuada que sea útil.

(2) El problema de la representación y el razonamiento. El problema de representar la información simbólica y que el agente la manipule para que los resultados sean útiles.

En primer problema tenemos que trabajar en: visión, reconocimiento del habla, aprendizaje, etc. En el segundo tenemos que trabajar en: representación de conocimiento, razonamiento automático, planeamiento automático, etc. Incluso problemas aparentemente triviales como el sentido del razonamiento es extremadamente difícil. A pesar de esto la idea de construir este tipo de agente es seductora.

6.3.2. Agentes deductivos

Para ver como trabaja esta idea, desarrollaremos un modelo de agente basado en lógica, los cuales llamaremos agentes deductivos. En tales agentes el estado interno es una base de datos de fórmulas de lógica de primer orden. Por ejemplo:

abierta (valvula221)
temperatura(reactor4726, 321)
presión (tanque776, 28)

Estas formulas pueden ser usadas para representar las propiedades de un ambiente. La base de datos es la información que tiene el agente sobre el ambiente y juega un rol parecido al de los conocimientos en los seres humanos.

Por supuesto, como los seres humanos los agentes se pueden equivocar. Nosotros podemos pensar que la válvula 221 esta abierta cuando de hecho esta cerrada; de la misma forma el agente puede tener *abierta(valvula221)* en su base de datos, pero esto no significa que la válvula 221 este abierta. Los censores del agente pueden estar defectuosos, su razonamiento fallar, la información puede estar desactualizada o la fórmula *abierta(valvula221)* puede significar otra cosa para los diseñadores del agente.

Sea L el conjunto de fórmulas de lógica de primer orden, y sea el conjunto $D = \wp(L)$, entonces el estado interno de un agente es un elemento de D. Denotemos Δ, Δ_1, … a los elementos de D. El proceso de toma de decisión de un agente puede ser modelado por un conjunto de reglas de deducción ρ. El cual son solo reglas de inferencias para la lógica. Escribiremos $\Delta \vdash_\rho \varphi$ si la fórmula φ puede ser probada de la base de datos Δ usando solo las reglas de derivación ρ. Luego la función de percepción *see* permanece sin cambios:

$$see: E \rightarrow P$$

y nuestra función de *next* toma la forma:

$$next: D \times P \rightarrow D$$

la cual toma una base de datos y una percepción y devuelve una nueva base de datos. La función *action* queda:

$$action: D \rightarrow A$$

la cual esta definida en términos de las reglas de deducción. Un pseudo código para esta función sería:

```
1. function action(Δ:D) : A
2. begin
3.  for each α ∈ A do
4.    if Δ |--ρ Do(α) then
5.      return α
```

```
6.    end-if
7.  end-for
8.  for each α ∈ A do
9.    if Δ |-/-ρ ¬Do(α) then
10.     return α
11.   end-if
12. end-for
13. return null
14.end function action
```

La idea es que el programador del agente construya las reglas de deducción y la base de datos Δ, de tal forma que si la fórmula $Do(\alpha)$, donde α es una acción, puede ser deducida entonces α es la mejor acción que se puede realizar.

De esta forma el comportamiento del agente esta determinado por las reglas de deducción y de su base de datos actual.

Ejemplo de agentes deductivo: *El mundo de la aspiradora*.

Para ilustrar esta idea, veamos un pequeño ejemplo basado en el mundo de las aspiradoras.

Tenemos un pequeño robot que limpia la casa (ver figura 1). El robot esta equipado con un censor que le indica si hay o no churre y una aspiradora que le permite eliminar el churre. Además el robot tiene definida una orientación (una de: norte, sur, este y oeste). También el agente podrá avanzar un paso hacia delante o rotar 90° grados a la derecha. El agente se moverá por la habitación, la cual esta dividida en celdas del mismo tamaño, correspondiente a la unidad de movimiento del agente. Asumiremos que el agente solo limpiará y nunca saldrá de la habitación. Por solo simplificar el problema supongamos que la habitación esta dividida en 3 x 3 celdas y el agente siempre empezara en el celda (0, 0) dirigido al norte.

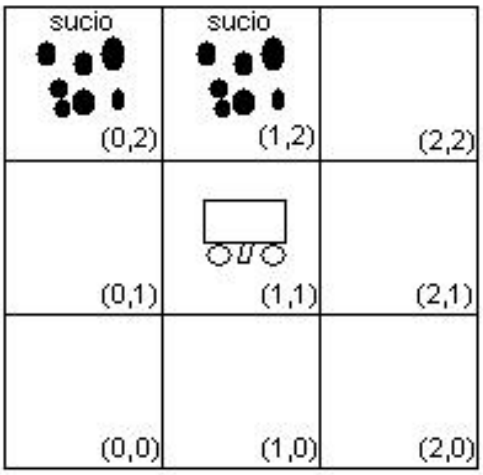


Figura 1. El mundo de la aspiradora.

El objetivo del agente es recorrer la habitación continuamente buscando y limpiando el churre. Primero que todo note que usaremos tres predicados simples en este ejercicio:

in(x, y)	el agente se encuentra en la celda (x, y)
dirt(x, y)	la celda (x, y) esta sucia.
facing(d)	el agente esta orientado hacia la dirección d

Ahora especificaremos la función *next*. Esta función debe tomar una información perceptual obtenida del ambiente (sucio o no sucio) y generar una nueva base de datos la cual incluye esta información. Pero además, deberá quitar toda la información vieja o irrelevante; y también, deberá definir una nueva localización y orientación del agente. Denotemos como *old(Δ)* al conjunto de toda la información vieja en la base de datos, la cual queremos que la función de actualización *next* quite:

$$old(\Delta) = \{ P(t_1, \ldots t_n) / P \in \{in, dirt, facing\} \; y \; P(t_1, \ldots t_n) \in \Delta\}$$

Ahora necesitamos una función *new*, la cual dado un conjunto de predicados los añade a la base de datos, la cual es definida:

$$new: D \; x \; P \rightarrow D$$

La definición de esta función no es difícil pero si un poco larga, por lo que se deja como ejercicio (esta función debe generar los predicados: in(…), con la nueva posición del agente; facing(…), indicando la nueva orientación del agente y dirt(…), si se ha detectado churre en la nueva posición). Entonces dadas las funciones *old* y *new* podemos definir la función *next* como:

$$next(\Delta, \rho) = (\Delta - old(\Delta)) \cup new(\Delta, \rho)$$

Ahora podemos ocuparnos de las reglas que gobernaran el comportamiento del agente. Las reglas de deducción tiene la forma:

$$\psi(\ldots) :- \emptyset(\ldots).$$

Donde Ø y ψ son predicados sobre una lista arbitraria de constantes y variables. La idea es que si Ø unifica contra la base de datos del agente, entonces ψ puede ser deducido.

La primera de las reglas que podemos tener es la regla básica de limpieza, la cual tiene prioridad sobre todas las demás acciones del agente, la cual queda:

$$do(clear) :- in(X, Y), dirt(X,Y). \qquad (1.1)$$

Por lo tanto, si el agente esta en la posición (x, y) y encuentra churre, entonces si acción es limpiarlo. Sino, la acción del agente será atravesar la habitación. Aprovechando la

simplicidad el ambiente, implementaremos el algoritmo de navegación básico: el agente siempre se moverá de (0,0) a (0,1) a (0,2) y después a (1,2) a (1,1) y así sucesivamente. Cuando el agente llegue a (2,2), debe regresar y empezar nuevamente en (0,0). Las reglas para el moviendo hasta (0,2) son muy simples:

do(forward) :- in(0, 0), facing(north), not(dirt(0,0)). (1.2)
do(forward) :- in(0, 1), facing(north), not(dirt(0,1)). (1.3)
do(turn) :- in(0, 2), facing(north), not(dirt(0,2)). (1.4)
do(forward) :- in(0, 2), facing(east). (1.5)

Reglas similares se pueden generar para que el agente viaje hasta (2,2) y una vez alli regresar hasta (0,0). No es difícil ver que estas reglas junto con la función *next*, generaran el comportamiento de nuestro agentes.

En este punto vale la pena volver atrás y examinar el paradigma de la construcción de agentes basados en lógica. Probablemente, el punto más importante, es que tratar construir, literalmente, un agentes usando esta forma, puede ser más o menos impracticable. Veamos porque, supongamos que diseñamos un conjunto de reglas ρ para un agente para una base de datos Δ, si podemos probar *Do(a)* entonces a es la acción optima. Imaginemos que echamos a andar el agente.

En un tiempo t_1, el agente ha generado una base de datos Δ_1 y empieza a aplicar las reglas de deducción para encontrar una acción para realizar. Un tiempo después, t_2, el logra establecer que Δ_1 |-- *Do(a)* para un $a \in A$, por lo que *a* es la mejor acción que el agente podía realizar en t_1; pero el ambiente puede cambiar entre t_1 y t_2, y nada nos garantiza que *a* continué siendo óptima más aun si la diferencia entre t_1 y t_2 es grande. Si t_2-t_1 es infinitesimal, la toma de decisión es instantánea. Pero de hecho, si tenemos el agente usa predicados de lógica de primer orden clásica para representar el ambiente y sus reglas son completas y correctas, entonces no hay garantía de que el proceso termine.

> UN AGENTE SE DICE QUE ES CUMPLE CON LA PROPIEDAD DE CALCULO RACIONAL SI Y SOLO SI SU MECANISMO DE TOMA DE DECISIÓN DA UNA ACCIÓN QUE FUE OPTIMA CUANDO SE COMENZÓ EL PROCESO DE TOMA DE DECISIÓN.

Esta propiedad, claramente, no es aceptable para ambientes que cambien con más rapidez que el agente puede hacer sus decisiones.

Podemos construir agentes que tengan un rendimiento respetable sin usar la representación estrictamente lógica y los conjuntos de reglas de deducción completas; pero perderemos la gran ventaja que este método nos proporciona de: una semántica simple y elegante.

Existen otros problemas asociados a la construcción de agentes basados en lógica. Primeramente, la función *see* del agente traduce los estados del agente en percepciones. En el caso de los agentes basados en lógica, normalmente es un conjunto de fórmulas. Pero en muchos ambientes no es fácil realizar esta conversión. Por ejemplo, transformar

una imagen a un conjunto de expresiones declarativas que representen a la imagen es todavía un problema abierto.

Otro problema es que realmente es que representar la propiedad de dinamismo, es en extremo complicado. Por ejemplo, representar y razonar sobre información temporal: como una situación cambia con el tiempo es extraordinariamente difícil.

Por último, como muestra el ejemplo de la aspiradora, representar procedimientos sobre el conocimiento, incluso simples; en la lógica tradicional puede ser engorroso y no intuitivo.

6.3.3. Ejercicios

Todos los ejercicios que a continuación se relacionan se refieren al ejemplo "El mundo de la aspiradora".

- Dé una definición completa de la función new, la cual define los predicados que hay que añadir a la base de datos del agente.
- Complete las reglas de deducción para el moviendo del agente a través del ambiente.
- Complete todo el ejemplo en PROLOG, utilice assert y retract para lograr que el programa realice las operaciones de la función next.
- Trate de programar el ejemplo en su lenguaje de programación imperativo favorito. ¿Qué piensa usted sobre esta solución en comparación a la solución lógica?
- Suponga que existen obstáculos que el agente debe evitar. Trate de adaptar el ejemplo para que detecte y evite obstáculos. Compárelo con una implementación en un lenguaje imperativo.
- Agrande el ambiente a 10 × 10 celdas. ¿Aproximadamente cuantas reglas se necesitan ahora?
- Trate de generalizar las reglas y reestructure las reglas de deducción.

6.4. Agentes de razonamiento práctico.

6.4.1. Razonamiento práctico = deliberación + planeamiento.

En esta conferencia abordaremos la arquitectura creencia-deseo-intención (BDI). Esta arquitectura tiene sus raíces en la tradición filosófica de la comprensión del *razonamiento práctico*: el proceso de decidir, momento a momento, que acción realizar para lograr nuestros objetivos.

El razonamiento práctico comprende dos procesos importantes: decidir que objetivos queremos lograr y como vamos a lograr estos objetivos. El primer proceso es conocido como deliberación y el segundo como planeamiento. Para comprender el modelo BDI, vale la pena considerar un ejemplo simple. Cuando terminamos la Universidad, nos enfrentamos a la decisión de que hacer en nuestras vidas. El proceso de decisión comienza tratando de obtener que opciones tenemos disponibles. Por ejemplo si se obtuvieron buenas calificaciones, entonces una opción sería convertirse en académico (si no se obtuvieron buenas calificaciones, esta opción no esta disponible). Otra opción sería convertirse en ingeniero. Después de generar este conjunto de alternativas, se debe escoger entre ellas y llevarla a cabo. Estas opciones se convierten entonces en intenciones, las cuales determinan las acciones del agente. Las intenciones, entonces, retroalimentan el razonamiento práctico del agente en el futuro. Por ejemplo, si YO *decido*: YO *quiero* ser un académico, entonces YO *debo* realizar este objetivo y dedicar tiempo y esfuerzos para lograrlo.

6.4.2. Intención en el razonamiento práctico.

Las intenciones juegan un papel crucial en el proceso de razonamiento práctico. Quizás, la propiedad más obvia de las intenciones es que llevan a la acción: si de verdad YO *quiero* convertirme en un académico, entonces se puede esperar de mí que actué en base a esa intención. Por ejemplo, usted podría esperar que yo ponga en práctica varios programas de doctorados. Usted esperaría que yo realizará un grupo de acciones que yo crea van a satisfacer la intención de la mejor manera. Por otra parte, si este grupo de tareas no satisfacen la intención, entonces se puede esperar que lo vuelva a intentar. Por ejemplo, se rechaza mi primer programa de doctorado, entonces se puede esperar que lo intente en otras universidades. Además, una vez que YO asuma una intención, entonces el hecho de tener esa intención obligara mi razonamiento práctico futuro. Por ejemplo, cuando alguien tiene una determinada intención, no se distrae con opciones que son inconsistentes con la intención. Pretender convertirse en un académico, excluye la opción de "fiestar" todas las noches: estas opciones son mutuamente exclusivas.

Además, las intenciones persisten. Si Yo adopto la intención de convertirme en académico, entonces YO persistiré en esa intención y trataré de lograrla. Si no inmediatamente dejo mis intenciones sin dedicar esfuerzo para lograrla, entonces nunca la lograré. Por otro lado, no se debe persistir por mucho tiempo en las intenciones: si se me hace evidente que nunca me convertiré en académico, entonces es racional que deje esa intención. De la misma forma, si la razón por la cual yo tengo una intención ya no existe, entonces es racional dejar la intención. Por ejemplo, si YO adopto la intención de ser académico porque creo que es una vida fácil, pero descubro que lo que se espera realmente es que enseñe, entonces la justificación de la intención ya no existe por lo que YO debo dejar la intención.

Por ultimo, las intenciones están muy relacionadas con las creencias futuras. Por ejemplo, si YO trato de de convertirme en un académico, entonces YO debo creer que de hecho me convertiré en académico. Por otro lado, si realmente yo creo que nunca seré

académico, entonces no me es esencial tener la intención de convertirme en académico. De esta manera, si intento convertirme en académico, YO debo creer que existe una oportunidad de que lo logre.

En resumen:

- Las intenciones dirigen el planeamiento: Si tenemos la intención de ver académicos, entonces trataremos de lograrlo; lo cual involucra, entre otras cosas, decidir como lo vamos a lograr.

- Las intenciones limitan las futuras deliberaciones: Si tratamos de convertirnos en académicos, entonces no nos entretendremos a considerar acciones que son inconsistentes con nuestra intención.

- Las intenciones son persistentes: Normalmente no tendremos interés en lograr una intención sin una buena razón. Las intenciones persistirán mientras creamos que es conveniente lograrla.

- Las intenciones influyen en las creencias sobre las cuales está basado el razonamiento práctico en el futuro: Si adoptamos una intención, entonces podemos planear el futuro sobre la base de que se logrará la intención.

Un problema fundamental en el diseño de agentes de razonamiento práctico, es el de lograr un balance entre estos consensos. Específicamente, es claro ver que un agente puede en determinado momento dejar algunas intenciones, por las razones que vinos anteriormente. Siguiendo esto, vale la pena que el agente se detenga a reconsiderar sus intenciones. Pero, reconsiderar tiene un costo, tanto en tiempo como en recursos. Pero esto nos trae un dilema:

- Un agente que no se detiene el tiempo suficiente para reconsiderar, continuará tratando de lograr intenciones que claramente no podrá alcanzar o no hay ninguna razón para lograrlas.
- Un agente que constantemente reconsidera, no tiene mucho tiempo para trabajar por lograr sus intenciones y corre el riesgo de nunca lograr ninguna.

Estos problemas son n esencia los mismos de balancear los comportamientos pro-activo y el reactivo, vistos en la conferencias #1.

La naturaleza de este problema fue examinado por David Kinny & Michael Georgeff, en varios experimentos usando una arquitectura BDI para agentes. Consideraron como dos tipos de agentes: agentes *atrevidos* (nunca se detienen a reconsiderar) y agentes *precavido* (constantemente están deliberando); actuando en una variedad diferentes de ambientes. El parámetro mas importante de los ambientes para este experimento es el índice de cambio del ambiente, γ. Con los siguientes resultados:

- Si γ es bajo (el ambiente no cambia rápidamente), los agentes *atrevidos* se comportaron mejor que los *precavidos*, debido a que estos últimos gastaban mucho tiempo reconsiderando sus obligaciones, mientras que los primeros trabajaban para lograr sus objetivos.
- Si γ era alto (el ambiente cambia con mucha frecuencia), entonces los agentes *precavidos* funcionaron mejor que los *atrevidos*, debido a fueron capaces de reorganizarse cuando sus intenciones caducaban y tomaban ventajas de las situaciones casuales y las nuevas oportunidades.

Por lo que, para ambientes con diferentes propiedades son necesarias diferentes estrategias de tomas de decisión.

6.4.3. Arquitectura BDI.

El proceso de razonamiento práctico en agentes BDI se resume en la figura 1. Como se aprecia en esta figura existen siete componentes principales:

- un conjunto de creencias actuales: representan la información del agente sobre su ambiente

- una función de revisión de creencias (*brf*), la cual toma una entrada perceptual y el conjunto de creencias actuales del agente y determina un nuevo conjunto de creencias.

- una función de generación de opciones (*options*): la cual determina cuales opciones están disponibles para el agente (sus deseos), sobre la base de creencias sobre su ambiente y sus de intenciones.

- un conjunto de posibles opciones: representando los posibles cursos de acción del agente (planes).

- una función de filtrado (*filter*): la cual representa el proceso de deliberación del agente, la cual determina las intenciones del agente sobre la base de sus creencias, deseos e intenciones actuales.

- un conjunto de intenciones: representan en lo que el agente esta enfocado, los estados en los que esta trabajando para lograr.

- Una función de selección (*execute*): la cual determina que acción debe realizar sobre la base de sus intenciones,

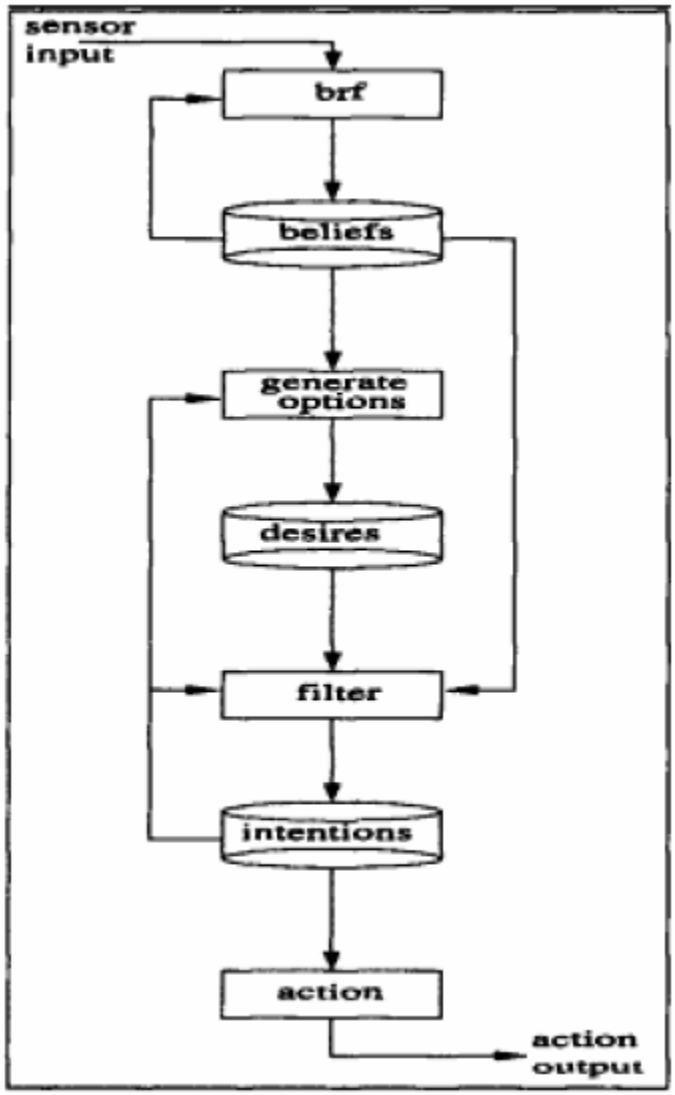


Figure 1.5 Diagrama de una arquitectura BDI genérica

Formalicemos estas componentes.

Sean *Bel* el conjunto de todas las posibles creencias; *Des* el conjunto de todos los posibles deseos e *Int* el conjunto de todas las posibles intenciones. Para el propósito de esta conferencia el contenido de estos conjuntos no nos interesa (aunque, normalmente los elementos de estos conjuntos son fórmulas). Cualesquiera sean los elementos de estos conjuntos, pueden tener definido alguna noción de consistencia; por ejemplo, se puede realizar la siguiente pregunta: es la intención *x* consistente con la creencia *y*.

El estado de un agente BDI es un trio *(B, D, I)*, donde $B \subseteq Bel$, $D \subseteq Des$, $I \subseteq Int$. La función de revisión de creencias queda definida:

$$brf : \wp(Bel) \times P \rightarrow \wp(Bel)$$

la cual dada una percepción y un conjunto de creencias determina un nuevo conjunto de creencias.

La función de generación de opciones, toma un conjunto de creencias y otro de intenciones y de como resultado un conjunto de deseos.

$$opcions : \wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$$

Esta función tiene varios roles. Primero, es responsable del proceso de planificación del agente: el proceso de cómo lograr sus intenciones. Pro lo que, si un agente tiene una intención *x*, este debe considerar opciones para lograrla. Estas opciones pueden ser más o menos abstractas que *x*. Algunas opciones entonces se convertirán a su vez en intenciones, por lo que estas retroalimentan la generación de opciones trayendo como resultado la generación de opciones más concretas. Podemos ver al proceso de generación de opciones de los agentes BDI como una elaboración recursiva de estructuras de planes jerárquicos; considerando y realizando intenciones cada vez más especificas, hasta alcanzar una intención que se corresponda con una acción inmediata.

Aunque el propósito principal de la función de generación de opciones es el planeamiento, debe a su vez satisfacer determinadas restricciones. Primero, debe ser consistente: cualquier opción generada debe ser consistente tanto con las creencias del agente como con sus intenciones actuales.

Segundo, debe ser oportunista: se debe reorganizar cuando el ambiente cambie favorablemente, para ofrecer al agente nuevas formas de lograra sus intenciones o dándole la oportunidad de alcanzar intenciones que en otras circunstancias no serían posible.

El proceso de deliberación es representado por la función *filter*:

$$filter : \wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$$

la cual actualiza las intenciones del agente teniendo en cuenta las intenciones, creencias y deseos actuales. Esta función debe satisfacer varias cosas. La primera es que debe ser eliminada cualquier intención que ya no sea valida o que el costo de lograrla sea mayor que las ganancias que se esperan al lograrla. La segunda, se deben mantener las intenciones no logradas, pero que se espera tengan un beneficio positivo. Finalmente, no puede adoptar una nueva intención para lograr las ya existentes o para explorar nuevas posibilidades. Note que esta función debe satisfacer:

$$\forall B \in Bel,\ \forall D \in Des,\ \forall I\ Int,\ filter(B,D,I) \subseteq I \cup D$$

En otras palabras las intenciones actuales son las que ya se tenían o nuevas opciones. La función de ejecución, se asume que devuelve cualquier intención ejecutable: una que se corresponda directamente con una acción ejecutable.

execute : Ac

La función de acción de un agente BDI es una función: *action* $P \rightarrow A$. Definida por el siguiente pseudo-código:

```
1. function action (p: P): A
2. begin
3.    B = brf (B, p)
4.    D = options (D, I)
5.    I = filter (B, D, I)
6.    return execute (I)
7. end function action
```

Note que representar las intensiones de un agente a través de un conjunto es generalmente muy simplista en la práctica. Una alternativa simple sería asociar prioridades a las intenciones indicando si importancia relativa. Otra idea es representarlas con una pila, una intención se *empuja* en la pila cuando se adopta y se *saca* cuando se logra o cuando no es alcanzable. Las intenciones más abstractas estarán en el fondo de la pila y las más concretas en el tope.

Para resumir; las arquitectura BDI es una arquitectura de razonamiento práctico, en la cual el proceso de decidir que hacer, se asemeja al tipo de razonamiento práctico que aparentemente usamos en las vida. La componente básica de esta arquitectura son estructuras de datos para representar el conocimiento, los deseos y las intenciones del agente y funciones que representan la deliberación (decidir que hacer) y el planeamiento (decidir como se va ha hacer). Las intenciones juegan el papel central en este modelo: brindan estabilidad para la toma de decisiones y actúa como foco del razonamiento del agente. El principal problema de esta arquitectura es balancear el proceso de deliberación debe de estar adaptado al ambiente; asegurando que en ambientes muy dinámicos, el agente reconsidera sus intenciones con frecuencia; en los más estáticos lo hagan con menos frecuencia.

Este modelo es intuitivo, todos reconocemos los procesos de decidir que hacer y como hacerlo y tenemos una noción informal de: creencias, deseos e intenciones. Brinda una descomposición funcional clara, que indica cuales tipos de subsistemas se necesitan para construir un agente. Pero su problema principal es el de cómo saber cuan eficiente se implementan estas funciones.

6.5. Agentes Reactivos.

6.5.1. Agentes Reactivos

Los problemas, aparentemente intratables, de los métodos simbólicos para construir agentes, permitió a algunos investigadores a cuestionarse, y finalmente rechazar; las suposiciones sobre las cuales se basan estos métodos. Estos plantearon que cambios menores a los métodos simbólicos, no eran suficientes para construir agentes que operaran en ambientes con restricciones de tiempo: por lo que era necesario desarrollar nuevos métodos.

A finales de la década de los 80, se empezaron a investigar alternativas al paradigma de la IA simbólica. Es difícil caracterizar limpiamente estos diferentes métodos, ya que sus defensores se apoyan principalmente en el rechazo a la IA simbólica. Sin embargo, determinados temas se repiten:

- El rechazo a la representación simbólica y a la toma de decisión basada en la manipulación de estas representaciones.

- La idea de que la inteligencia y el comportamiento racional esta naturalmente unido al ambiente que el agente ocupa: es el resultado de la interacción que el agente mantiene con su ambiente.

- La idea de que el comportamiento inteligente emerge de la interacción de varios comportamientos simples.

6.5.2. Arquitectura de Brooks para agentes reactivos: Arquitectura de categorización

A estos modelos alternativos se les llama de varias formas, una de esta es *reactivo*, debido a que estos sistemas reaccionan al los cambios del ambiente en cuanto los perciben. En esta conferencia examinaremos la "*arquitectura de inclusión*", la cual es hasta ahora la mejor arquitectura para agentes reactivos. Esta arquitectura fue desarrollada por Rodney Brooks, uno de los críticos más influyentes de los métodos simbólicos para agentes.

Esta arquitectura esta definida por dos características. La primera es que la toma de decisión del agente se realiza a través de un conjunto comportamientos para lograr objetivos; cada uno de estos comportamientos puede ser una función de acción tal como la vimos anteriormente, la cual toma constantemente entradas preceptúales y elige una acción a realizar. Cada uno de estos módulos de comportamiento trata de lograr un objetivo en particular.

En la implementación de Brook, los módulos de comportamiento son máquinas de estados finitos. Debe quedar claro que estos módulos no incluyen representación simbólica compleja y se asume que no hay razonamiento simbólico. En muchas implementaciones, estos comportamientos son implementados por reglas de la forma:

$$\text{situación} \rightarrow \text{acción}$$

la cual selecciona una acción directamente de una entrada perceptual.

La segunda característica de esta arquitectura es que varios de estos comportamientos pueden "dispararse" simultáneamente, por lo que debe de haber un mecanismo para escoger entre todas estas acciones. Brook propone ordenar estos módulos en una jerarquía por categorías, con los comportamientos ordenados por capas. Las capas inferiores de la jerarquía inhiben a las capas más altas: mientras más abajo este una capa, más alta será su prioridad. La idea es que las capas más altas representen comportamientos más abstractos. Por ejemplo, en un robot móvil el comportamiento "esquivar un obstáculo" se le debe una prioridad alta, por lo que normalmente se codifica en la capa más baja.

Para ilustrar la *arquitectura de categorización* con más detalle, presentaremos a continuación un modelo formal de este, e ilustraremos que como funciona por medio de un pequeño ejemplo. Y por último discutiremos sus ventajas relativas así como sus desventajas.

La función *see*, la cual representa la habilidad perceptual del agente se mantiene como antes:

$$see: E \rightarrow P$$

Sin embargo; en las implementaciones de esta arquitectura, las entradas preceptúales no se procesan o transforman mucho.

La función de toma de decisión (*action*) se define a través de un conjunto de conductas, unido con una relación de inhibición entre estas conductas. Una conducta es un par $<c, a>$, donde $c \subseteq P$ es un conjunto de percepciones llamadas condiciones y $a \in Ac$ es una acción. Una conducta $<c, a>$ se "*dispara*" cuando el ambiente esta en el estado $e \in E$, si y solo si $see(e) \in c$. Sea $Beh = \{(c, a) / c \subseteq P \text{ y } a \subseteq Ac\}$ el conjunto de todas las conductas.

Asociado al conjunto de reglas de conducta de un agente $R \subseteq Beh$ se encuentra una relación binaria de inhibición $:<: R \times R$. Esta relación debe ser un orden total estricto en R (reflexiva, antisimétrico y transitiva). Escribiremos $b_1 < b_2$, si $< b_1, b_2> \in <$ y lo leeremos como "b_1 inhibe a b_2"; por lo que b_1 es menor en la jerarquía que b_2 y b_1 tiene mayor prioridad que b_2. Entonces la función *action* se define como:

```
1. function action(p : P) : A
2. var fired : ℘(R)
3. var selected : A
4. begin
5. fire = {(c, a) | (c, a) ∈ R y p ∈ c}
6. for each (c, a)∈ fired do
7.     if ¬(∃(c′, a′) ∈ fired such that (c′, a′) < (c, a) then
```

```
8.       return a
9.     end-if
10. end-for
11. return null
12. end function action
```

Por lo que, la función *action* comienza por calcular el conjunto de todas las conductas que se disparan (5). Entonces, para cada conducta *(c, a)* se chequea que no haya otra de mayor prioridad que se dispare. Si no, entonces la parte acción de la conducta, *a*, es la acción que se da como resultado (8). Si no se dispara ninguna de las conductas entonces se dispara la acción *null*, que indica que no se realizará ninguna acción.

Dado esto, uno de los principales problema con la toma de decisiones basada en lógica era su complejidad teórica, ahora examinaremos cuan bueno es el desempeño de este sistema basado en conductas. El tiempo promedio de la función *action* en esta arquitectura no es peor que $O(n^2)$ donde n es el número de percepciones. Por lo que, con el sencillo algoritmo expuesto anteriormente la toma de decisiones es tratable. En la práctica, se puede considerar podemos hacer esto de una forma mucho mejor que esto: la toma de decisión se puede codificar en hardware dando un tiempo de cómputo constante; con la tecnología actual esto quiere decir que podemos garantizar la selección de una acción en nanosegundos. Quizás, más que otra cosa, esta simplicidad en el cómputo es el poder principal de la arquitectura de categorización.

Ejemplo: Experimento de de Steel para explorar Martes

A continuación veremos como fueron construidos agentes con la arquitectura de categorización para el siguiente problema

> EL OBJETIVO ES EXPLORAR UN PLANETA DISTANTE, MÁS CONCRETAMENTE, RECOLECTAR UN TIPO PARTICULAR DE PIEDRAS. EL LUGAR DONDE SE ENCUENTRAN LAS PIEDRAS NO SE CONOCE DE ANTEMANO, PERO PRESENTAN UN CIERTO TIPO DE MANCHAS. UN NÚMERO DE VEHÍCULOS AUTÓNOMOS DISPONIBLES PARA RECORRER EL PLANETA RECOLECTANDO LOS FRAGMENTOS DE PIEDRAS Y DESPUÉS LLEVARLOS A LA NAVE NODRIZA Y QUE ESTA LOS TRAIGA A LA TIERRA. NO HAY MAPA DISPONIBLE DEL PLANETA, PERO SE CONOCE QUE EL TERRENO ESTA LLENO DE OBSTÁCULOS, QUE IMPIDE QUE LOS VEHÍCULOS INTERCAMBIEN INFORMACIÓN

El problema al que nos enfrentamos es entonces con el de construir una arquitectura de control de agente para cada vehículo, por lo que deben cooperar en la recolección de los fragmentos de piedra, tan eficiente como se pueda. Luc Steel planteo que, el uso de agentes basados en lógica descritos anteriormente era "enteramente irreal" en este problema. Por lo que propuso una solución usando la arquitectura de categorización. Para la solución de este problema, Steel, introdujo dos mecanismos:

El primero es un campo de gradiente: Para que los agentes sepan en que dirección es la nave nodriza, esta genera una señal de radio. Esta señal, obviamente se debilita a la distancia y se incrementa cuando se esta cerca: para encontrar la nave nodriza, un agente necesita solamente viajar subiendo por el gradiente de la fuerza de la señal. La señal no envía ninguna información solo existe.

El segundo mecanismo permite a los agentes comunicarse unos con otros. Las características del terreno impiden la comunicación directa, por lo que Steel adopto un método de comunicación indirecto. La idea es que los agentes "migajas radio activas", las cuales pueden ser tiradas, recogidas y detectadas por robots que estén cerca de ellas. Por lo que, si un agente deja caer una de estas migajas en un determinado lugar, entonces después otro agente que pase por ese lugar podrá detectarla. Este mecanismo tan simple posibilita una forma algo sofisticada de cooperación.

El comportamiento de un agente es construido entonces a partir de un número de conductas como las vista anteriormente. Primero, veremos como puede ser programado un agente para que recolecte fragmentos de forma individual, y después veremos como pueden ser programados para que generar una solución cooperativa.

Para agentes individuales, la conducta de menor nivel (y esta es la conducta de mayor prioridad) es la de esquivar los obstáculos. Esta conducta puede ser representada con la regla:

if *detect an obstacle* **then** *change direction* (1.1)

La segunda conducta asegura que cualquier fragmento recolectado por el agente será llevado a la nave nodriza:

if *carrying samples* and *at the base* **then** *drop samples* (1.2)
if *carrying samples* and *not at the base* **then** *travel up gradient*. (1.3)

al conducta (1.3) asegura que los fragmentos recolectados serán llevados a la nave nodriza. La próxima conducta asegura que los agentes recolectarán los fragmentos que encuentren.

if *detect a sample* **then** *pick sample up*. (1.4)

La última conducta asegura que un agente que no tiene nada que hacer explore aleatoriamente.

if *true* **then** *move randomly* (1.5)

La precondición de esta conducta siempre se dispara. Las conductas esta ordenadas entonces:

$$(1.1) < (1.2) < (1.3) < (1.4) < (1.5)$$

La jerarquía de este ejemplo asegura que; por ejemplo, un agente siempre se virara si detecta un obstáculo; si se encuentra en la nave nodriza y trae algún fragmento de piedra, entonces siempre los dejara caer si no esta en peligro inminente de choque y así sucesivamente. La conducta del nivel más alto, siempre se ejecutará si el agente no tiene

nada más urgente que hacer. No es difícil notar que este conjunto de conductas simples resolverán el problema: los agentes buscaran los fragmentos de piedra y al encontrarlos los llevarán a la nave nodriza.

Si los fragmentos están distribuidos por el terreno de forma completamente aleatoria, entonces un equipo de numerosos robots con este comportamiento trabajara extremadamente bien. Pero sabemos por las especificaciones del problema, que este no es el caso: los fragmentos tienden a estar localizados en grupos. En este caso, tiene sentido que los agente cooperen unos con otros para encontrar los fragmentos. Entonces si un agente encuentra una muestra grande, seria de muy conveniente comunicárselo a los otros para que lo ayuden a recolectar las piedras.

Desafortunadamente, sabemos que la comunicación directa no es posible. Steel desarrollo una solución muy simple para este problema, inspirado en el comportamiento de las hormigas. La idea es que el agente cree un rastro de migajas si encuentra una muestra de piedras. El rastro es creado cuando el agente regresa a la nave nodriza. Si en algún momento posterior, otro agente se encuentra con este rastro, entonces solo necesitara seguir el gradiente, para localizar la fuente de las rocas. Algunos pequeños arreglos mejoran mucho más este ingenioso diseño. Primero; cuando un agente siga un rastro, recogerá algunas migajas, de esta forma lo debilitará. Segundo, solo los agentes que regresan de la nave nodriza podrán es fortalecido por agentes que regresen a la nave nodriza. De esta forma; si un agente sigue el rastro hasta la supuesta fuente de rocas y no las encuentra, entonces habrá debilitado el rastro y no regresará con rocas para reforzarlo. Pasado varios agentes que sigan un rastro que no tenga rocas al final, el rastro será borrado.

Las modificaciones de las conductas para este ejemplo son como sigue. La conducta (1.1) se mantiene sin cambios. Pero, las dos reglas que definen que hacer cuando se carga una muestra se modifican de la siguiente forma:

if *carrying samples* and *at the base* **then** *drop samples* (1.6)
if *carrying samples* and *not at the base* **then** *drop 2 crumbs* and *travel up gradient*. (1.7)

La conducta (1.7) asegura que el agente arroje una migaja cuando este de vuelta a la base con una muestra, de esta forma crea el rastro o lo refuerza. La conducta (1.4) se mantiene como antes. Pero una conducta adicional es necesaria para lidiar con las migajas.

if *sense crumbs* **then** *pick up 1 crumb* and *travel down gradient* (1.8)

Finalmente, la conducta de movimiento aleatorio (1.5) se mantiene como antes. Estas conductas se ordenan:

$$(1.1) < (1.6) < (1.7) < (1.4) < (1.8) < (1.5)$$

Steel demostró que este simple ajuste logra un rendimiento próximo al óptimo en muchas situaciones. Mucho más; la solución es económica (el poder de computo que necesita

cada agente es mínimo) y robusta (la perdida de un agente no afecta significativamente el funcionamiento general del sistema).

6.5.3. Limitaciones de los agentes reactivos

En resumen, los métodos reactivos, como la arquitectura de Brooks, tienen ventajas obvias: simples, económicas, computacionalmente tratables, robustas y elegantes. Pero hay problemas fundamentales sin solución en estas arquitecturas reactivas.

- Si el agente no emplea ningún modelo del ambiente, entonces no tendrán la información suficiente disponible de si ambiente para determinar una acción aceptable.

- Debido a que estos agentes basan su toma de decisión sobre información local, es difícil ver como estos mecanismos de toda de decisiones pueden tomar en cuenta la información no local, esto los una toma de decisión inherentemente a corto plazo.

- Es difícil ver como un agente puramente reactivo puede aprender de sus experiencias y mejorar su rendimiento con el tiempo.

- Aunque agentes efectivos pueden ser generados por un número pequeño de conductas (normalmente menos de 10 capas), es muy complicado construir agentes con muchas capas. La dinámica de interacción entre conductas distintas se vuelve muy complicado.

- El comportamiento general de los sistemas puramente reactivos emerge de la interacción de las conductas y el ambiente. Pero el término surge sugiere que la relación entre conductas individuales, ambiente y el comportamiento general no es comprensible. Esto hace que construir agentes para cumplir con tareas específicas sea muy difícil. Finalmente, no existe una metodología para construir tales agentes: se debe usar un proceso de experimentación (prueba y fallo) para diseñar un agente.

6.6. Arquitecturas por capas para agentes.

6.6.1. Arquitecturas por capas

Debido a que los agentes deben ser capaces de de mostrar un comportamiento reactivo y proactivo, una descomposición evidente es crear dos subsistemas para tratar con estos dos tipos de conductas diferentes. La idea esta dirigida naturalmente a una clase de

arquitecturas en la cual varios subsistemas son organizados dentro de una jerarquía de capas interactuantes.

En esta conferencia consideraremos varios aspectos generales de estas arquitecturas y presentaremos dos ejemplos de tales arquitecturas: INTERRAP y TOURINGMACHINES.

Normalmente, en este tipo de arquitectura tendremos al menos dos capas, una para el comportamiento reactivo y otra para el comportamiento proactivo. En principio no hay ninguna razón para que no existan otras capas. Una topología para estas arquitecturas esta dada por la información y el control del flujo dentro de ellas. Se pueden identificar dos tipos de control de flujo en las arquitecturas por capas (ver Figura 1).

- Capas horizontales: En esta topología (figura 1a), cada capas esta conectada directamente al censor de entrada y a la función de acción. De esta forma, cada capa actúa como un agente, produciendo sugerencias sobre que acción se debe realizar.

- Capas verticales: En las arquitecturas de capas verticales (Figura 1b y 1c), el censor de entrada y la salida de la función acción tratan con al menos con una capa cada uno.

La gran ventaja de estas las arquitecturas horizontales es su simplicidad conceptual: si necesitamos un agente que muestre *n* comportamientos diferentes, entonces debemos implementar *n* capas distintas. Pero, debido a que las capas compiten unas con otras para generar acciones, se corre el riego que el comportamiento del agente sea incoherente.

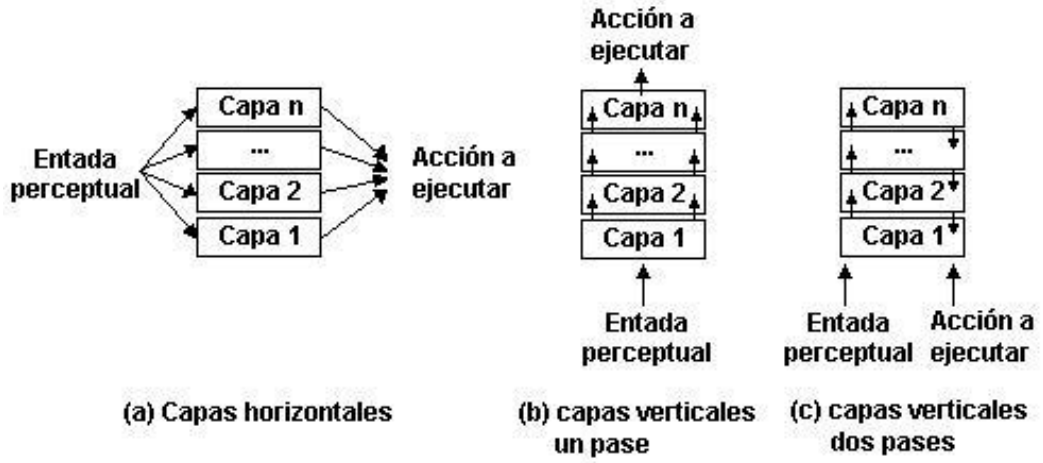


Figure 1: Control del flujo y la información en tres tipos de arquitecturas de capas para agentes.

Para asegurar que en las arquitecturas horizontales sean consistentes, se les incluye generalmente una función *mediadora*, la cual toma la decisión de cual capa tiene el control del agente en un determinado momento. La necesidad de este control a nivel central es problemático: esto quiere decir que el programador debe tener en cuenta todas las posibles interacciones entre las capas. Si hay *n* capas en la arquitectura y cada capa es capaz de sugerir *m* acciones, entonces hay m^n interacciones a considerar. Y esto resulta claramente difícil desde el punto de vista del diseño hasta en los sistemas más simples.

Este sistema de control central también introduce un embotellamiento dentro de la toma de decisión del agente. Este problema es aliviado parcialmente en las arquitecturas de capas verticales. Podemos subdividir las arquitecturas de capas verticales en: arquitecturas de un pase (figura 1b) y arquitecturas de dos pases (figura 1c). En las primeras el control del flujo es secuencial a través de cada capa, hasta la última capa que genera la acción a ejecutar.

En las de dos pases, el flujo de información es hacia arriba en la arquitectura (el primer pase) y después fluye hacia abajo. Existen algunas similitudes interesantes entre la idea de las arquitecturas de capas verticales y la forma de de organización del trabajo, con información que fluye hacia los niveles más altos de la organización y entonces las directivas arriba y bajan.

En ambos arquitecturas verticales, la complejidad de las interacciones entre las capas se reducen: son n-1 conexiones entre *n* las capas, y si cada capa es capas de sugerir *m* acciones entonces hay al menos $m^2(n-1)$ que se deben considerar. Esto es claramente menos que en las arquitecturas horizontales. Sin embargo, esta simplicidad ataca el costo de alguna flexibilidad: debido a que en las arquitecturas de capas verticales el control debe pasar entre cada capa diferente, no hay tolerancia a fallos: si una capa falla esto trae graves consecuencias en el rendimiento del agente.

En el resto de esta conferencia, consideraremos dos ejemplos de arquitecturas por capas: la TOURINGMACHINES de Innes Ferguson y la INTERRAP de Jörg Müller's. La primera es un ejemplo de una arquitectura de capas horizontales; la segunda es una arquitectura de capas verticales de dos pases.

6.6.2. Touring Machines

La arquitectura de la TOURINGMACHINES se ilustra en la figura 2. Como se puede ver en esta figura, esta arquitectura consistes de tres *capas de producción activa*: cada capa continuamente produce sugerencias sobre que acción debe realizar el agente. La *capa reactiva*, suministra respuestas casi inmediatas a los cambios que ocurren en el ambiente y está implementada con un conjunto de reglas de la forma: situación-acción; como en la arquitectura de categorización de Brook (conferencia #5).

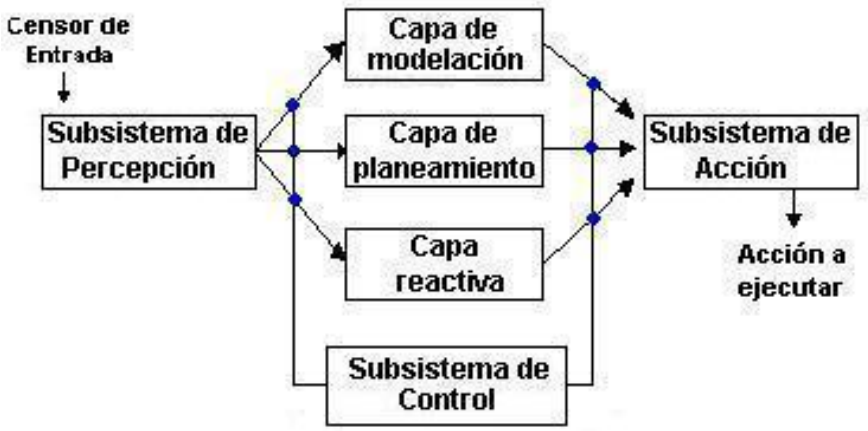


Figure 2. TOURING MACHINES: una arquitectura de capas horizontales para agentes.

La demostración original de la TOURINGMACHINES se realizó con vehículos autónomos que viajaban entre dos puntos a través de calles pobladas por agentes similares. En este ambiente las reglas reactivas normalmente se usan para evadir obstáculos. Por ejemplo, una regla para evadir el contén de la acera:

regla #1: evitar-contén

if (*is_in_front(Kerb, Observer)* and *speed(Observer)>0* and
separation(Kerb,Observer)<KerbThreshHold)
then *change-orientation(Kerb Avoidance Angle)*

donde *change-orientation(...)*, es la acción sugerida si la regla se dispara. Las reglas solo pueden hacer referencia al estado actual del agente y no puede realizar ningún razonamiento sobre su ambiente, por otro lado las acciones no son predicados. Por lo que, si una regla se dispara, el resultado no será actualizar el modelo que se tenga del ambiente, sino que solo se sugerirá realizar una acción por la capa reactiva.

Con la capa de planeamiento de la arquitectura, se consigue el comportamiento pro-activo del agente. Específicamente, esta capa es responsable de la conducta "diaria" del agente: bajo condiciones normales esta capa no es responsable de decidir que hace el agente. Esta capa emplea una librería de esquemas, las cuales son en esencia planes estructurados jerárquicamente que la capa confecciona en tiempo de ejecución para decidir que hacer. Entonces, para lograr un objetivo, la capa de planificación trata de encontrar un esquema el cual corresponda al objetivo. Estos esquemas pueden contener subobjetivos, que la capa elabora tratando de encontrar otros esquemas que correspondan con estos subobjetivos.

La capa de modelación representa las diferentes entidades del ambiente (incluyendo al agente, así como a otros agentes). Esta capa predice posibles conflictos entre los agentes y genera nuevos objetivos para resolver estos conflictos. Estos nuevos objetivos son pasados a la capa de planificación, la cual debe usar su librería para determinar como se deben realizar.

Estas tres capas de control están dentro del subsistema de control, el cual es el responsable de decidir que capa debe tener el control del agente. Este subsistema de control es implementado como conjunto de reglas de control. Las reglas de control pueden suprimir información entre las reglas de y las capas de control ó suprimir acciones de las capas de control. Un ejemplo de estas reglas:

```
regla de control #1:
```

if *entity(obstacle-6) in perception-buffer*
then *remove-sensory-record(layer-R, entity (obstacle-6))*

Esta regla impide a la capa reactiva conozca si el obstáculo-6 ha sido percibido. La idea es que aunque la capa reactiva se generalmente la encargada de la evadir de obstáculos, pueden existir determinados obstáculos para los cuales otras capa son más apropiadas. Esta regla asegura que la capa reactiva nunca conozca sobre estos objetos.

6.6.3. InterRaP

INTERRAP es un ejemplo de una arquitectura de capas verticales de dos pases para agentes, figura 3.

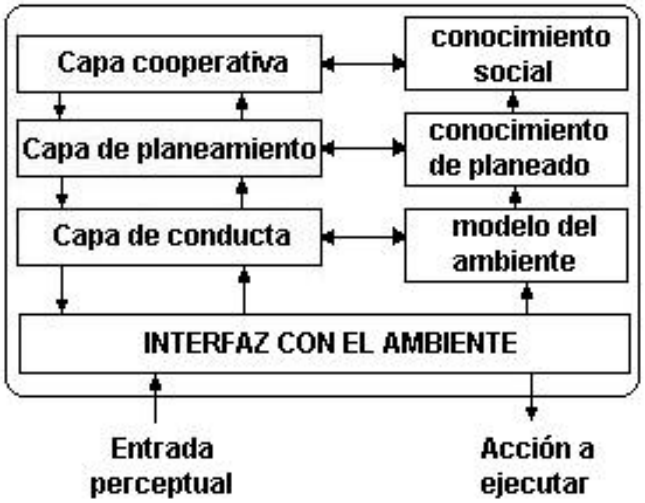


Figure 3. InterRAP: una arquitectura de capas vertical de dos pases para agentes.

Como se ve en la figura 3, INTERRAP tiene es capas de control, como en la TOURING MACHINES. Más aún, el propósito de cada capa en INTERRAP tiene igual propósito que su correspondiente en la TOURINGMACHINES. La capa de conducta, se encarga del comportamiento reactivo; la capa de planificación; es, nuevamente, la responsable del planeamiento diario para lograr los objetivos del agente y la capa cooperativa se encarga de las interacciones sociales.

En esta arquitectura cada tiene asociada una base de conocimiento. Estas diferentes bases de conocimiento representan el agente y su ambiente a diferentes niveles de abstracción.

Por lo que, la base de conocimiento de mayor nivel, representa los planes y acciones de otros agentes en el ambiente; la base de conocimiento intermedia, representa los planes y las acciones del agente mismo y la más baja, representa la información "*cruda*" sobre el ambiente. La introducción implícita de estas bases de conocimiento es lo que distingue a la INTERRAP de la TOURINGMACHINES.

La forma en que las capas de INTERRAP se interrelacionan para producir el comportamiento del agente, también es muy diferente que la de TOURINGMACHINES. La diferencia principal es la forma en que las capas interaccionan con el medio. En la TOURINGMACHINES, cada capa estaba conectada directamente al censor de entrada y a la salida, por lo que era necesario un sistema de control, para resolver conflictos o problemas entre las capas.

En INTERRAP, las capas interactúan unas con otras para lograr el mismo objetivo. Las interacciones principales entre las capas son: activación *bottom-up* y ejecución *top-down*. La activación *bottom-up* ocurre cuando una capa pasa el control la capa superior debido a que ella no es competente con la situación actual. La ejecución *top-down* ocurre cuando una capa superior hace uso de las facilidades que ofrece la capa inferior para lograr uno de sus objetivos.

El flujo de control en la arquitectura INTERRAP comienza cuando se recibe una entrada perceptual en la capa más baja de la arquitectura. Si la capa reactiva es capaz de responder a esta percepción entonces lo hará; sino ocurre una activación *bottom-up*, pasando el control a la capa de planeamiento. Si esta capa puede manejar esta situación, entonces lo hará, normalmente haciendo uso de una ejecución *top-down*. De lo contrario, usará una activación *bottom-up* para pasar el control a la capa superior. De esta forma, el flujo de control en INTERRAP pasa de las capas más bajas a las más altas y después regresa nuevamente.

La forma interna de estas capas no es objetivo de esta conferencia. Sin embargo, vale la pena notar que cada capa debe implementar dos funciones generales. La primera es la función de *reconocimiento de situaciones y activación de objetivos*. Esta función actúa como la función *option* en la arquitectura BDI, la cual toma por entrada una base de conocimientos y los objetivos actuales y da como resultado un nuevo conjunto de objetivos.

La segunda función es responsable del *planeamiento y programación*: es responsable la responsable de seleccionar el plan a ejecutar basado en los planes objetivos y base de conocimiento actuales de la capa.

Las arquitecturas por capas son actualmente la clase más popular de arquitecturas disponible. Las capas representan una descomposición natural de las funcionalidades: es fácil ver como el comportamiento reactivo, pro-activo y social es generado por las capas correspondientes.

El principal problema de las arquitecturas es que aunque es una solución pragmática, se pierde la claridad conceptual y semántica de los métodos que no usan capas. En partícula, mientras que los métodos basados en lógica tienen una semántica clara, es difícil ver como esta semántica puede ser diseñada para las arquitecturas de capas.

Capítulo 7: Inteligencia Artificial Distribuida, Sistemas multiagentes, Comunicación e Interacción

7.1. Inteligencia Artificial Distribuida

La Inteligencia Artificial Distribuida (IAD) analiza las formas en que un grupo descentralizado de entidades computacionales, coordinan sus actividades para lograr la solución de un problema inherentemente distribuido, es decir, un problema en que los recursos (bases de datos, bases de conocimiento, experiencia y capacidad de procesamiento) para resolverlo están distribuidos en el espacio cibernético. El objetivo principal de la IAD es crear arquitecturas que generen y/o interconecten diferentes sistemas de bases de datos (de conocimiento), siendo fundamental para la realización de estas arquitecturas la creación de programas en general denominados agentes encargados de realizar las tareas de enlace y utilización adecuada de estos recursos, estableciendo entre ellos las estrategias de coordinación que optimicen el objetivo común. Por lo tanto, el uso de tales sistemas es apropiado en problemas en los que:

- La experiencia está distribuida.
- La información está distribuida.
- Las decisiones están distribuidas.
- Las bases de conocimiento han sido desarrolladas por separado, pero pueden ser reusables y deben interconectarse para la solución de problemas que escapan a la capacidad individual de cada una por separado.

7.2. Sistema multiagentes

Algunos problemas enfrentados por los agentes son inherentemente distribuidos debido a que los recursos para resolverlos (datos, conocimiento, especialización, etc.) están espacial o temporalmente distribuidos.

Aunque teóricamente un solo agente puede enfrentar problemas de este tipo, en la práctica, como sucede entre los agentes humanos, un conjunto de agentes actuando de manera coordinada enfrentan estos problemas de manera más efectiva.

De aquí surge la idea de organizar a los agentes de forma tal que contribuyan, con las capacidades y recursos a su alcance, a la solución de problemas que exigen su concurso coordinado. Esta forma de organización de los agentes es lo que se conoce como sistema multiagentes (SMA), los cuales podrían definirse de la manera siguiente:

Un SMA es un sistema constituido por un conjunto de agentes, cada uno con determinadas capacidades y recursos computacionales para la solución de una clase de problemas o tareas. Siendo el objetivo de tales sistemas la solución de problemas inherentemente distribuidos, estos agentes deben interactuar entre ellos y determinar y coordinar tareas (subproblemas) a realizar que conduzcan a la solución de tales problemas, en la medida en que sus capacidades y recursos puedan contribuir a su solución.

En esta definición se hace referencia a dos problemas fundamentales dentro de este campo: la interacción y coordinación entre los agentes, los cuales pasamos a caracterizar.

7.2.1. Interacción

La razón principal para la interacción entre los agentes es la de coordinar las tareas que se determinan han de contribuir a la solución de un problema dado. Para facilitar la coordinación, los agentes a menudo necesitan comunicar sus intenciones, objetivos y resultados.

La forma general de interacción entre agentes es la comunicación. Una forma frecuente de comunicación es el intercambio de mensajes entre los agentes utilizando un protocolo de comunicación convenido.

7.2.2. Coordinación

La coordinación es una propiedad de un sistema de agentes que realiza alguna actividad en un ambiente compartido. El grado de coordinación es hasta que punto ellos evitan las actividades extrañas reduciendo la disputa del recurso y evitando el deadlock. Cooperación es la coordinación entre agentes no antagónicos, mientras que la negociación es la coordinación entre agentes competitivos o entre agentes con intereses

propios. Normalmente, para cooperar los agentes necesitan un modelo de los otros agentes y también un modelos de futuras interacciones. Coherencia es el grado en que el sistema se comporta como una unidad.

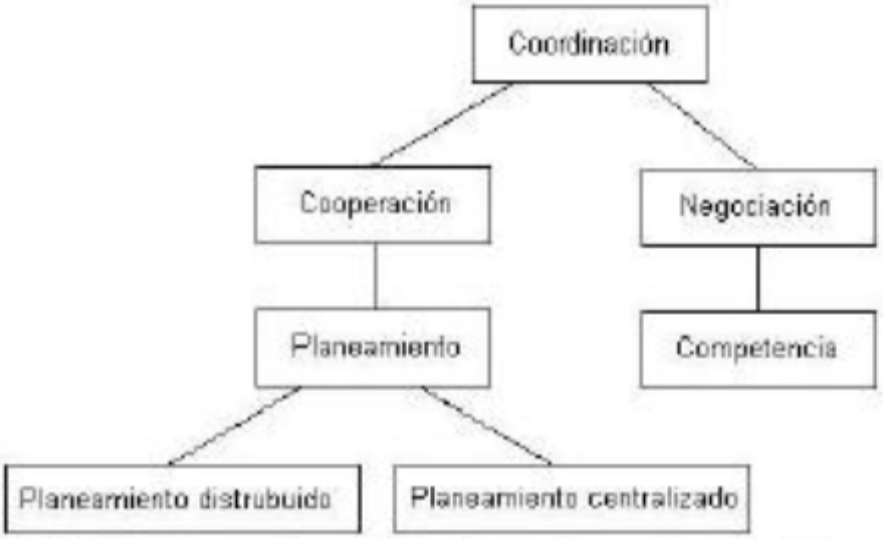


Figura #1: Distintas formas en que los agentes pueden coordinar su comportamiento y actividades.

Un problema en los sistemas multiagentes es como mantener la coherencia global sin un control global explícito. En estos casos los agentes deben ser capaces de determinar cuales objetivos el comparte con otros agentes, determinar tareas comunes, tratar de impedir conflictos innecesarios y combinar sus conocimientos. También los compromisos sociales puede ser un medio para lograr la coherencia.

7.2.3. Comunicación entre agentes

Los agentes se comunican con el objetivo de lograr de una forma más conveniente sus objetivos o los objetivos del sistema en el cual ellos se encuentran. La comunicación puede permitir que los agentes coordinen sus acciones y su comportamiento logrando así sistemas con un comportamiento más coherente.

Dimensiones del significado

Hay tres aspectos en el estudio formal de comunicación:

Sintaxis: Como los símbolos de la comunicación son estructurados.
Semántica: Que denotan los símbolos.
Pragmatismo: Como son interpretados los símbolos.

Significado es la combinación de semántica y pragmatismo. Los agentes se comunican para ser entender y ser entendidos, por lo que es importante considerar las diferentes dimensiones del significado asociadas con la comunicación.

Descriptivo vs. Prescriptivo: Algunos mensajes describen fenómenos, mientras que otros prescriben comportamiento. La descripción es importante para la comprensión humana pero es difícil de imitar por los agentes. Por esto los lenguajes de comunicación de agentes están diseñados para el intercambio de información sobre actividades y comportamientos.

Significado convencional vs. Personal: Un agente puede tener su significado propios para un mensaje, pero puede ser diferente del significado convencional aceptado por otros agentes con los cuales el se comunica. Tanto como sea posible, los sistemas multiagentes deben optar por significados convencionales, especialmente debido a que estos sistemas son normalmente ambientes abiertos en los cuales se pueden introducir nuevos agentes en cualquier momento.

Significado Subjetivo vs. Objetivo: De manera similar al significado convencional, donde el significado es determinado externamente a un agente, un mensaje en ocasiones tiene un efecto explícito que puede ser percibido objetivamente. El efecto puede ser diferente al entendido internamente.

Perspectiva del locutor vs. del que escucha vs. de la sociedad: Independientemente del significado convencional o objetivo de un mensaje; el mensaje puede ser expresado de acuerdo al punto de vista del que habla del que escucha o de otros observadores.

Semántica vs. Pragmatismo: El pragmatismo de una conversación concierne con como los interlocutores usan la comunicación. Esto incluye consideraciones de estados mentales de los interlocutores y el ambiente donde ellos se encuentran: consideraciones que son externas a la sintaxis y semántica de la comunicación.

Alcance: Lenguajes más pequeños son más manejables, pero deben ser lo suficientemente amplios para transmitir el significado de las intenciones del agente.

Identidad: Cuando ocurre una comunicación entre agentes, el significado depende de la identidad y los roles de los agentes involucrados y como son especificados los agentes involucrados. Un mensaje puede ser enviado a un agente en particular o a cualquier agente que cumpla con determinado criterio.

Cardinalidad: Un mensaje enviado a un agente puede ser interpretado de forma diferente que el mismo mensaje transmitido públicamente.

Tipos de mensajes

Es importante para agentes de capacidades diferentes poder comunicarse. Para ser de interés a otros, los agentes deben poder participar en un diálogo. Su papel en estos diálogos puede ser: activo, pasivo, o ambos; permitiéndoles funcionar como: amo, esclavo, o como similares, respectivamente.

Siguiendo la definición anterior sobre un agente, asumimos que un agente puede enviar y puede recibir los mensajes a través de una red de comunicación. Los mensajes pueden ser de varios tipos, como veremos a continuación.

Hay dos tipos básicos de mensajes: afirmaciones y preguntas. Cada agente, ya sea activo o pasivo; debe tener la habilidad de aceptar información. En su forma más simple, esta información se le comunica al agente por una fuente externa por medio de una afirmación. Para asumir un papel pasivo en un diálogo, un agente debe se capas de contestar las preguntas, es decir, debe ser capaz de 1) aceptar una pregunta de una fuente externa y 2) enviar una respuesta a la fuente haciendo una afirmación. Note que desde el punto de vista de la red de comunicación, no hay ninguna distinción entre una afirmación no solicitada y una afirmación realizada en respuesta a una pregunta.

Para asumir un papel activo en un diálogo, un agente debe ser capaz de emitir preguntas y hacer afirmaciones. Con estas capacidades, el agente puede entonces potencialmente controlar a otro agente haciéndolos responder a la pregunta o aceptar la información afirmada. Este medio de control puede extenderse al control de subagentes tales como redes neuronales y bases de datos.

Un agente que funciona como similar de otro agente, puede asumir papeles activos y pasivos en un diálogo. Debe poder hacer y aceptar tanto afirmaciones como preguntas.

Un resumen de las capacidades necesarias por las clases diferentes de agentes se muestra en siguiente tabla.

	Agentes pasivos	Agentes activos	Agentes similares
Recibir afirmaciones	x	x	x
Recibir preguntas	x		x
Enviar afirmaciones	x	x	x
Enviar preguntas		x	x

Otros tipos de mensajes.

	Agentes pasivos	Agentes activos	Agentes similares
Recibir afirmaciones	x	x	x
Recibir preguntas	x		x
Enviar afirmaciones	x	x	x
Enviar preguntas		x	x

Niveles de comunicación

Los protocolos de comunicación son normalmente definidos a diferentes niveles. El nivel más bajo especifica los métodos de interconexión, el nivel intermedio especifica el formato o sintaxis en que la información debe ser transmitida y el nivel más alto especifica el significado (no solo se refiere a la sustancia del mensaje sino también al tipo del mensaje).

Existen tanto protocolos de comunicación binarios como n-arios. Un protocolo binario involucra un solo remitente y un solo receptor, mientras que los protocolos n-arios involucran un solo remitente y varios receptores. Un protocolo es especificado por una estructura de dato con los 5 campos siguientes:

1. remitente
2. receptor(es)
3. lenguaje en el protocolo
4. funciones de codificación y decodificación
5. acciones que deben ser tomadas por el(los) receptor(es).

7.3. Acciones del habla

La comunicación humana se usa como modelo para la comunicación entre agentes computacionales. Una base popular para analizar la comunicación humana es la teoría de acciones del habla. Esta teoría el lenguaje natural humano es visto como acciones, tales como: demandas, sugerencias, compromisos, y respuestas. Por ejemplo, cuando usted pide algo, usted no está haciendo una simple declaración sino que hace una demanda propiamente. Cuando un jurado declara a un demandado culpable, hay una acción tomada: el estado social del demandado cambia. Un acto del habla tiene tres aspectos:

1. La locución, la pronunciación física del locutor.
2. La alocución, el significado intencional de la pronunciación por del locutor.
3. Perlocución, la acción que es el resultado de la locución.

Por ejemplo Juan puede decirle a Maria: Por favor cierra la ventana. Este acto consiste del sonido físico generado por Juan o la secuencia de caracteres tecleados por Juan

(locución). La intención del menaje de Juan es una demanda o instrucción (alocución) y si todo va bien, la ventana será cerrada (Perlocución). En la comunicación humana, no siempre es fácil de determinar. Por ejemplo: "Tengo frío", puede ser visto como una afirmación; o una demanda de un abrigo o para incrementar la temperatura de la habitación. Pero en la comunicación entre agentes queremos asegurar que no quede duda sobre el tipo del mensaje.

Esta teoría usa el término performative para identificar la intención del mensaje. Ejemplos de los verbos incluidos en los performative son: prometer, reportar, convencer, insistir, decir, pedir, demandar. En resumen, la teoría del las acciones del habla ayuda a definir los tipos de mensajes usando el concepto de fuerza de elocución la cual restringe la semántica del acto de comunicación.

Las intenciones del remitente se definen claramente y el receptor no tiene dudas sobre el tipo de mensaje. Esta restricción simplifica el diseño de nuestros agentes.

El mensaje enviado con el protocolo puede ser ambiguo, puede no tener una respuesta simple, puede ser necesaria una descomposición y la ayuda de otros agentes; sin embargo, el protocolo de comunicación debe identificar claramente el tipo del mensaje que se envía.

7.3.1. Knowledge Interchange Format (KIF)

Los agentes necesitan describir las cosas del mundo real. Esta descripción puede ser expresada en lenguaje natural como inglés o japonés, los cuales pueden representar una gran variedad de cosas y situaciones. Pero como ya vimos, el significado de las afirmaciones puede esta sujeto a diferentes 100 Sección 7.3 Acciones del habla
interpretaciones. Como sabemos también la lógica simbólica es una herramienta matemática usada para describir cosas.

Las lógicas simples (por ejemplo, el cálculo de predicado de primer orden) se han encontrado adecuadas para describir casi todas las cosas de interés o utilidad para las personas y otros agentes inteligentes. Estas cosas incluyen hechos concretos, definiciones, reglas de la inferencia, restricciones e incluso meta-conocimiento.

KIF, es un lenguaje lógico particular, se ha propuesto como un standard para describir las cosas dentro de los sistemas expertos, bases de datos, agentes inteligentes, etc. Es legible tanto por sistemas computarizados y como por personas.

KIF es una versión de cálculo de predicado de primer orden que usa notación prefija. La descripción del lenguaje incluye tanto una especificación para su sintaxis como para su semántica.

KIF incluye una variedad de operadores lógicos para ayudar en la codificación de la información lógica, como la negación, la disyunción, reglas, y cuantificación de fórmulas. Además, brinda un vocabulario de objetos como números, caracteres y cadenas de caracteres. También ofrece un conjunto de funciones y relaciones básicas como la suma y la relación de menor igual. Y por último una representación de lista como la de LISP. Con este conjunto de básico es posible definir nuevos objetos, funciones y relaciones entre estos objetos.

KIF mantiene expresiones simples de datos. Por ejemplo, las expresiones mostradas a continuación codifican los datos de tres tuplos en una base de datos de personal (los argumentos simbolizan el número de identificación de un empleado, la sección en que trabaja y sueldo respectivamente):

```
(salary 015-46-3946 widgets 72000)
(salary 026-40-9152 grommets 36000)
(salary 415-32-4707 fidgets 42000)
```

El siguiente ejemplo afirma que la temperatura de m1 es de 83 grados Celsius.

```
(= (temperature m1)
(scalar 83 Celsius))
```

La relación = es la relación de igualdad que conocemos que brinda el standard, temperatura es una función unaria y scalar es una función binaria estas dos funciones deben ser definidas.

Información más complicada puede ser expresada a través del uso de términos complejos. Por ejemplo, la siguiente sentencia afirma que un barco es más grande que otro:

```
(¿(* (width chip1) (length chip1))
(* (width chip2) (length chip2)))
```

La expresión mostrada debajo es un ejemplo de una frase compleja en KIF. Afirma que el número obtenido levantando cualquier número real ?x a una potencia par ?n es positivo:

```
(=> (and (real-number ?x)
(even-number ?n)
)
(> (exp ?x ?n) 0)
)
```

En la siguiente expresión muestra como las definiciones pueden ser usadas para introducir nuevos conceptos en términos de los ya existentes. Aquí se afirma que un objeto es soltero si es un hombre y no esta casado:

```
(deffuntion bachelor (?x) :=
(and (man ?x)
```

```
(not (married ?x))
)
)
```

?x es una variable, man y married son relaciones unarias y el símbolo := significa “por definición”.

Este otro ejemplo muestra como las relaciones entres los individuos del dominio pueden ser declaradas:

```
(defrelation (person ?x) :=> (mamal ?x))
```

Aquí se declara que todos los objetos que sean personas son también mamíferos. KIF ofrece para la codificación de conocimiento sobre conocimiento, usando los operadores coma (,), la comilla (’) y vocabulario relacionado. Por ejemplo, la siguiente sentencia afirma que Joe esta interesado en recibir triplo en la relación salario. El uso de las comas indica que las variables deben tomarse literalmente.

```
(interested Joe ’(salary ,?x ,?y ,?z))
```

Sin las comas, esta sentencia diría que el agente Joe esta interesado en la sentencia (salary ?x ?y ?z) en vez de en sus instancias. KIF también puede ser usado para describir procedimientos: programas para que los agentes ejecuten. El último ejemplo es un programa de tres líneas; el primer paso asegura que haya una línea vacía en la salida standard, la segunda imprime “Hello!” y el último paso vuelve a cambiar de línea.

```
(progn (fresh-line t)
(print "Hello!")
(fresh-line t)
)
```

7.3.2. Knowledge Query and Manipulation Language (KQML)

Una decisión fundamental para la interacción entre agentes es separar la semántica del protocolo de comunicación de la semántica del mensaje que se envía. El protocolo de comunicación debe ser compartido universalmente por todos los agentes. Debe ser conciso y con solo un número limitado de primitivas de acciones de comunicación.

KQML es un protocolo para el intercambio de información y conocimiento. Lo interesante de KQML es que toda la información para entender el mensaje se incluye en la comunicación propiamente.

El protocolo básico se define por la siguiente estructura:

```
(KQML-performative
        :sender <word>
        :receiver <word>
        :language <word>
        :ontology <word>
        :content <expression>
        ...)
```

Los argumentos, definidos por una palabra clave precedido de dos puntos, se pueden poner en cualquier orden. Los performative de KQML se modelan como las preformativas de la teoría del Actos del habla. De esta forma, los performative de KQML son independientes del dominio mientras que la semántica del mensaje es definido por el campo :content (que es propiamente el mensaje), : lenguaje es el lenguaje en que esta expresado el mensaje y :ontology es el vocabulario de las "palabra" usadas en el mensaje. En efecto KQML envuelve el mensaje en una estructura que puede ser entendida por cualquier agente, pero para entender el mensaje el receptor debe de entender el lenguaje y tener acceso a la ontología. Los términos :content, :language y :ontology delinean la semántica del mensaje. Otros argumentos como :sender, :receiver, :reply-with, :inreply-to son parámetros para la transmisión del mensaje. KQML asume que la comunicación es asíncrona y los campos :reply-with de un remitente y in-reply-to del agente que responde enlazan el mensaje enviado con la respuesta esperada.

KQML es parte de un amplio esfuerzo para desarrollar metodologías para distribuir información entre diferentes sistemas. Una parte de este esfuerzo involucra la definición de Knowledge Interchange Format (KIF). Otra parte de estos esfuerzos es definir ontologías que definan los conceptos comunes, atributos y relaciones para diferentes subconjuntos de conocimientos del mundo La definición de términos antológicos dan significado a las expresiones representadas en KIF. Por ejemplo, en el ya archiconocido mundo de los bloques, si el concepto de bloque de madera de dimensiones determinadas es representado por el predicado unario Block, entonces el hecho que el bloque A esto sobre el bloque B puede ser comunicado de la siguiente manera:

```
 (tell
        :sender Agent1
        :receiver Agent2
        :language: KIF
        :ontology: Blocks-World
        :content (AND (Block A) (Block B) (On A B))
)
```

El lenguaje en KQML por supuesto no se restringe a KIF; otros lenguajes como PROLOG, SQL o cualquier otro lenguaje pueden ser usados. Los agentes que se comunican mediante KQML pueden ser tanto master como slaves. La comunicación puede ser tanto sincrónica como asíncrona. En la comunicación sincrónica el agente que envía un mensaje espera por la respuesta. En la comunicación asíncrona el agente que

envía el mensaje, continua con su ejecución, la cual es interrumpida cuando la respuesta llegue un tiempo después.

Es interesante hacer notar que los mensajes de KQML pueden estar anidados: en el contexto de un mensaje KQML puede haber otro mensaje KQML, el cual es independiente. Por ejemplo, supongamos que el agente1 no puede comunicarse directamente con el agente2, pero si con el agente agente3, entonces el agente1 puede pedirle al agente3 que le envié el mensaje al agente2.

```
(forward
        :from Agent1
        :to Agent2
        :sender Agent1
        :receiver Agent3
        :language KQML
        :ontology kqml-ontology
        :content (tell
                        :sender Agent1
                        :receiver Agent2
                        :language KIF
                        :ontology: Blocks-World
                        :content (On (Block A) (Block B))
                )
)
```

En un mensaje KQML del tipo forward, el valor del campo :from se convierte en el valor del campo :sender del mensaje del :content y el valor del campo :to se convierte en el valor del campo :receiver.

Las performative de KQML pueden organizarse dentro de 7 categorías:

- Preguntas básicas (evaluate, ask-one, ask-all, ...)
- Preguntas de respuestas múltiples (stream-in, stream-all, ...)
- Respuestas (reply, sorry, ...)
- Información genérica (tell, achieve, cancel, untell, unachieve, ...)
- Generación (standby, ready, next, rest, ...)
- Definición de capacidades (advertise, subscribe, monitor, ...)
- Comandos de Redes (register, unregister, forward, broadcast, ...)

La performative advertise, es usada por :sender agent2 para informar a :receiver agent1 sobre las capacidades del remitente:

```
(advertise
        :sender Agent2
        :receiver Agent1
        :language KQML
```

```
        :ontology kqml-ontology
        :content (ask-all
                        :sender Agent1
                        :receiver Agent2
                        :in-reply-to id1
                        :language Prolog
                        :ontology: Blocks-World
                        :content "on(X,Y)"
                )
)
```

Ahora agent1 puede preguntar a agent2:

```
(ask-all
        :sender Agent1
        :receiver Agent2
        :in-reply-to id1
        :reply-with id2
        :language: Prolog
        :ontology: Blocks-World
        :content "on(X,Y)"
)
```

Agent2 puede responder con:

```
(tell
        :sender Agent2
        :receiver Agent1
        :in-reply-to id2
        :language: Prolog
        :ontology: Blocks-World
        :content "[on(a,b),on(c,d)]"
)
```

Algunas cuestiones:

El remitente y el receptor deben entender el lenguaje de comunicación que se usa, la ontología debe ser creada y ser accesible para los agentes que sigan la comunicación.
KQML debe operar dentro de una infraestructura de comunicación que permita a los agentes localizarse unos a otros. Esta infraestructura no es parte de la especificación de KQML y sistemas implementados usan programas llamados router o facilitators para realizar estas funciones.

KQML es todavía un trabajo en progreso y su semántica no esta completamente definida. Labrou and Finin han propuesto recientemente una nueva especificación que redefine el trabajo original.

Sin embargo, no existe una especificación oficial confiable.

7.3.3. FIPA agent communication lenguaje (ACL)

En 1995, la fundación para agentes físicos inteligentes (FIPA de sus siglas en inglés), comenzó a desarrollar estándares para agentes. El centro de esta iniciativa es fue el desarrollo de un ACL.

Este ACL es superficialmente parecido a KQML: define un lenguaje exterior para los mensajes, define 22 perfomatives para definir la interpretación intencional del mensaje y no acuerda ningún lenguaje en específico para los mensajes en el contexto. Además, la sintaxis es similar a la de KQML:

```
(inform
        :sender agent1
        :receiver agent2
        :content (price good2 150)
        :language sl
        :ontology hpl-auction
)
```

La diferencia principal entre FIPA ACL y KQML es la colección de performative que brindan. En la tabla siguiente se listan las performative de FIPA ACL:

Performative	Paso información	Pedido de información	Negociación	Realizando acciones	Tratamiento de Errores
Accept-proposal			x		
Agree				x	
Cancel		x		x	
Cfp			x		
Confirm	x				
Disconfirm	x				
Failure					x
Inform	x				
Inform-if	x				
Inform-ref	x				
Not-understood					x
Propagate				x	
Propose			x		
Proxy				x	
Query-if		x			
Query-ref		x			

Refuse				x	
Reject-proposal			x		
Request				x	
Request-when				x	
Request-whenever				x	
Subscribe		x			

El significado informal de los performative es:

Accept-proposal: Este performative permite a un agente afirmar que acepta la propuesta hecha por otro agente.

Agree: Es usado por un agente para indicar que va a realizar la una acción pedida.

Cancel: Se usa para indicar que no se desea continuar realizando una acción.

Cfp(call for proposals): se usa para iniciar una negociación. El contento de este mensaje debe incluir dos cosas: una acción y una condición. En esencia dice "Estoy dispuesto a realizar esta acción bajo estos términos".

Confirm: Permite al emisor de un mensaje confirmar la veracidad del content al receptor.

Disconfirm: Similar a Confirm, pero indica que al receptor que el content del mensaje es falso.

Failure: Para indicar que el intento de realizar una determinada acción fallo.

Inform: Junto con request, esta performative es una de las más de las dos más importante de FIPA ACL. Es el mecanismo básico para comunicar información. El content del un mensaje inform es una afirmación y la idea es que el emisor del inform quiere que el receptor crea el content.

Inform-if: Es usado para saber si el content del mensaje es verdadero o falso. Normalmente es l performative del content del un mensaje. Un agente envía un request a otro agente, en el content del mensaje hay un mensaje inform-if. La idea es que el que envía el request dice: "Dime si el contenido del inform-if es verdadero o falso".

Inform-ref: Similar a inform-if, pero en vez de preguntar por si el content es verdadero o falso el agente pregunta por el valor (evaluación) de una expresión.

Not-understood: Es usado por un agente para indicar a otro agente que reconoció que realizó alguna acción, pero no entendió por qué esta acción fue realizada. Es usado por un agente para indicarle a otro agente que el mensaje que no entendió el mensaje que acabó de recibir. El content de este mensaje es la acción que no se entendió y una afirmación que explica el porque no se entendió la acción.

Propagate: El content de un mensaje propagate tiene dos parte: otro mensaje y una expresión que denota un conjunto de agentes. La idea es que el receptor de un mensaje de este tipo envíe el mensaje del content a todos los agentes del conjunto.

Propose: Esta performative permite a un agente hacer una propuesta a otro agente.

Proxy: Este tipo de mensaje permite al agente que lo envía usar al receptor como un proxy para un conjunto de agentes. El content de este mensaje debe tener otro mensaje y un conjunto de agentes a los cuales se les enviara el mensaje.

Query-if: Permite que un agente le pregunte a otro su una afirmación en especifico es verdadero o no.

Query-ref: Es usada por un agente para determinar el valor de una expresión.

Refuse: Es usado por un agente para afirmar que no realizara una acción determinada. El content debe incluir la acción y una expresión que caracterice el porque no se realizara la acción.

Reject-proposal: Permite a un agente indicar que no acepta un proposal de otro agente. El content debe tener el proposal que se rechaza y una expresión que caracterice que caracterice el porque no.

Request: Permite a un agente pedir a otro agente que realice una acción.

Request-when: El content de este mensaje debe ser: una acción y una afirmación, la idea es que el que envía el mensaje quiere que se ejecute la acción cuando la afirmación sea verdadera.

Request-whenever: Similar al anterior pero el receptor puede ejecutar la acción cada vez que la afirmación sea verdadera.

Subscribe: Como en KQML: el content es una afirmación y el emisor quiere ser notificado cada vez que algo respecto a la afirmación cambie.

Dado que una de las cosas más criticadas a KQML es la pérdida de la semántica. Los desarrolladores de FIPA ACL dieron una gran importancia a dar una semántica formal a su lenguaje.

La semántica fue dada con respecto a un lenguaje llamado SL. Este lenguaje permite representar creencia, deseos y creencia inciertas de un agente. La semántica de FIPA ACL transforma cada mensaje del ACL en una fórmula de SL, la cual define una restricción que el debe satisfacer (FIPA se refiere a esta restricción como condición de factibilidad). La semántica también transforma un mensaje a una fórmula de SL que define el efecto “racional” de la acción: el propósito del mensaje (lo que el agente

pretende lograr enviando el mensaje: acción perlocutiva). Sin embargo, en una sociedad de agentes no hay nada que garantice el efecto racional de un mensaje. Por lo que conforma a esto, el receptor de un mensaje no tiene porque respectar la acción racional de un mensaje ACL solo la condición de factibilidad.

Como se menciono anteriormente las primitivas más importantes de FIPA ACL son inform y request. De hecho, las demás primitivas están definidas en términos de estas dos. A continuación se presenta la semántica de inform:

> $< i, inform(j, \varphi) >$
> feasibility condition: $B_i\varphi \wedge \neg B_i(B_if_j\varphi \vee U_if_j\varphi)$
> rational effec: $tB_j\varphi$

$B_i\varphi$ significa que el "agente i cree"; $B_if_j\varphi$ significa que el agente i tiene una opinión sobre la falsedad o veracidad de φ y $U_ifj\varphi$ significa que el agente i tiene incertidumbre sobre φ.

Entonces, un agente i envía un mensaje inform a un agente j con content φ, si él cree φ y no cree de j que j conoce si φ es verdadero o falso; ni j tiene incertidumbre si φ es verdadero o falso. Si el agente realiza exitosamente el inform entonces el agente j creerá φ.

La semántica del request (algo simplificada):

> $< i, request(j, \varphi) >$
> feasibility condition: $B_iAgent(\alpha, j) \wedge \neg B_iI_jDone(\alpha)$
> rational effect: $Done(\alpha)$

En una extensión de SL $Agent(\alpha, j)$ significa que el agente j es capaz de realizar la acción alpha y Done(α) significa que la acción alpha se realizó. Entonces el agente i le pide al agente j que realice la acción alpha si i cree que j puede realizar la acción y i cree que el agente j no cree actualmente que α se ha realizado. El efecto racional es que se realice la acción α.

7.4. Ontologías

Una ontología es una especificación de objetos, conceptos y relación en un área de interés. En el ejemplo del mundo de los bloques que vimos anteriormente el término Block representa un concepto y el término On representa una relación. Conceptos pueden ser representados con lógica de primer orden como predicados y predicados de mayor aridad representan relaciones. Para expresar la idea de que un bloque es un objeto físico, debemos usar la expresión de primer orden: $\forall$x(Blockx) => (PhysicalObjectx). Existen otras representaciones más generales. En vez de usar (Block A), se puede usar la

expresión (instanceOf A Block), ahora tanto A como Block son objetos del universo y se introducen las nuevas relaciones instanteOf y subclass:

(class Block)
(class PhysicalObject)
(subclassOf Block PhysicalObject)
∀x, y, z(instanceOfxy)(subclassyz)) (instancexz)

Esta última expresión es una regla que expresa la noción de jerarquía.

Por ejemplo, no hay ninguna necesidad de representar A en la ontología para (Block A) o (instanceOf A Block). Una ontología es análoga al esquema de una base de datos, no al contenido de la base de datos. Un agente debe representar su conocimiento en el vocabulario de una ontología específica. Debido a que los agentes son construidos por personas entonces el creador de un agente debe usar una ontología especificada para representar el conocimiento del agente. Todos los agentes que comparten la misma ontología para la representación de conocimiento tienen una comprensión de las ”palabras” en el lenguaje de comunicación.

Muchos agentes tienen bases de conocimiento en que se definen las relaciones en más detalle que con una simple cadena de caracteres. Por ejemplo, pueden especificarse el dominio y rango de una relación binaria;

(domain On PhysicalObject)
(range On PhysicalObject)

Esta restricción limita los valores permitidos en una relación. (On A B) se permite ya que tanto A como B son instancias de PhysicalObject por la transitividad de la cláusula subclassOf; por otra parte (On A Dream1) no debe ser permitido asumiendo que Dream1 no es del tipo PhysicalObject.

Los editores de ontologías normalmente son sistemas que permiten al usuario definir una ontología y sus componentes: clases, instancias, relaciones y funciones. KIF ha sido el lenguaje más usado para crear ontologías y se han desarrollado muchas herramientas que usan KIF para desarrollar las ontologías. El más conocido de estos editores es: Ontolingua Server (http://ontolingua.stanford.edu/frame-o http://www-ksl-svc.stanford.edu/ o http://www.ksl.stanford.edu/software/ontolingua/).

La estructura de Ontolingua Server se muestra en la figura #2. La componente principal es una librería de ontologías expresadas el lenguaje de definición de ontologías Ontolingua (basado en KIF). El Server brinda el acceso a esta librería por diferentes vías: editando directamente las ontologías a través de una interfaz Web o con programas que se conectan a través de la interfaz NGFP.

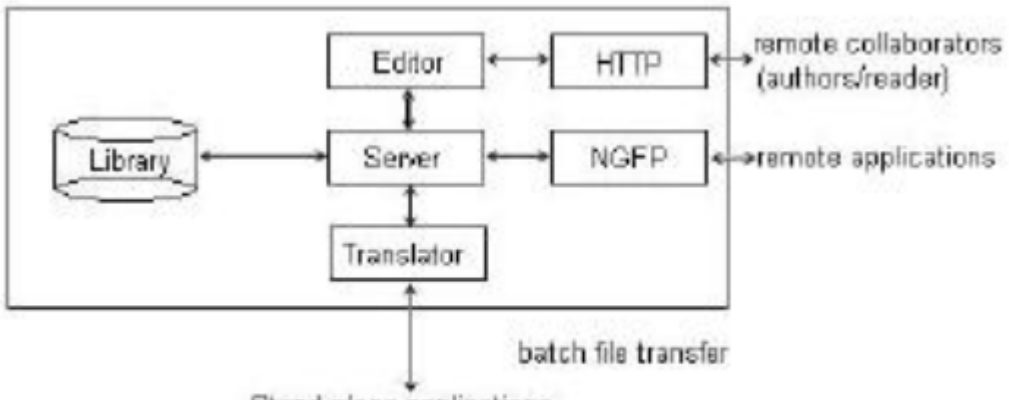


Figura #2: Arquitectura de Ontolingua server.

Muchos otros lenguajes se usan para definir ontologías como el XML.

7.4.1. Protocolos de interacción de agentes

Hemos estudiado anteriormente mecanismos de comunicación. Los protocolos de interacción controlan el intercambio de una serie de mensajes entre agentes: una conversación. Se han diseñados muchos varios protocolos de este tipo. En los casos donde los agentes tienen conflictos con objetivos o simplemente tiene intereses propios, el objetivo del protocolo es maximizar las utilidades del agente. En los casos en que los agentes tiene objetivos o problemas comunes, el objetivo del protocolo es mantener un rendimiento global coherente de los agentes sin violentar su autonomía (sin un control global explícito). En este último caso se deben tener en cuenta los siguientes aspectos:

- determinar objetivos comunes.
- determinar tareas comunes
- evitar conflictos innecesarios
- combinar el conocimiento.

7.4.2. Protocolos de cooperación

Una estrategia básica usada por muchos de los protocolos de cooperación es descomponer y distribuir tareas. Como divide y vencerás pueden reducir la complejidad de una tarea: tareas más pequeñas necesitan agentes menos capaces y menos recursos. Sin embargo, el sistema debe decidir entre descomposiciones alternativas, si existen, y el proceso de descomposición debe considerar los recursos y capacidades de los agentes.

También, pueden existir interacciones entre subtareas y conflictos entre los agentes. La descomposición de las tareas pueden ser realizada por los diseñadores del sistema, por lo que la descomposición es programada durante la implementación o por los agentes usando planes jerárquicos.

Una vez que las tareas son descompuestas, pueden ser distribuidas de acuerdo a los siguientes criterios:

- Evitar sobrecarga recursos críticos.
- Asignar tareas a agentes con capacidades correspondientes.
- Diseñar un agente con una visión global para asignar tareas a otros agentes.
- Asignar responsabilidades solapadas a agentes para lograr coherencia.
- Asignar tareas altamente interdependientes a agentes con proximidad espacial o semántica. Esto minimiza la comunicación y los costos de sincronización.
- Reasignar tareas si es necesario para completar tareas urgentes.

Los siguientes mecanismos son normalmente usados para distribuir tareas.

- Mecanismo de mercado (market mechanisms): Las tareas son asignadas a agentes por consenso general o selección mutua.
- Red de contrato (Contract net): anuncio, propuestas y ciclo de otorgamiento de tareas.
- Planeamiento multiagentes: Agentes planeadores tiene la responsabilidad de asignar las tareas.
- Estructura organizativa: los agentes tienen responsabilidades fijas.

7.4.3. Contract Net

El mas conocido de los mecanismos mencionados anteriormente es la Red de contrato. Este es un protocolo de interacción para agentes que cooperan en la solución de problemas. Se modela mediante el mecanismo de contrato usado en los comercios para controlar el intercambio de mercancías y servicios. La red de contrato brinda una solución para el llamado problema de conexión: encontrar un agente apropiado para trabajar en una tarea determinada. El proceso se define como sigue.

Un agente que no puede resolver una tarea es llamado: el jefe; agentes que pueden resolver la tarea son llamados: potenciales contratista. Desde la perspectiva del jefe el proceso es:

- Anunciar una tarea que debe ser realizada.
- Recibir y evaluar las ofertas de los potenciales contratistas.
- Otorgar un contrato al contratista adecuado.

- Recibir y sintetizar los resultados.

Desde el punto de vista del los contratistas el proceso es:

- Recibir el anuncio de la tarea.
- Evaluar su capacidad de repuesta.
- Responder (declinar, proponer).
- Realizar la tarea si su propuesta fue aceptada.
- Reportar los resultados.

El rol de un agente no esta predefinido. Cualquier agente puede actuar como jefe haciendo anuncios de tareas; y cualquier agente puede actuar como contratista respondiendo a los anuncios. Esta flexibilidad permite futuras descomposiciones de tareas: un contratista de una tarea en específica puede actuar como jefe solicitando la ayuda de otros agentes para resolver parte de la tarea.

La red de contrato ofrece la ventaja de una degradación del rendimiento elegante. Si un contratista no puede brindar una solución satisfactoria el jefe puede buscar otro contratista para la tarea.

La estructura del anuncio de tareas incluye: asignación, especificación de elegibilidad, abstracción de la tarea, especificación de propuestas, tiempo de vencimiento. La tarea puede ser asignada a uno o varios contratistas quienes deben cumplir con el criterio de elegibilidad. La abstracción de la tarea, una pequeña descripción de la tarea, se usa para los contratistas para clasificar las tareas de varios anuncios. La especificación de propuestas le informa a los contratistas que información deben enviar con la propuesta. El tiempo de vencimiento es un límite para la recepción de propuestas.

Cada contratista evalúa las tareas anunciadas para determinar si puede emitir alguna propuesta.

El contratista escoge la tarea que le es más atractiva y hace su propuesta al jefe, este recibe y evalúa las propuestas para cada tarea anunciada, y notifica a los contratistas con un mensaje de otorgamiento. Una limitante de este mecanismo es que una tarea puede ser asignada a contratistas con capacidades limitadas si el mejor contratista esta ocupado. Otra limitación es que el jefe no esta en la obligación de informar a los contratistas que se ya se realizó un otorgamiento.

Un jefe puede que no reciba ofertas por varias causas: todos los contratistas potenciales están ocupados con otras tareas, un contratista puede estar desocupado pero clasifica la tarea por debajo de otras, ningún contratista es capaz de realizar la tarea. Para tratar con estos problemas, un jefe puede pedir propuestas inmediatas a las cuales los contratistas pueden responder con mensajes como: elegible pero ocupado, no elegible o no interesado. El jefe puede esperar hasta que se desocupe un contratista.

La red de contrato ofrece contratos directos sin negociación. El contratista seleccionado responde con un mensaje de aceptación o rechazo. Esta capacidad puede simplificar el protocolo o mejorar la eficiencia para determinadas tareas.

7.4.4. Sistemas de pizarras

La solución de problemas basados en pizarras se presenta con frecuencia de la siguiente manera:

"Imagine un grupo de especialistas humanos o no, situados enfrente de una pizarra. Los especialistas trabajan cooperativamente para resolver el problema, usando la pizarra como lugar de trabajo para desarrollar la solución. La solución del problema comienza cuando se escriben en la pizarra el problema y los datos iniciales. Los especialistas observan la pizarra, buscando una oportunidad de aplicar sus conocimientos y habilidades para desarrollar la solución. Cuando un especialista tiene suficiente información para realizar una contribución, él escribe su contribución en la pizarra. Esta información adicional puede permitir a otros agentes aplicar sus habilidades. Este proceso de adicionar contribuciones a la pizarra continua hasta que el problema es solucionado"

Esta metáfora captura varias características importantes en los sistemas con pizarras, que se describen a continuación. Independencia de las habilidades: Los especialistas (llamados fuentes de conocimientos FsC) no están entrenados para trabajar solamente un grupo de especialistas en especifico.

Cada uno es un experto en algún aspecto del problema y pude contribuir a la solución independientemente de la mezcla de especialistas. Diversidad de técnicas en la solución del problema: En los sistemas de pizarras, la representación interna y la maquinaria de inferencia usada por cada FsC se encuentran ocultos.

Representación flexible de la información de la pizarra: El modelo de la pizarra no impone ninguna restricción a priori de que información puede ser puesta en la pizarra. Lenguaje interacción común: FsC en estos sistemas deben ser capaces de interpretar correctamente la información puesta en la pizarra por otras FsC. En la práctica, no hay diferencia entre la expresividad de la representación de una representación compartida por unas pocas FsC y representación general entendida por todas las FsC.

Activación por eventos: Las FsC en los sistemas de pizarras se activan en respuesta a eventos de la pizarra y externos. Los eventos de la pizarra son: adición de información nueva, cambio de la información existente o eliminación de información. Por otro lado, en vez de que cada FsC busque en la pizarra, cada FsC informa sobre el tipo de evento que esta interesado. El sistema de pizarra guarda esta información y considera la activación de la FsC si el tipo de evento ocurre.

Necesidad de control: Una componente de control aparte de cada FsC es responsable de dirigir el curso de la solución del problema. La componente de control puede ser vista como un especialista en control de solución de problemas, considerando el beneficio global de las contribuciones que pueden ser hechas por las FsC activadas. Cuando la activación de actual de una FsC se completa, la componente de control selecciona la activación de FsC pendiente más apropiada. Cuando una FsC, usa sus habilidades para evaluar la calidad e importancia de su contribución se activa informa a la componente de control de la calidad y costo de su contribución sin realizar ningún trabajo para computar la contribución. La componente de control con estas estimaciones decide que hacer.

Generación creciente de la solución: Las FsC contribuyen a la solución como estime mas conveniente, en ocasiones refinado, en ocasiones contradiciendo y en ocasiones iniciando una nueva línea de razonamiento.

7.4.5. Negociación

Una forma frecuente de interacción que ocurre entre agentes con diferentes objetivos es llamada negociación. La negociación es el proceso por el cual una decisión unánime es lograda por dos o más agentes, cada uno tratando de lograr sus objetivos individuales.

Los agentes primeramente comunican sus intenciones, las cuales pueden ser conflictivas, y después tratan de obtener un acuerdo haciendo concesiones o buscando alternativas. Las características principales de las son:

1. El lenguaje usado por los agentes que participan.
2. El protocolo seguido por estos agentes
3. El proceso que cada agente usa para determinar sus posiciones, concesiones y criterios de consenso.

Muchos grupos han desarrollado sistemas y técnicas para negociar. Estas pueden ser o centradas en el ambiente o centradas en el agente. Los desarrolladores de técnicas centradas en el ambiente, fijan su atención en el siguiente problema: “Como pueden las reglas del ambiente ser diseñadas para que los agentes en él, a pesar de sus capacidades o intenciones originales, interactúen productiva y claramente”. El mecanismo resultante debe tener los siguientes atributos:

Eficiencia: Los agentes no deben derrochar recursos en llegar a un acuerdo.

Estabilidad: Ningún agente debe tener un intensivo para desviarse de las estrategias acordadas.

Simplicidad: El mecanismo de negociación debe imponer demandas de bajo costo a los agentes.

Distribución: el mecanismo no debe necesitar una toma de decisión central.

Simetría: El mecanismo no debe parcializarse contra ningún agente arbitrariamente o por razones inapropiadas. Se han identificado dos tipos de ambientes: dominios orientados al valor y dominios orientados a tareas. En estos últimos, los agentes tienen un conjunto de tareas que debe lograr, todos los recursos necesarios para lógralos están disponibles y el agente puede lograrla sin la ayuda o interferencia de otros agentes. Sin embargo, los agentes pueden beneficiarse si comparten algunas de las tareas. Por ejemplo, "el dominio de descargas de Internet", cada agente tiene una lista de documentos que deben descargar. Cada documento tiene asociado un costo de descarga, los cuales el agente debe querer minimizar. Si un documento es común para varios agentes, ellos pueden economizar el costo de la descarga, descargando el documento una sola vez y después compartiéndolo. El ambiente debe brindar los siguientes mecanismos simples de negociación y restricciones:

- cada agente declara los documentos que quiere,
- los documentos comunes a dos o más agentes son asignados a agentes al azar,
- los agentes pagan por los documentos que ellos descargan y
- se le concede a los agentes acceso a los documentos que descargan así como a cualquier de su lista común.

El mecanismo es simple, simétrico, distribuido y eficiente. Para determinar la estabilidad, debemos considerar las estrategias de los agentes.

Una estrategia óptima para un agente es declarar el conjunto verdadero de documentos que necesita, sin importar cual es la estrategia que adopten los otros agentes o los documentos que necesite.

Debido a que no hay incentivo para un agente se desvié de su estrategia, entonces el mecanismo es estable.

Los desarrolladores de mecanismos de negociación centrados en los agentes fijan su atención en el problema: "Dado un ambiente en el cual mi agente debe operar. ¿Cuál es la mejor estrategia que debe seguir?" Se han desarrollados muchas estrategias de negociación para problemas específicos pero solo unos pocos principios de negociaciones generales. Sin embargo, hay dos métodos generales, ambos basados en suposiciones sobre el tipo de agentes involucrados.

El primer método, un clasificador de acciones del habla junto con una posible semántica del mundo se usan para formalizar los protocolos de negociación. Esto aclara las condiciones de satisfacción para los diferentes tipos de mensajes.

El segundo método se basa en suponer que los agentes son económicamente racionales. Como se vera mas adelante, el conjunto de agente debe ser pequeño, deben tener un lenguaje común y abstracción común del problema; y deben buscar una solución común.

Bajo estas suposiciones se desarrolló un protocolo unificado de negociación. Los agentes que siguen este protocolo hacen un trato: un plan conjunto entre los agentes que debe satisfacer todos los objetivos. La utilidad del trato para un agente es la cantidad que el quiere pagar menos el costo del trato. Cada agente quiere maximizar su utilidad. Los agentes discuten un conjunto de negociación, que es el conjunto de todos los tratos que tienen utilidades positivas para todos los agentes.

En términos formales. Un dominio orientado a tarea bajo este método es un triplo < T, A, c > donde T es el conjunto de tareas, A es el conjunto de agentes y c(X) es una función monótona para el costo de ejecutar las tareas X. Un trato es una redistribución de tareas. La utilidad de un trato d para el agente k es: $U_k(d) = c(T_k) - c(d_k)$.

El trato conflictivo D ocurre cuando el agente no puede lograr un trato. Un trato d es individualmente racional si d >. Un trato d es óptimo si no existe otro trato d_0 tal que d_0 > d. Al conjunto de todos los tratos individualmente racionales y óptimos se le llama conjunto de negociación (CN).

Hay tres casos:

1. Conflicto: no hay conjunto de negociación
2. Compromiso: los agentes prefieren estar solos, pero si no lo están ellos aceptarán un trato negociado.
3. Cooperativo: todos los tratos en el conjunto de negociación son preferidos por los agentes antes que tratar de lograr sus objetivos por separado.

Cuando hay un conflicto, entonces los agentes no se benefician de la negociación (prefieren actuar solos). En los otros dos casos la negociación es la mejor alternativa.

7.4.6. Mecanismo de Mercado

Todos los mecanismos descritos anteriormente necesitan la comunicación entre los agentes directamente por lo que son apropiados para grupos pequeños de agentes. Por lo que otros mecanismos son necesarios para un número grande o desconocido de agentes. Un mecanismo propuesto para estos casos es conocido como: “mecanismo de mercado”.

Es efectivo para coordinar la actividad de varios agentes con comunicaciones directas mínimas.

Cada cosa de interés para un agente se describe con precios actualizados. Hay dos tipos de agentes: consumidores, los que intercambia mercancías y los productores, quienes convierten algunas mercancías en otras. Los agentes ofrecen las mercancías a varios precios, pero todos los intercambios se realizan a los precios actuales del mercado.

Todas las propuestas de los agentes son para maximizar sus ganancias o utilidades. Para convertir el problema en términos de mercado computacional necesitamos especificar:

- las mercancías que serán comercializadas
- los agentes consumidores que comercializan mercancías.
- los agentes productores, con sus tecnologías para transformar algunas mercancías en otras.
- el orden y las conductas comerciales de los agentes.

Como el intercambio de mercancías está interconectado, el precio de una mercancía puede afectar el suministro y demanda de otras. El mercado alcanzará un equilibrio competitivo tal que:

1. las propuestas de los consumidores maximizan sus utilidades, sujeto a sus restricciones de presupuesto.
2. Las propuestas de los productores maximizar sus ganancias, sujeto a sus capacidades tecnológicas.
3. La demanda neta es cero para todas las mercancías.

Una propiedad de importante es que el equilibrio corresponde a una asignación de recursos y dicta las actividades y uso de los agentes. En general, el equilibrio no tiene porque existir o ser único, bajo ciertas condiciones, se puede garantizar que el equilibrio existe y es único.