UNIVERSIDAD COMPLUTENSE DE MADRID

SISTEMAS INFORMÁTICOS
CURSO 2006-2007

SISTEMA DE RIEGO INTELIGENTE BORROSO

María Guijarro Mata-García

Estefanía Tortajada Agudo

Fernando González Rivas

Dirigido por:

Luís Garmendia Salvador
Departamento de Ingeniería del Software e Inteligencia Artificial

Nosotros los autores del proyecto: *Sistema de Riego Inteligente Borroso*, de la asignatura de *Sistemas Informáticos*:

María Guijarro Mata-García **DNI: 75244418-X**
Estefanía Tortajada Agudo **DNI: 02666746-B**
Fernando González Rivas **DNI: 08856619-D**

Dirigidos por:

Luís Garmendia Salvador

Autorizamos a la Universidad Complutense de Madrid a utilizar y difundir, con fines académicos, el contenido de este documento de texto, así como del contenido del CD complementario que adjuntamos con él mismo.

María Guijarro Marta García

Estefanía Tortajada Agudo

Fernando González Rivas

Índice

1. La idea inicial

El proyecto que hemos propuesto consiste en una aplicación práctica de la lógica borrosa al campo del regadío, de tal manera que se pueda controlar el tiempo de apertura de válvulas de regadío en función de determinados parámetros meteorológicos, como pueden ser la temperatura ambiente, humedad relativa de la tierra, la incidencia del sol según la época del año en la que nos encontremos así como el grado de nubosidad ambiental, factores determinantes en el riego de cultivos.

Dadas las necesidades de agua hoy en día y la falta de reserva hidráulica en determinados años, nuestro sistema dispondrá también como entrada del nivel de la reserva hidráulica disponible para establecer, en caso de sequía, restricciones de riego impuestas por la Administración.

Se aplicará lógica borrosa (*fuzzy*) para el control de apertura de las válvulas. Para ello se elegirán distintos tipos de lógica para evaluar cual de ellas se adapta mejor al modelado óptimo.

Para ello se elegirán conjuntos borrosos que modelen los distintos parámetros principales elegidos, como son la temperatura ambiental, la humedad relativa de la tierra, percepción solar (dependiente de la época del año y el grado de nubosidad ambiental).

Para poder simular el control del riego, será necesario modelar determinados parámetros del suelo con el fin de recrear el comportamiento de un entorno de regadío concreto para poder validar el correcto funcionamiento del sistema y posibilitar en un futuro su implantación hardware.

La aplicación permitirá realizar una simulación continua de un sistema, pudiendo modificar las condiciones ambientales por el usuario, o cargando datos reales desde fichero para poder evaluar la respuesta del sistema. También se permitirá programar

riegos diarios y mensuales y poder comparar un posible ahorro de agua frente a sistemas de riego convencionales.

Para facilitar la comprensión, el usuario de la aplicación podrá analizar mediante gráficas, el comportamiento del sistema frente a ciertos parámetros de entrada.

2. Objetivo Final

2.1. Español

El proyecto consiste en un sistema de riego inteligente utilizando lógica borrosa que permite simular por software la evolución de un entorno de regadío simple en el tiempo, permitiendo modificar ciertas condiciones ambientales y posibilitando la elección de realizar inferencias para el cálculo del tiempo de riego según lógicas del producto, de Lukasiewicz o de Zadeh.

Para ello se desarrollará un entorno de simulación en Java para visualizar gráficamente el comportamiento de la aplicación frente a la variación de los distintos parámetros de entrada, ya sea de manera instantánea, diaria o mensual.

Se mostrará un estudio comparativo de la respuesta de nuestro sistema frente a otros sistemas habituales utilizados en el regadío.

2.2. English

This Project consists in an Intelligent Irrigation System using fuzzy logic for the correct software simulation of a simple irrigation environment. The system will show the evolution of some weather conditions making possible choosing some fuzzy logic inference method to evaluate the system output. In our case, we are using the product, Lukasiewicz and Zane logic.

In order to simulate the environment, we will develop a Java environment interface to show easily our application functioning when we are changing some input parameters into the instantaneous, daily or monthly simulation.

We will show comparative results of the output of our intelligent system against the other usual irrigation systems.

3. Introducción

3.1. Regadío

3.1.1. Contexto

El agua es un recurso escaso pero imprescindible para la vida: sólo cerca del 1% del agua del planeta es dulce y accesible para las personas.

Las fuentes, manantiales, cuencas, etc. están en acelerada vía de extinción debido a los cambios de clima y de suelo pero principalmente debido a la acción humana: deforestación, contaminación, uso ineficiente del agua...

El consumo de agua se ha triplicado desde el año 1950 sobrepasando la equivalencia al 30% de la dotación renovable del mundo que se puede considerar como estable.

La necesidad de lograr un equilibrio hidrológico que asegure el abasto suficiente de agua a la población sólo se puede conseguir armonizando la disponibilidad natural con las extracciones del recurso mediante el **uso eficiente del agua**.

3.1.2. Estado del arte

El sector agrícola es el mayor consumidor de agua (65%), no sólo porque la superficie irrigada en el mundo ha tenido que quintuplicarse, sino porque no se cuenta, en muchas ocasiones, con un sistema de riego eficiente.

El proyecto "*Optimizagua*" (octubre 2003 – septiembre 2006) en sus jornadas en mayo de 2006 en Logroño ha demostrado que combinando sistemas tradicionales de recogida y almacenamiento de aguas pluviales con la implantación de sistemas de riego inteligentes se consigue un ahorro de agua superior al 60% en zonas verdes públicas, 50% en zonas verdes privadas y 40% en agricultura (dependiendo del cultivo).

La mayoría de los sistemas de riego automático "inteligente" que existen en este momento en España son bastante limitados: evitan regar a determinadas horas, pueden ajustarse con la temperatura y llegan a parar si comienza a llover.

También hay sistemas automáticos que programa un experto en base a su experiencia pero no se realimentan con información obtenida del cultivo/ jardín.

Existen algunos sistemas de riego inteligente más complejos, como "*Tigris*", utilizado por el ayuntamiento de Valencia. Este sistema recibe información detallada de cada uno de los jardines, sobre la situación de riego y factores externos como lluvia, averías o actos de vandalismo. En caso necesario se paraliza el riego e incluso, se avisa a los operarios mediante mensajes al teléfono móvil. Es un proyecto que está en su inicio y ya está dando resultados.

En estas condiciones, consideramos importante el desarrollo de un sistema de riego inteligente que, utilizando técnicas de inteligencia artificial y basándose en lógica borrosa, consiga regular el riego en tiempo real optimizando el agua de un depósito del que se surte y reaccionando ante la situación externa.

Riego del Césped

En la mayoría de regiones, la lluvia no cubre las necesidades de los Céspedes y es necesario recurrir al riego.

El tiempo que se debe regar depende de varios factores:

- Si es un **clima** muy **lluvioso**, serán necesarios menos riegos, obviamente.
- Un césped que está a la sombra o que está protegido del viento, requiere menos riego que otro que está a pleno **sol** todo el día y además expuesto a **vientos** fuertes y secos, ya que el calor y el viento hacen que se evapore el agua más rápido y la cantidad de la misma que le llega a la planta es mucho menor.

- Si el **suelo** es arenoso hay que regar más que si fuera arcilloso, puesto que éste último retiene más agua. Los suelos arenosos son secos, retienen muy poca agua.

- Hay especies y **variedades** de Césped que consumen más agua que otras. Por ejemplo, un Ray-grass exige más riego que una Bermuda (Cynodon dactylon 'Numex-Sahara').

- La **época** del año. En verano hay máximo consumo de agua y en invierno el mínimo.
- También depende de la **calidad** que se quiera conseguir de Césped. Se puede regar menos si basta con un Césped un algo menos verde y vigoroso. Si se quiere máxima calidad hay que regar bastante más.

- Un césped **acostumbrado** a que viva con poca agua tiene que ser regado mucho menos que otro acostumbrado a un riego continuo.

Cuando un Césped se riega muy a menudo, las raíces se desarrollan muy superficialmente, no profundizan, puesto que no tienen necesidad de buscar agua abajo. Un Césped que se **riega poco** desarrolla un sistema **radicular más profundo** y más potente (dentro de lo que cabe para ser un Césped).

En el Sur de España es muy común regar todos los días del verano 30 minutos de aspersores. Sin embargo, un Césped se puede mantener perfectamente regando mucho menos acostumbrándolo a un riego moderado. Quizás se pierda un poco de verdor, pero se puede ahorrar muchísima agua.

Datos de riego para un clima como el de Madrid y un suelo medio (ni arcillos ni arenoso):

- **Invierno**: sólo riegos de apoyo o mantenimiento cada 20-25 días si la lluvia no es suficiente o el Césped muestra necesidad.
- **Primavera**: marzo, abril y mayo, riego día sí, día no.
- **Verano**: junio, julio y agosto, riego diario.
- **Otoño**: 2 veces por semana, interrumpiendo si hay lluvias.

Estas son frecuencias generales pero se puede regar mucho menos. El césped se puede mantener verde y perfectamente estético con un riego más moderado.

3.1.2.1. Momento de regar

Se debe regar las horas centrales del día, cuando hace más calor. Por 3 razones:

- Se pierde más agua por evaporación.
- El viento es mayor, con lo que hay más pérdidas por evaporación y el riego es menos uniforme, es decir, que en unos sitios cae más agua que en otros.
- Se favorece el ataque de hongos.

Los **momentos** para regar son: a **primeras horas** de la mañana o al atardecer. O, con riego automático, también por la noche.

3.1.2.2. Ahorrar agua

Hay que racionalizar la cantidad de agua a aportar en el riego. El agua es un bien escaso que, por lo general, se usa en exceso. La cantidad de agua que se desperdicia en el riego de Parques y Jardines Públicos es, en muchos casos, alarmante. Es necesario un control racional.

Se puede ahorrar agua racionalizando el consumo de la misma en riego haciendo estudios de la necesidad de agua de la planta.

Hay muchas técnicas para ahorrar agua manteniendo al césped en buenas condiciones. Una de las más utilizadas es el intentar ir acostumbrando al césped a vivir con poco agua, regando cada vez menos y observando como se mantiene. Haciéndolo así se consiguen buenos resultados, pero si se pudiera ajustar la cantidad de agua a la que necesita el césped de forma más exacta: teniendo en cuenta la temperatura, la cantidad de luz que le llega, la humedad del suelo… se conseguirían resultados óptimos.

3.2. LÓGICA BORROSA

3.2.1. Introducción

La lógica borrosa (Fuzzy Logic) ha surgido como una herramienta lucrativa para el control de subsistemas y procesos industriales complejos, así como también para la electrónica de entretenimiento y hogar, sistemas de diagnóstico y otros sistemas expertos. Con esta lógica se pretende representar de forma rigurosa el significado de los enunciados imprecisos del lenguaje natural.

Lo central en la lógica borrosa o "fuzzy logic" es que, de modo distinto a la lógica clásica de sistemas, se orienta hacia la modelización de modos de razonamiento imprecisos, los cuales juegan un rol esencial en la destacable habilidad humana de trazar decisiones racionales en un ambiente de incertidumbre e imprecisión. Esta habilidad depende, en cambio, de nuestra habilidad de inferir una respuesta aproximada a preguntas basadas en un conjunto de conocimiento que es inexacto, incompleto o no totalmente confiable" (Zadeh 1988: 1)

Tal y como Zadeh expone en la frase anterior, si la lógica clásica esta considerada como la ciencia de los principios formales y normativos del razonamiento, la lógica borrosa define los principios formales del razonamiento aproximado, siendo capaz de reproducir los modos usuales del razonamiento humano basando la certeza de una proposición en una cuestión de grado. Esta lógica nos permite representar el conocimiento común, el cual será en su mayoría del tipo lingüístico cualitativo y no necesariamente cuantitativo, en un lenguaje matemático a través de la teoría de conjuntos difusos que generaliza la teoría de conjuntos clásica.

Un conjunto borroso [7] μ sobre un universo X asigna un grado de pertenencia a cada elemento en el intervalo [0, 1], y no en el conjunto {0, 1} como los conjuntos clásicos, que pasan a ser un caso particular de conjuntos borrosos.

Las operaciones conjuntistas de unión, intersección y negación de conjuntos borrosos pueden definirse mediante operadores llamados Conorma Triangular o t-Conorma, Norma Triangular, o t-Norma y operadores de negación, generalizando los operadores lógicos OR, AND y NOT [3].

Asimismo, se utilizan operadores borrosos de implicación para definir relaciones borrosas que representan las reglas para hacer inferencia mediante la regla composicional de inferencia. [5], y que deben elegirse apropiadamente para aplicaciones de control [1] [9].

Aunque la lógica borrosa se inventó en Estados Unidos el crecimiento rápido de esta tecnología ha comenzado desde Japón y ahora nuevamente ha alcanzado USA y también Europa. La lógica borrosa es todavía un *boom* en Japón, el número de cartas patentando aplicaciones aumenta exponencialmente. Principalmente se trata de aplicaciones más bien simples de lógica borrosa.

Lo borroso ha llegado a ser una palabra clave para vender. Los artículos electrónicos sin componentes borrosos se están quedando gradualmente desfasados. Como una mordaza, que muestra la popularidad de la lógica borrosa, cada vez es más frecuente un sello con "*fuzzy logic*" impreso sobre el producto.

En Japón la investigación sobre lógica borrosa es apoyada ampliamente con un presupuesto enorme. En Europa y USA se están realizando esfuerzos para alcanzar al tremendo éxito japonés. Por ejemplo, la NASA emplea lógica borrosa para el complejo proceso de maniobras de acoplamiento.

La lógica borrosa es básicamente una lógica multievaluada que permite valores intermedios para poder definir evaluaciones convencionales como sí/no, verdadero/falso, negro/blanco, etc. Las nociones como "más bien caliente" o "poco frío" pueden formularse matemáticamente y ser procesados por computadoras. De esta forma se ha realizado un intento de aplicar una forma más humana de pensar en la programación de computadoras. La lógica borrosa se inició en 1965 por Lotfi Asker Zadeh, profesor de ciencia de computadoras en la Universidad de California en Berkeley.

3.2.2. Conjuntos borrosos

Los elementos fundamentales de la teoría de sistemas borrosos son los conjuntos borrosos (*fuzzy*). Sobre ellos se pueden realizar operaciones, algunas de ellas semejantes a las que se realizan con los conjuntos clásicos. Finalmente, la lógica borrosa está basada en combinaciones de conjuntos difusos mediante las mencionadas operaciones.

Los conjuntos borrosos, son conjuntos en los que se define una función llamada "función de pertenencia", que asigna a cada elemento del conjunto, un valor que representa el grado en que dicho valor pertenece al conjunto. A cada conjunto se le asocia un cuantificador lingüístico, como por ejemplo, mucho, poco, bastante etc., de forma que relacionamos el modelo matemático con el lenguaje humano.

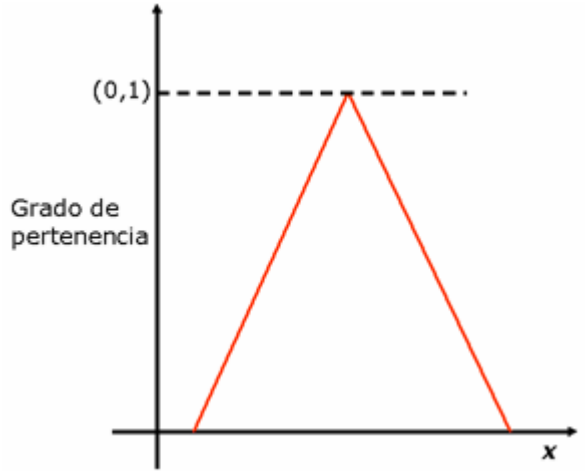
Sea X un universo de discurso. Un conjunto difuso (*fuzzy*) A se define mediante su función de pertenencia:

$$\mu_A : X \to [0,1]$$

Así, dicho conjunto queda identificado mediante parejas de elementos

$$A = \{(x, \mu_A(x)) \mid x \in X\} = \{\mu_A(x) / x \mid x \in X\}$$

donde el símbolo "/" no representa ningún cociente sino que tiene la función de un separador. La forma específica de la función μ_A depende de los objetivos del modelo fuzzy. Por ejemplo, puede ser continua lineal a trozos. Si μ_A es la función característica de A, entonces A es un conjunto clásico (no fuzzy).



Cuando el conjunto soporte es finito, existen notaciones habituales que merece la pena mencionar. Consideremos el universo

$$X = \{x_1, x_2, \ldots, x_n\}$$

El subconjunto difuso A de X se representa por

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \ldots + \frac{\mu_A(x_n)}{x_n}$$

donde interpretamos la barra horizontal como un separador y la suma como una reunión o agregación de elementos.

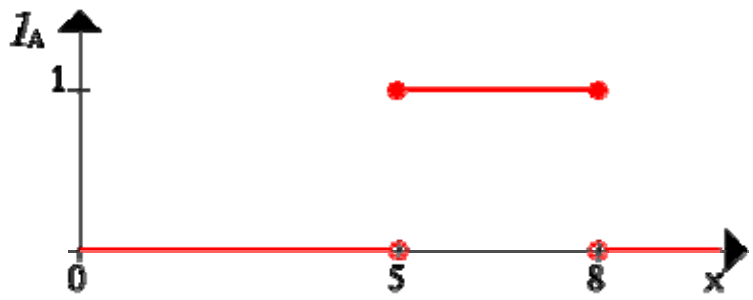
Si el conjunto soporte es un continuo infinito, se utiliza una notación análoga con el símbolo de integración, cuyo significado es el análogo continuo de + en la escritura anterior:

$$A = \int_X \frac{\mu_A(x)}{x}$$

Para entender mejor el concepto de conjuntos borrosos vamos a ver una serie de ejemplos en los que se muestra la diferencia entre el uso de lógica clásica y el de lógica fuzzy.

En primer lugar consideramos un conjunto X con todos los números reales entre 0 y 10 que nosotros llamado el universo de discurso. Ahora, definimos un subconjunto A de X con todos números reales en el rango entre 5 y 8, A = [5,8].

Mostramos el conjunto A por su función característica, es decir, la función que asigna un número 1 o 0 al elemento en X, dependiendo de si el elemento está en el subconjunto A o no. Esto conlleva a la figura siguiente:



Se interpretan los elementos que han asignado el número 1 como los elementos que están en el conjunto A y los elementos que han asignado el número 0 como los elementos que no están en el conjunto A.

Este concepto, que define la lógica clásica, es suficiente para muchas áreas de aplicación. Hay muchas situaciones en las que este concepto carece de flexibilidad. Por ejemplo:

Queremos describir el conjunto de gente joven. Más formalmente, denotamos:

B = {conjunto de gente joven}

Como, en general, la edad comienza en 0, el rango más inferior de este conjunto está claro. El rango superior, por otra parte, es más bien complicado de definir. Como un

primer intento colocamos el rango superiora en, digamos, 20 años. Por lo tanto nosotros definimos B como un intervalo denominado:

B = [0,20]

Ahora la pregunta es: ¿por qué alguien es en su 20 cumpleaños joven y al día siguiente no? Obviamente, este es un problema estructural, porque si movemos el límite superior del rango desde 20 a un punto arbitrario podemos plantear la misma pregunta.

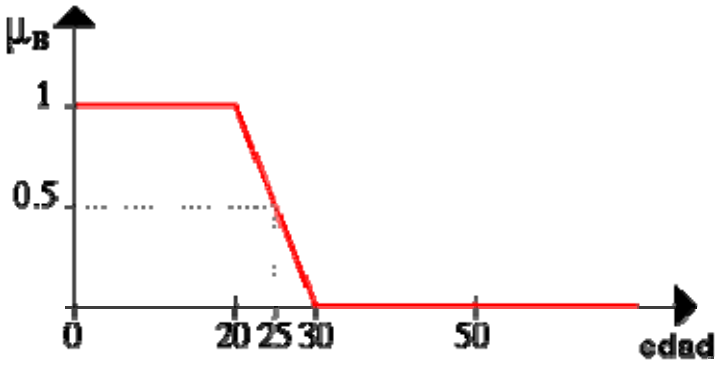
Una manera más natural de construir el conjunto B estaría en suavizar la separación estricta entre el joven y el no joven. Nosotros haremos esto para permitir no solamente la (crispada) decisión "él/ella SI está en el conjunto de gente joven" o "él/ella NO está en el conjunto de gente joven", sino también las frases más flexibles como "él/ella SI pertenece un poquito más al conjunto de gente joven" o "él/ella NO pertenece aproximadamente al conjunto de gente joven".

Pasamos a continuación a mostrar como un conjunto borroso nos permite definir una noción como "él/ella es un poco joven".

En nuestro ejemplo primero codificamos todos los elementos del Universo de Discurso con 0 o 1. Una manera de generalizar este concepto está en permitir más valores entre 0 y 1. De hecho, nosotros permitimos infinitas alternativas entre 0 y 1, denominando el intervalo de unidad Y = [0, 1].

La interpretación de los números ahora asignados a todos los elementos del Universo de Discurso es algo más difícil. Por supuesto, el número 1 asignado a un elemento significa que el elemento está en el conjunto B y 0 significa que el elemento no está definitivamente en el conjunto el B. El resto de valores significan una pertenencia gradual al conjunto B.

Para ser más concretos mostramos ahora gráficamente el conjunto de gente joven de forma similar a nuestro primer ejemplo por su función característica.



De esta forma unos 25 años de edad todavía sería joven al grado de 50 por ciento.

3.2.3. Operaciones con conjuntos borrosos

Dados dos conjuntos difusos

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$
$$B = \{(x, \mu_B(x)) \mid x \in X\}$$

definimos otros conjuntos difusos a través de sus funciones de pertenencia respectivas. Para la unión de A y B,

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$$

y para la intersección,

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$$

Finalmente, la operación de complementación se define por medio de

$$\mu_{\bar{X}-A}(x) = 1 - \mu_A(x)$$

En estas condiciones, queda claro que

$$\mu_{A\cap(X-A)}(x) = \min\{\mu_A(x), 1-\mu_A(x)\}$$

no es, en general, 0, luego

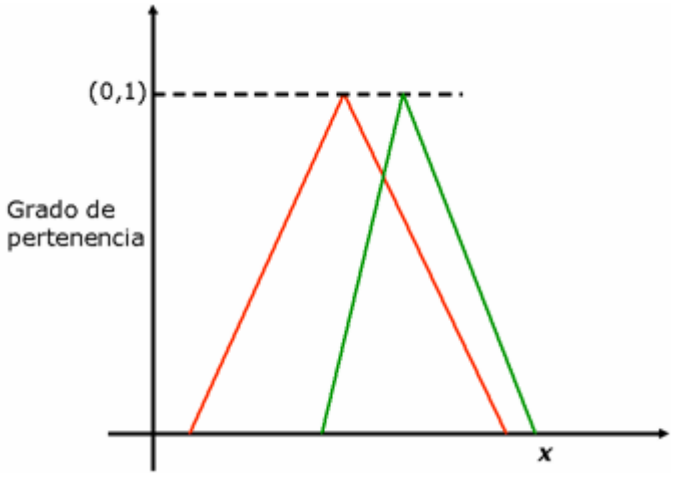
$$A\cap(X-A)\neq\varnothing$$

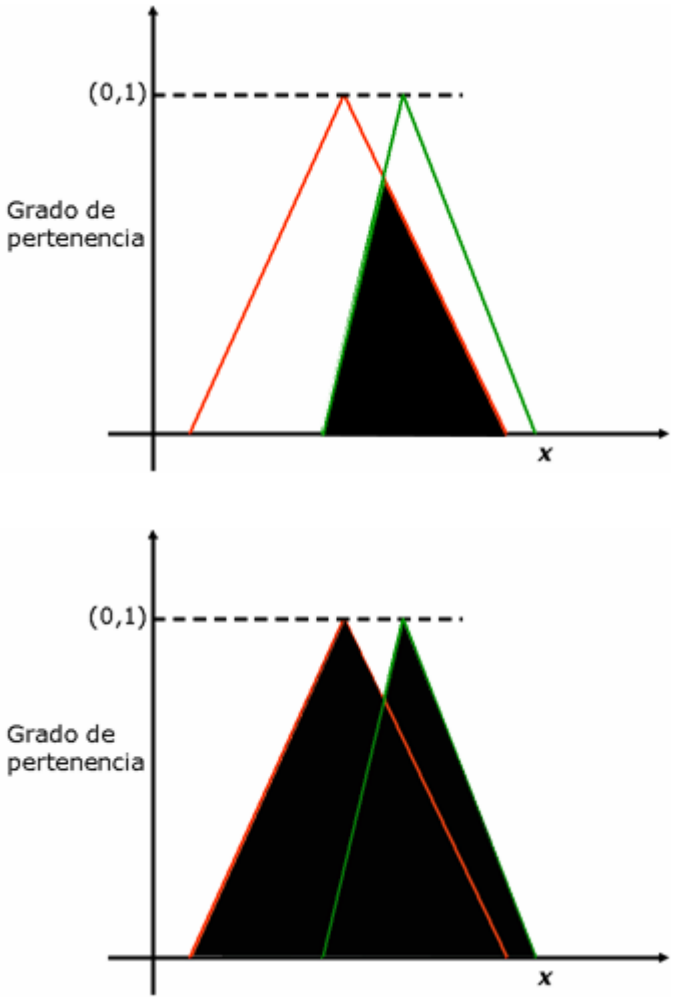
Análogamente,

$$\mu_{A\cup(X-A)}(x) = \max\{\mu_A(x), 1-\mu_A(x)\}$$

no es, en general, idénticamente 1, luego

$$A\cup(X-A)\neq X$$





El mecanismo de las funciones de pertenencia permite también definir la inclusión:

$$A \subset B \Leftrightarrow \mu_A(x) \leq \mu_B(x)$$

y la identidad:

$$A = B \Leftrightarrow \mu_A(x) = \mu_B(x)$$

En uno de sus primeros artículos sobre conjuntos borrosos, L. A. Zadeh sugirió el operador mínimo para la intersección y el operador máximo para la unión de dos conjuntos borrosos. Es fácil ver que estos operadores coinciden con la unificación

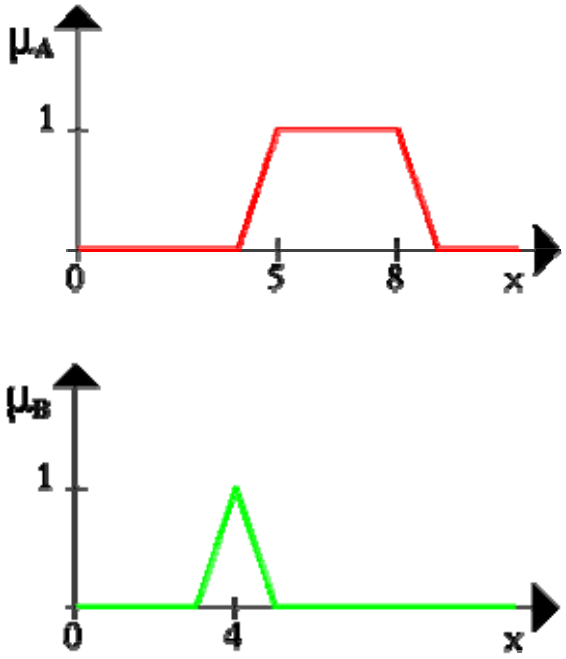
booleana, e intersección si nosotros únicamente consideramos los grados miembros 0 y 1. Así podemos estudiar las diferentes familias de conectivas lógicas.

Una familia de conectivos lógicos [5] borrosos (T, S, N) está formada por una norma triangular T, una conorma triangular S y una negación N, y se denomina una terna de De Morgan o terna de Morgan cuando S es la t-conorma dual de T respecto a la negación N.

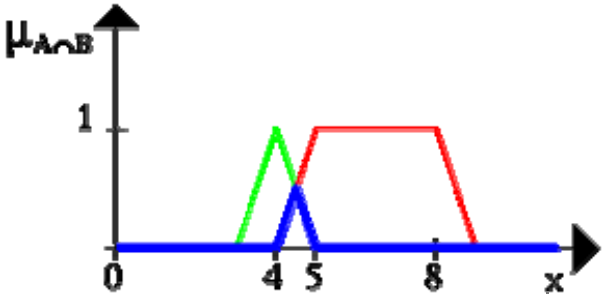
Las familias de conectivos lógicos más utilizadas son:

	T(x, y)	**S(x, y)**	**N(x)**
Zadeh	$Min(x,y)$	$Max(x,y)$	$1-x$
	$x.y$	$x+y-xy$	
$Yager_p$	$1\text{-}Min(((1\text{-}x)^p+(1\text{-}y)^p)^{1/p},1)$	$Min((x^p+y^p)^{1/p},1)$	$(1\text{-}x^p)^{1/p}$
$Dombi_\lambda$ $\lambda>0$	$\frac{1}{1+\left[(\frac{1}{x}-1)^{-\lambda}+(\frac{1}{y}-1)^{-\lambda}\right]^{-1/\lambda}}$	$\frac{1}{1+\left[(\frac{1}{x}-1)^{\lambda}+(\frac{1}{y}-1)^{\lambda}\right]^{1/\lambda}}$	$1-x$
$Weber_\lambda$ $\lambda>-1$	$\text{Max}\left(\frac{x+y-1+\lambda xy}{1+\lambda},0\right)$	$Min\ (x+y+\lambda xy,1)$	$\frac{1-x}{1+\lambda x}$
Łukasie-wicz	Max(x+y-1,0)	Mín{1, x+y}	1-x
$Hamacher_\gamma$ $\gamma>0$	$\frac{xy}{\gamma+(1-\gamma)(x+y-xy)}$	$\frac{x+y-xy-(1-\gamma)xy}{1-(1-\gamma)xy}$	$1-x$

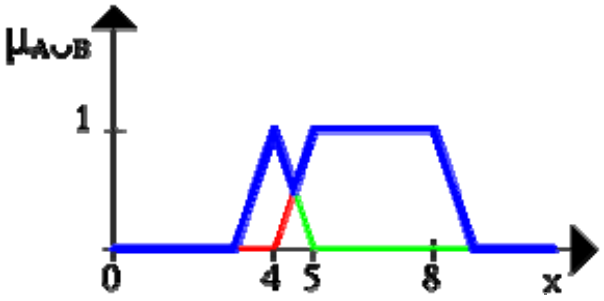
A fin de aclarar esto, mostraremos varios ejemplos, usando la lógica definida por Zadeh. Sea A un intervalo borroso entre 5 y 8, y B un número borroso entorno a 4. Las figuras correspondientes se muestran a continuación:



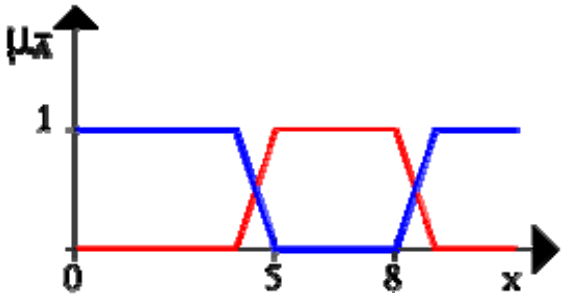
La figura siguiente muestra la operación AND (Y) del conjunto borroso A y el número borroso B (el resultado es la línea azul).



La operación OR (O) del conjunto borroso A con el número borroso B se muestra en la próxima figura (nuevamente, es la línea azul).



Esta figura da un ejemplo para una negación. La línea azul es la NEGACIÓN del conjunto borroso A.



3.2.4. Razonamiento Aproximado

El concepto de Razonamiento Aproximado ("Fuzzy Reasoning") puede ser interpretado como el proceso de obtener, a través de un proceso de inferencia, unas conclusiones imprecisas a partir de unas premisas también imprecisas.

3.2.4.1. Relaciones borrosas

El concepto de relación borrosa procede de una generalización de las relaciones entre los conjuntos abruptos de la Teoría de Conjuntos Clásica. En términos de sistemas expertos, las relaciones son relevantes para expresar la asociación formal entre antecedentes y consecuentes. Una regla como "SI x es A ENTONCES y es B" describe una relación entre las variables x e y. Al igual que en la Teoría de Conjuntos Clásica, estas asociaciones se expresan a través de subconjuntos del Producto Cartesiano (×) entre los dos universos.

3.2.4.2. Mecanismo de inferencia borrosa

El mecanismo de interpretación de una regla se realiza aplicando como Regla de Inferencia la composición supremo-estrella, de manera que si R es una relación borrosa en U×V y *x* es un conjunto borroso en U entonces el conjunto borroso *y* en V que se infiere de la aplicación de la regla viene dado por y = x ° R donde x ° R es la composición supremo-estrella de x y R.

Por su parte la composición supremo-estrella viene definida por:

- *Composición supremo-estrella*: Si R y S son relaciones borrosas en U×V y en V×W respectivamente, la composición de R y S (R ° S) es una relación borrosa definida por:

$$R \circ S = \left\{ \left[(u,w), \sup_{v} \left(\mu_R(u,v) * \mu_S(v,w) \right) \right], u \in U, v \in V, w \in W \right\}$$

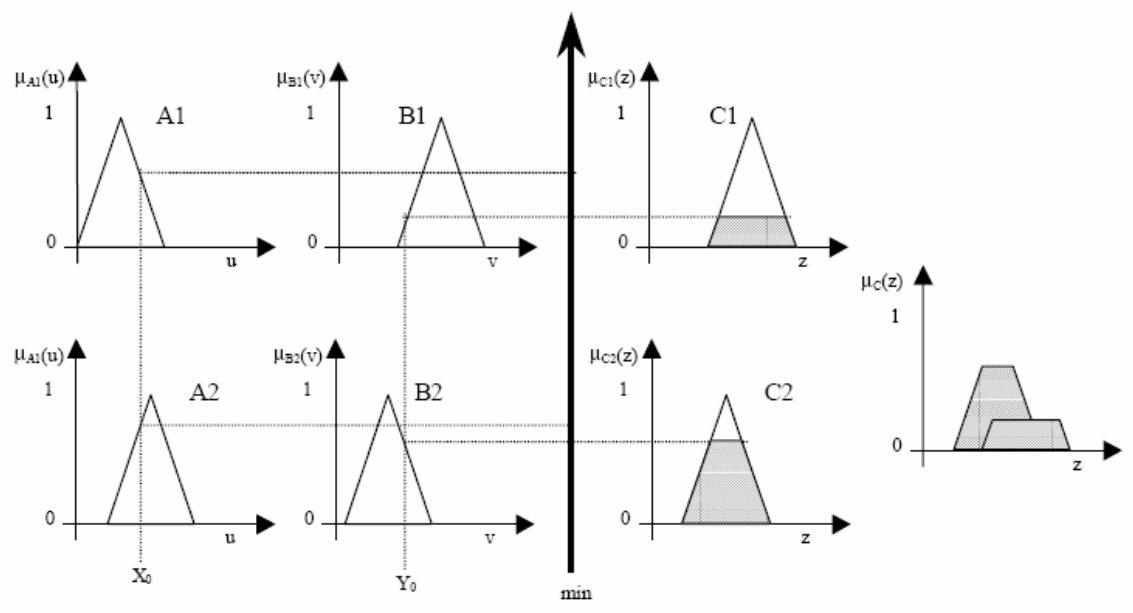
donde * puede ser cualquier operador de la clase de las normas triangulares como el mínimo, producto algebraico, producto acotado o producto drástico.

Un sistema borroso está formado por un conjunto de reglas borrosas, por lo que el comportamiento de todo el sistema vendrá dado por la unión o suma de todas ellas. Por consiguiente, un sistema borroso puede ser caracterizado por una simple relación borrosa que es la combinación de todas las relaciones borrosas del conjunto de reglas:

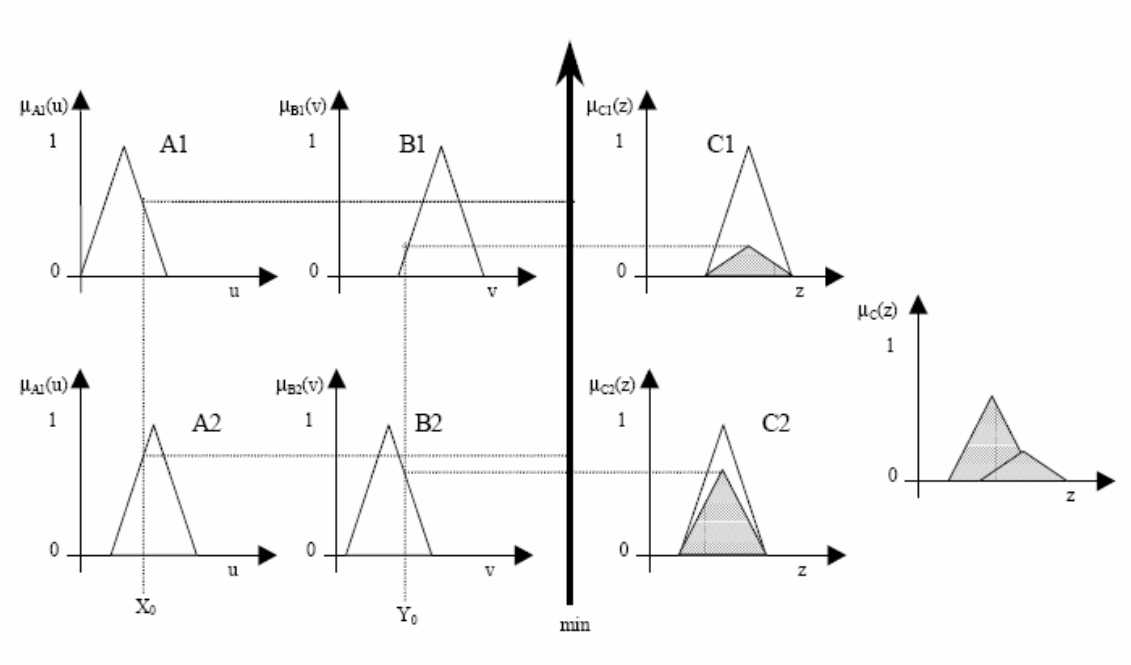
$R = \text{suma}(R_1, R_2, ..., R_i, ..., R_n)$

Los dos procedimientos más utilizados habitualmente son la Inferencia según Mandami (Max-min) y la Inferencia según Larsen (Max-dot):

- En el caso de Mandami se interpreta la unión como el valor máximo, la intersección como el valor mínimo y el operador * como el valor mínimo. Tal y como expone Zadeh en la definición de su lógica para operadores.



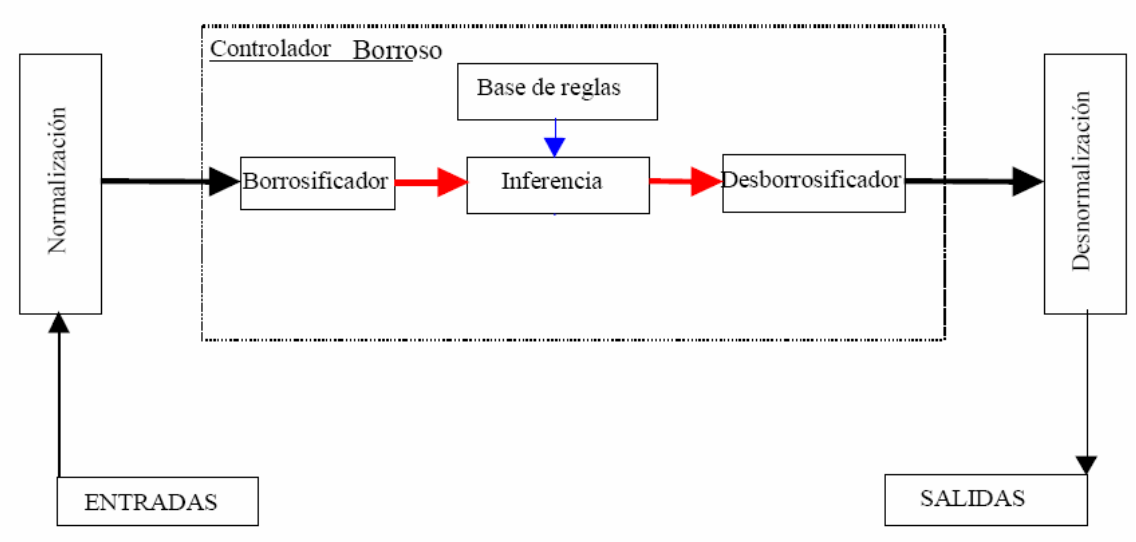
- En el caso de Larsen se interpreta la unión como el valor máximo, la intersección como el mínimo y el operador * como el producto.



3.2.5. Control borroso

Los controladores borrosos trabajan de una forma bastante diferente a los controladores convencionales; el conocimiento experto se usa en vez de ecuaciones diferenciales para describir un sistema. Este conocimiento puede expresarse de una manera muy natural, empleando las variables lingüísticas que son descritas mediante conjuntos borrosos.

El control borroso, permite, en algunos casos, una interpretación por parte del usuario final y una mejor integración con otras etapas superiores en un marco único. En una situación puramente académica, la comparación con estrategias analíticas en situaciones diseñadas para estas últimas (control óptimo, estabilización, etc.) saca a relucir una posible inferioridad teórica del enfoque borroso. Sin embargo, en una situación práctica, las reglas borrosas permiten expresar en lenguaje "natural" otras consideraciones del problema que serían difíciles de modelar (o requerirían mucho más tiempo). Por ejemplo, en un control de vehículos de metro en Sendai, Japón, realizado por Zadeh en los ochenta, se pueden tener en cuenta (aproximadamente) de una forma fácil y rápida consideraciones de confort de los pasajeros, número de los mismos, fuerza de frenado, consumo de electricidad, etc.



En la figura podemos observar la estructura básica de un controlador borroso.

Como se puede observar en la figura, el controlador borroso está constituido principalmente por 4 componentes:

- Borrosificador: Realiza la función de convertir los valores de entrada en los correspondientes valores lingüísticos asociados a cada uno de los conjuntos borrosos. Es decir, proporciona el grado de pertenencia de la variable de entrada a cada una de las variables lingüísticas del sistema.
- Inferencia: Es el componente encargado de proporcionar el valor de salida realizando la evaluación de las reglas que componen el sistema borroso. Es la etapa encargada de aplicar el Razonamiento Aproximado.
- Base de reglas: Contiene el conjunto de reglas lingüísticas de control que caracterizarán los objetivos y la estrategia de control definida por los expertos.
- Desborrosificador: Realiza la función inversa del borrosificador. Proporciona un valor numérico de salida a partir del valor borroso de salida generado por la etapa de inferencia.

Además de estos cuatro componentes que forman el controlador difuso podemos observar que entre los valores de entrada y el controlador existe una etapa de normalización que realiza una adaptación del rango de las variables de entrada a sus correspondientes dominios dentro del controlador (universos de discurso). De la misma manera la etapa de denormalización realiza la adaptación del dominio de las variables de salida del controlador a sus correspondientes dominios físicos.

Parámetros de diseño de un controlador borroso
Los principales parámetros de diseño de un controlador borroso son los siguientes:

- Estrategia de borrosificación.
- Bases de reglas
- Inferencia.
- Estrategia de desborrosificiación.

3.2.5.1. Estrategia de borrosificación

El proceso de borrosificar está relacionado con las vaguedades e imprecisiones del lenguaje natural, es decir, se trata de realizar una valoración completamente subjetiva que transformará un valor medido en un valor subjetivo, a partir de realizar un mapeado entre el espacio de entrada observado y unos conjuntos borrosos en un determinado universo de discurso de la entrada.

En la definición de los conjuntos borrosos es muy importante el conocimiento del problema que posea el experto, aunque es importante también tener en cuenta las siguientes reglas:

- Si el número de conjuntos borrosos definidos sobre la variable lingüística es elevado entonces obtendremos una gran resolución, pero el coste computacional también será alto.
- Las funciones de pertenencia más utilizadas generalmente son las triangulares y trapezoidales dada su menor complejidad de implementación que, por ejemplo, las funciones gaussianas.

En general, la densidad de conjuntos difusos cerca del punto de equilibrio del sistema a controlar suele ser mayor que en los extremos, lo que permite realizar un control más ajustado.

3.2.5.2. Base de reglas

El comportamiento dinámico de un sistema borroso está caracterizado por un conjunto de normas lingüísticas basadas en el conocimiento del experto. Estos conocimientos se expresan con sentencias del tipo:
SI condiciones_de_entrada ENTONCES acción_a_ejecutar

Dependiendo de la estructura del controlador el conjunto de condiciones a satisfacer pueden una o varias al igual que el conjunto de acciones_a_ejecutar.

El conjunto de condiciones a satisfacer siempre está asociado a conceptos borrosos, mientras que el conjunto de acciones_a_tomar puede estar asociado de dos formas diferentes:

1. Asociado según Mandami

Cada uno de los consecuentes de las reglas está formado por un conjunto borroso.

2. Asociado según Sugeno

Cada uno de los consecuentes de las reglas contiene una ecuación con una combinación de las variables de entrada.

3.2.5.3. Inferencia

El proceso de inferencia realizará la evaluación de cada una de las reglas del sistema.
Se puede escoger el tipo de inferencia a realizar decidiendo qué tipo de operador utiliza para realizar cada una de las operaciones aplicables a los conjuntos difusos. Las dos estrategias de inferencia más utilizadas son la de Mandani (Max-min) y la de Larsen (Max-dot).

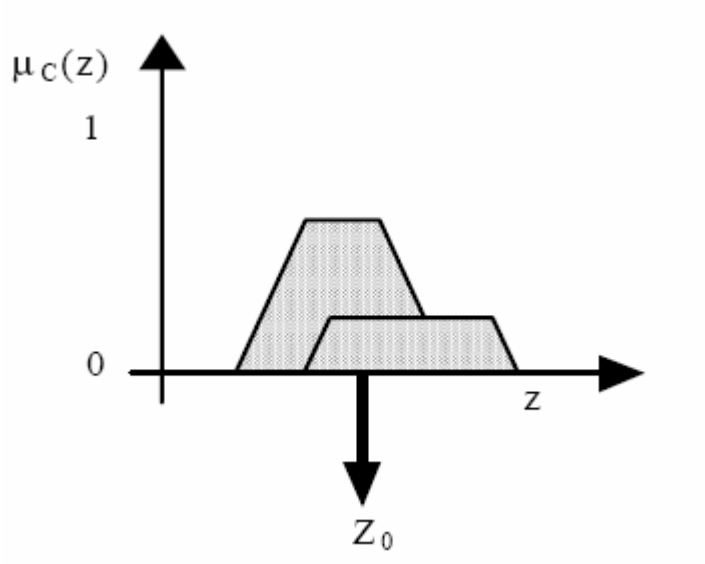
3.2.5.4. Desborrosificación

Como ya hemos visto anteriormente, la etapa de desborrosificación es la última etapa del controlador borroso y la encargada de generar un valor no-borroso a partir del valor borroso generado en la etapa de inferencia.

Los métodos más habituales de desborrosificación son:

- Centro de los máximos (CoM).
- Centro de Gravedad (CoG) o Centro de Área (CoA).
- Mediana de los máximos (MoM).

El método más utilizado es el Centro de Gravedad o de Área. Como su nombre indica, este método determina el centro de gravedad del área generada por la combinación de todos los valores de la salida.



La salida del sistema viene en este caso determinada por la siguiente operación:

$$Z_0 = \frac{\sum_{i=1}^{N} x_i \cdot \mu_x(x_i)}{\sum_{i=1}^{N} \mu_x(x_i)}$$

en la que N representa el número de valores borrosos involucrados en la salida.

¿Cuándo el control borroso es recomendable?

- Para procesos muy complejos, cuando no hay un modelo matemático simple.
- Para procesos altamente no lineales.
- Si el procesamiento del (lingüísticamente formulado) conocimiento experto puede ser desempeñado.

¿Cuándo el control borroso no es recomendable?

- El control convencional teóricamente rinde un resultado satisfactorio.
- Existe un modelo matemático fácilmente soluble y adecuado.
- El problema no es soluble.

4. SISTEMA DE RIEGO INTELIGENTE

4.1. Justificación del uso de un sistema de riego inteligente con control borroso

4.1.1. Estimación humana del tiempo de riego

Determinar las necesidades hídricas de las plantas de un jardín resulta básico en la organización del riego: la cantidad de agua que las plantas pierden por evotranspiración (ET). Conociendo las características de humedad del suelo local, la eficiencia del riego y la ET estimada para la plantación, un profesional de la jardinería o del riego puede desarrollar un plan efectivo de riego.

Las tasas de evotranspiración han sido establecidas para céspedes y ciertos cultivos agrícolas. La relación ETc = Kc * ETo se utiliza para calcular la tasa de ET de un cultivo específico (su ETc) cuando dos factores, el coeficiente de cultivo (Kc) y la evotranspiración de referencia (ETo) son conocidas. Si no se conoce Kc, se puede calcular a partir de la ETc y de la ETo (Kc = ETc + ETo). ETc se suele determinar a partir del balance de entradas y pérdidas de agua. Los experimentos se desarrollan en parcelas de campo bien controladas. Los valores para la ETo se suelen obtener a partir de tablas oficiales. Una vez se ha establecido la Kc para una planta en particular se puede utilizar la ETo para calcular su ETc.

La cantidad de agua perdida por un jardín a causa de la ET varía en función de la especie plantada, la densidad de la vegetación y las condiciones microclimáticas.

Evaluando cada factor y asignándole un valor numérico, podemos estimar cuanta agua puede perderse en relación a la evaporación de referencia. No obstante, establecer si un jardín es de elevada o de baja densidad es subjetivo. Por otra parte las condiciones

ambientales pueden variar considerablemente en un jardín. Las estructuras o pavimentaciones típicas de los jardines urbanos pueden influir considerablemente en las temperaturas foliares y del aire, el viento y la humedad. Por ejemplo, los árboles de las zonas de aparcamiento están sometidos a mayor temperatura y menor humedad que los árboles de los parques. Con lo cual este sistema no consigue los mejores resultados posibles puesto que es estático y en cierta forma subjetivo.

Aunque estos cálculos consiguen resultados bastante buenos, los responsables del riego deberán basarse en su propia experiencia y en manuales para determinar la categoría de una especie determinada.

El método del coeficiente del jardín proporciona a los responsables de jardinería y del riego una estimación de la cantidad de agua necesaria para mantener un jardín de calidad aceptable. Este método puede resultar también útil en la estimación previa de los costes de los arquitectos paisajistas, diseñadores y organizadores. A medida que se obtenga más información sobre el uso del agua por las plantas de jardín, la influencia del microclima y de la densidad, se podrá ajustar y refinar el método. Por el momento, este método sirve como punto de referencia práctico para desarrollar sistemas de riego efectivos y eficientes. [11]

4.1.2. Comparación con sistemas inteligentes

Tal y como se muestra en apartado anteriores, no sólo es importante la necesidad de agua de la planta sino también la cantidad de agua utilizada así como su optimización, manteniendo a la planta en buenas condiciones.

Observando el método anterior y su forma de aplicación, se puede ver que aunque se pueda conseguir estimar bastante bien la necesidad de riego e incluso la cantidad de agua utilizada, los resultados no son óptimos para la planta ni para el ahorro de agua puesto que las determinaciones tomadas son estáticas y cualquier cambio en las condiciones

significarían tener que volver a realizar el estudio. Cualquier otro método de riego humano requiere de un estudio similar a éste y es igualmente estático con las desventajas que esto supone.

Sin embargo, un sistema que, una vez ajustado respondiera dinámicamente a los cambios en las condiciones (las condiciones que más cambian como la temperatura o la humedad, ya que otras condiciones como el tipo de suelo o la inclinación del mismo, aunque son muy importantes son fijas) optimizaría tanto el uso de agua como la cantidad que le llega a la planta.

Un **sistema de riego inteligente** necesita un estudio similar al explicado anteriormente para ser realizado (en el caso de un sistema pequeño y específico) o configurado (en el caso de un sistema grande y general). En cualquier caso, una vez terminado, el sistema funcionaría de forma automática respondiendo a los cambios de forma prácticamente instantánea y sin necesidad de repetir el estudio.

En el caso de realizar un riego estimado mediante un profesional utilizando un sistema que se programe por días y minutos en lugar de un sistema inteligente, no se conseguiría optimizar el riego porque el sistema al ser fijo no podría responder a cambios inesperados como la lluvia, una subida grande de temperatura, una evaporación rápida del agua por el viento … Sin embargo un sistema inteligente con sensores es capaz de responder instantáneamente ante estos cambios manteniendo a la planta en las mejores condiciones y ahorrando agua.

Un sistema de riego inteligente recogerá, en el momento de regar, los datos necesarios del terreno y tomará una decisión del tiempo de riego acorde con ello. De esta forma se tendrá una estimación de las necesidades hídricas muy realista (dependiendo, por supuesto, de la calidad del sistema). Además como el sistema normalmente revisará su decisión durante el riego, se podrá modificar el tiempo determinado de forma dinámica con las ventajas obvias que esto supone, cosa que cualquier otro sistema no puede conseguir, ya que una vez decidido el tiempo se riega hasta completarlo.

En cualquier caso, siempre será necesario contar con expertos para realizar el sistema, configurarlo, controlar su buen funcionamiento y actualizarlo (con técnicas de aprendizaje automático se podrían minimizar las actuaciones del experto).

4.1.3. Sistema de riego inteligente con control borroso

Teniendo en cuenta la situación actual del mundo con respecto al agua, es necesario conseguir un riego eficiente y efectivo de los jardines y cultivos.

Como se ha explicado antes, aunque un sistema de riego basado en estudios que se programe y se deje fijo puede ser bastante bueno no consigue mantener a la planta en perfecto estado con un consumo mínimo de agua.

En este contexto se comprende la necesidad de un sistema automático que consiga una optimización del agua utilizada en el riego. Para ello, la mejor alternativa es el control borroso puesto que las condiciones determinantes del tiempo de riego no tienen valores nítidos sino valores pertenecientes a conjuntos. Por ejemplo, según la nubosidad la planta tendrá más o menos percepción solar lo que se traduce en necesidad de agua. La nubosidad no se puede medir como un número, es más natural medirla de forma difusa: muy nublado, poco nublado…

Como las variables de entrada al sistema de riego inteligente son en su mayor parte difusas, lo más adecuado es utilizar un sistema de control borroso para realizar los razonamientos e inferencias.

Los sistemas basados en lógica difusa imitan la forma en que toman decisiones los humanos, con la ventaja de ser mucho más rápidos. Estos sistemas son generalmente robustos y tolerantes a imprecisiones y ruidos en los datos de entrada.

Utilizando un sistema de riego inteligente con control borroso se consiguen las siguientes ventajas, además de las ya nombradas:

- Eficiencia: Ofrece salidas de una forma veloz y precisa, disminuyendo así las transiciones de estados fundamentales en el entorno físico que controla. Por ejemplo: si se determinara un tiempo de riego asociado a cada temperatura, los pequeños cambios en la misma provocarían continuos cálculos y cambios en la salida. Al ser un sistema borroso y moverse en conjuntos borrosos, los pequeños cambios dentro del mismo conjunto mantendrían estable la salida y mejoraría la eficiencia.
- Simplicidad: Al utilizar conceptos parecidos a los usados por los humanos al razonar, se puede conseguir que las reglas sean sencillas de forma que la comunicación con el experto y las modificaciones de las mismas se pueden realizar fácilmente.
- Estabilidad de los resultados: Capacidad de adelantarse en el tiempo a los acontecimientos, estabilizando siempre el entorno físico que controla.

En conclusión: para conseguir utilizar el mínimo de agua posible al regar un jardín o cultivo y mantenerlo en perfectas condiciones sin tener que estar haciendo cambios continuos, programarlo, vigilar su funcionamiento, etc., la mejor opción es un sistema de riego inteligente con control borroso.

5. DESARROLLO DEL SISTEMA

Para nuestro desarrollo utilizaremos una mezcla de las fases del ciclo de vida de un sistema basado en conocimiento con las fases típicas del desarrollo de software (análisis, diseño, implementación y pruebas).

5.1. Análisis

En esta parte del desarrollo damos los primeros pasos: definimos el sistema, identificamos características, decidimos los pasos a dar y realizamos ciertas etapas relacionadas con la obtención de conocimiento.

5.1.1. Definición del sistema

La **función** de la aplicación será ajustar el riego de una zona a unas determinadas condiciones externas.

Nuestro sistema obtendrá unas determinadas **características externas** (humedad, temperatura...) en ciertos rangos y decidirá la forma de riego (tiempo de apertura de los aspersores) basándose en las **reglas** obtenidas a partir de un **estudio del conocimiento** sobre el tema.

5.1.2. Características principales del sistema

Por las características del problema, el sistema es un **sistema basado en conocimiento (SBC)**. La complicación de su realización estriba principalmente en la gestión de conocimiento: adquisición, conceptualización, organización... y en su formalización en reglas.

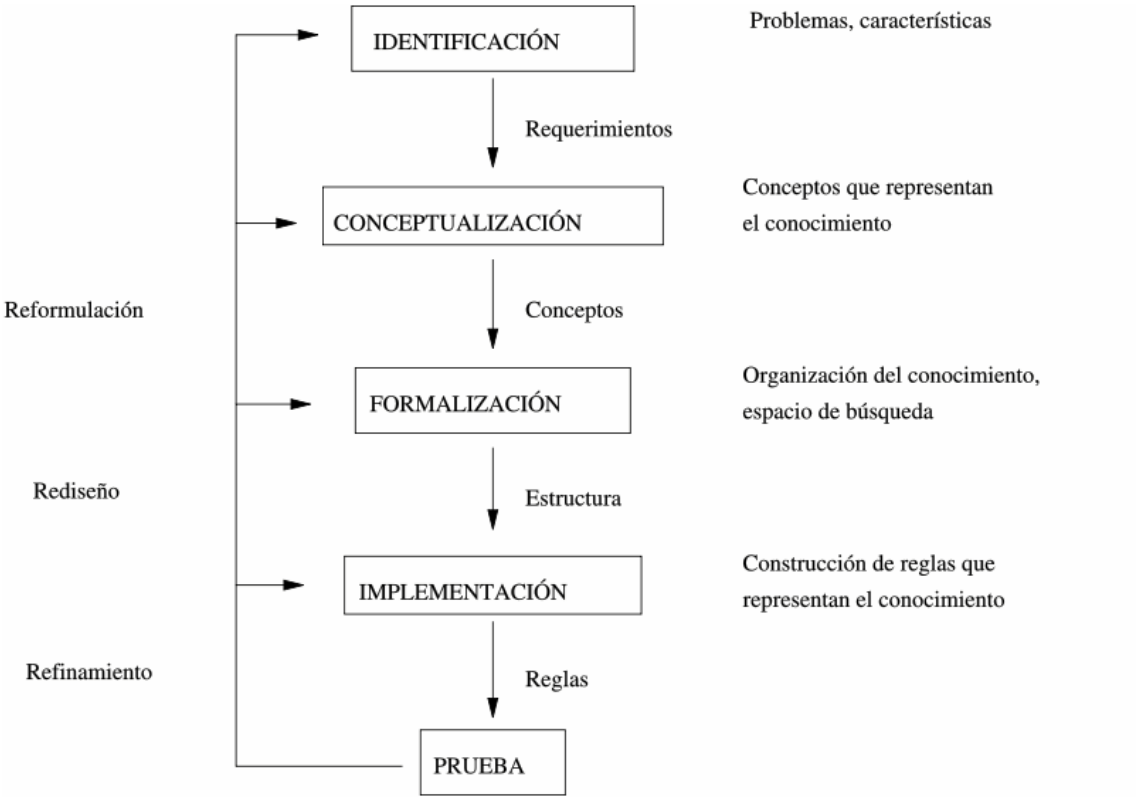
Se utiliza **lógica borrosa**, ya que los valores de las entradas y la salida se moverán en conjuntos imprecisos (poca humedad, bastante humedad, mucha humedad,...), de forma

que se podrán controlar los matices ajustándose, así, a la realidad de forma más precisa. Esta característica implica **complicación adicional**.

El sistema tiene una **simulación gráfica** en la que se podrán hacer cambios externos y observar la respuesta del mismo. La implantación real con sensores no es posible por problemas de presupuesto.

5.1.3. Pasos a seguir en el desarrollo del sistema

El desarrollo de un sistema experto se explica en el siguiente cuadro:



En nuestro caso utilizaremos estas fases junto con las típicas del desarrollo de software para guiarnos en el desarrollo. También utilizaremos un proceso basado en prototipos en el que realizaremos un sistema sencillo completo que funcione y lo iremos ampliando hasta conseguir el sistema total. De esta forma siempre tendremos un sistema funcionando y las ampliaciones serán más sencillas.

5.1.4. Estudio del conocimiento sobre riego

A continuación, procedemos a realizar las fases necesarias para obtener y formalizar el conocimiento necesario para el desarrollo de nuestro sistema

5.1.4.1. Identificación

El problema al que nos enfrentamos, desde el punto de vista de adquisición de conocimiento, es el de encontrar las variables de entrada y de salida con respecto a las cuales formular las reglas que determinan las respuestas del sistema ante ciertas condiciones.

5.1.4.2. Adquisición de conocimiento

Para adquirir el conocimiento lo que se suele hacer es entrevistar a un **experto** en el tema. Después de la entrevista, en la que se toman notas, se agrupa y organiza el conocimiento obtenido de forma que se pueda utilizar por el sistema.

Entrevista con el experto

Lo primero que hicimos antes de la entrevista con el experto fue aprender un poco por nuestra cuenta sobre plantas y riego de forma que el experto pudiera exponer su conocimiento más fácilmente sin tener que entretenerse en explicar conceptos básicos.

Nos entrevistamos con D. Raúl Sánchez, **profesor de Hidráulica y Riegos** de la E.T.S.I Agrónomos de la UPM (Universidad Politécnica de Madrid), el cual poseía mucho conocimiento teórico sobre las plantas y su forma de obtener el agua, así sobre cómo administrársela correctamente.

Aunque, como es usual en estas entrevistas, nos proporcionó mucho más conocimiento del necesario, alejándose en algunas ocasiones del tema principal, conseguimos unas notas bastante completas con la información que necesitábamos.

Por recomendación del experto, estudiamos las siguientes **variables** a tener en cuenta:

- Radiación Solar -> Incidencia de rayos solares
- Nubosidad -> Hace variar la incidencia de los rayos solares
- Humedad relativa -> % de agua en la tierra

- Superficie foliar de la planta -> Para conocer crecimiento planta
- Viento -> Varía la evaporación y transpiración de la planta
- Precipitación -> Para controlar el riego
- Permeabilidad del suelo -> Permite control del encharcamiento
- Calidad del agua -> Según condiciones químicas
- Presión y caudal de riego -> Se suponen constantes.

Durante el diseño del sistema decidimos cuales de estos parámetros eran más importantes. Por ejemplo, como se toma la humedad del suelo, no es necesario tener en cuenta el viento puesto que sus efectos se notan principalmente en la evaporación de agua y podemos obtener esa modificación mediante la humedad.

5.1.4.3. Conceptualización

Como se ha explicado en apartados anteriores, el sistema se está realizando de forma incremental. El conocimiento se va añadiendo poco a poco según va creciendo el programa y de esa forma se va documentando.

Para la ***primera versión***, el objetivo era realizar un sistema pequeño con todas las características del sistema total. Para ello, se buscó en la información obtenida de las entrevistas, el conjunto de variables mínimo pero identificativo.

Para la **salida** se escogió el tiempo de apertura de los aspersores, puesto que el ángulo de apertura no es suficiente, según nos explicó el experto, ya que es necesario que llegue el mismo caudal de agua a todas las plantas a todas las distancias. La presión tampoco se puede utilizar para la salida por la misma razón.

Para la **entrada** después de agrupar el conocimiento y estructurarlo, decidimos las siguientes variables:

- **Temperatura** considerando la temperatura que se tomaría de un termómetro colocado en algún lugar del cultivo / jardín.

- **Humedad** considerando la humedad del suelo, donde de implementarse en un sistema real habría sensores, para tener en cuenta el agua absorbida por la /s planta /s y poder regular el riego.
- **Percepción solar** con esta variable se pretende simular la cantidad de sol que le llega a la planta. Sería necesario tener en cuenta la nubosidad, la estación..., pero para esta primera versión no se toman aún estos valores sino que se toma la percepción solar como una variable similar a las dos anteriores sin preocuparse de cómo se obtendría en la realidad.

Para la ***segunda versión***, después de volver a estudiar las entrevistas y ampliar un poco más el conocimiento con libros y viendo otros sistemas inteligentes se decidió introducir **otro motor de inferencia** (como en el caso de un sistema que dirigen automáticamente vehículos, el cual utiliza varios motores de inferencia para razonar sobre distintas direcciones y luego los une en uno que dará la decisión final) para calcular la **percepción solar** ya que depende de nuevos valores obtenidos.

Se mantiene la **salida** ya que sigue pareciendo la más adecuada a todos los aspectos.

Para la **entrada**, entonces, las variables son:

Para el *sistema general* se mantienen las anteriores:

- Temperatura.
- Humedad del suelo.
- Percepción solar.

Las dos primeras se obtienen directamente y la tercera se obtiene a partir de la salida de otro motor de inferencia el cual tiene como entradas:

- ✓ Estación: la estación del año en que se encuentra
- ✓ Nubosidad: la nubosidad en el momento de tomar los valores.

Con estas mejoras el sistema es más preciso y responde mejor a los cambios. Además, se consigue una mayor ***modularidad*** con lo que es más fácil depurarlo, modificarlo y

ampliarlo (si se adquiere mayor conocimiento sobre la percepción solar, no es necesario modificar las reglas generales sólo las de ese motor de inferencia).

5.1.4.4. Formalización

Para la formalización del conocimiento obtenido se utilizaron un conjunto de **reglas** con lógica borrosa, ya que los conceptos utilizados no son nítidos, es decir, se mueven en torno a un rango de valores y no a valores binarios (uno y cero) como la lógica clásica.

5.2. Diseño

En esta etapa del desarrollo definimos de forma más concreta el sistema, decidimos las herramientas a utilizar y las explicamos, exponemos los patrones de diseño utilizados, etc.

5.2.1. Identificación de las partes del sistema

La aplicación consta de dos partes principales:

- **El sistema de reglas**. Implementa el controlador *fuzzy* que toma las decisiones según las entradas.
- **La interfaz Java**. Para poder mostrar el sistema de forma más sencilla, como no tenemos la posibilidad de hacer una implementación hardware, la única manera de hacerlo es mediante una simulación software.

5.2.2. Especificación de las partes del sistema

El **sistema de reglas** ha sido diseñado utilizando la información y las decisiones realizadas en la fase anterior (estudio del conocimiento). Para su implementación se utilizará el XFUZZY.

Para la **interfaz software** se ha utilizado el patrón MVC que se explica a continuación. Para su implementación se ha utilizado Java (JRE 1.5).

5.2.2.1. Patrón de Diseño Modelo Vista-Controlador

Introducción

Modelo-Vista-Controlador (MVC) es un **patrón de diseño** que separa la interfaz de una aplicación, los datos de la misma y la lógica de control.

Los patrones son una disciplina de resolución de problemas reciente en la ingeniería del software que ha emergido en mayor medida de la comunidad de orientación a objetos, aunque pueden ser aplicados en cualquier ámbito de la informática y las ciencias en general.

Los patrones de software facilitan la **reutilización** del diseño y de la arquitectura, capturando las estructuras estáticas y dinámicas de colaboración de soluciones exitosas a problemas recurrentes que surgen al construir aplicaciones.

Una definición clara de patrón es la que se observa al comienzo del libro "Pattern Oriented Software Architecture, Volume 1" [Buschmann96]:

"*Los patrones le ayudan a construir sobre la experiencia colectiva de ingenieros de software experimentados. Estos capturan la experiencia existente y que ha demostrado ser exitosa en el desarrollo de software, y ayudan a promover las buenas prácticas de diseño. Cada patrón aborda un problema específico y recurrente en el diseño o implementación de un sistema de software.*"

Los patrones de diseño o más comúnmente conocidos como "**Design Patterns**" son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos basadas en la experiencia.

Un **patrón de diseño** describe una estructura recurrente de componentes que se comunican para resolver un problema general de diseño en un contexto particular. Nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. Identifica las clases e instancias participantes, sus roles y colaboraciones y la distribución de responsabilidades.

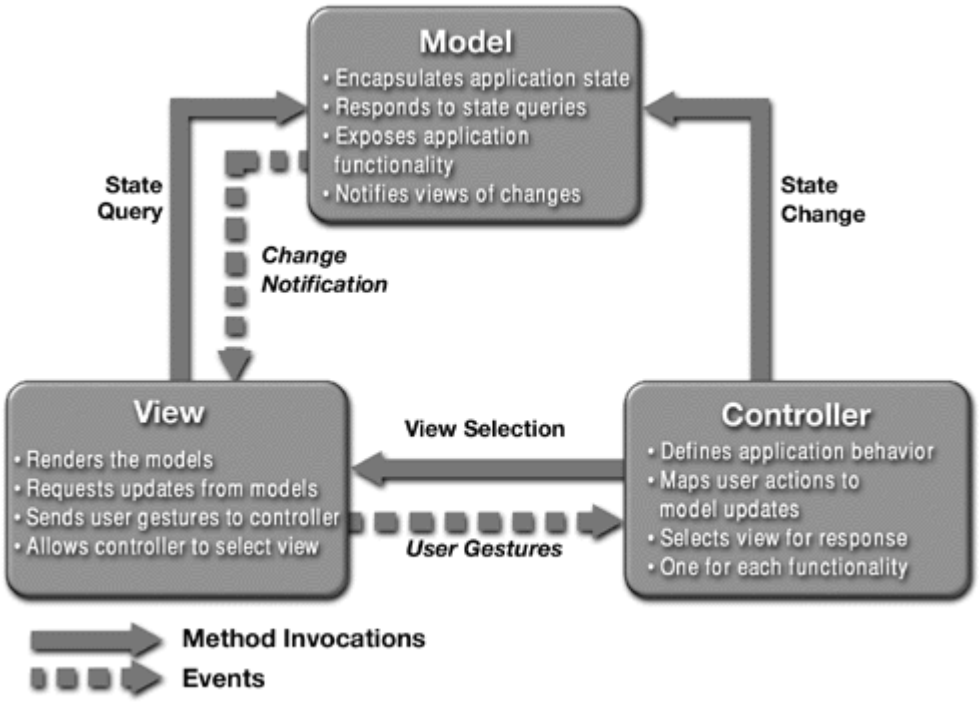
El patrón MVC

El **Model-View-Controller** (Modelo-Vista-Controlador o MVC) fue introducido inicialmente en la comunidad de desarrolladores de Smalltalk-80.

El **patrón MVC** es un patrón que define la organización independiente del Modelo (Objetos de Negocio), la Vista (interfaz con el usuario u otro sistema) y el Controlador (controlador del flujo de la aplicación).

MVC divide una aplicación interactiva en 3 áreas: procesamiento, salida y control. Para esto, utiliza las siguientes abstracciones:

- **Modelo** (Model): Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada. Representa los datos.
- **Vista** (View): Muestra la información al usuario. Pueden existir múltiples vistas del modelo.
- **Controlador** (Controller): Reciben las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio ("service requests") para el modelo o la vista.



Si el objetivo es desacoplar los objetos dominio (Modelo), la forma de mostrarlos (Vista) y los manejadores (Controlador), a fin de brindar soporte a una mayor reutilización de los objetos del dominio y reducir al mínimo el impacto que los cambios de la interfaz y los manejadores tienen en ellos este patrón proporciona numerosas ventajas a tal efecto. Definir las clases dominio (Modelo) para que no tengan acoplamiento ni visibilidad directa respecto a las clases ventana (Vista) y para que los datos de la aplicación y de la funcionalidad se conserven en las clases de dominio, no en las de ventana. Definir las clases manejadores (Controlador) para que procesen los eventos (peticiones) al sistema y redireccionen a las clases dominio y ventana tanto el procesamiento como la visualización de resultados respectivamente.

El patrón MVC se suele **utilizar** principalmente en aplicaciones web, ya que en ellas es frecuente cambiar la vista manteniendo la lógica del sistema y el control de flujo. Además en este tipo de desarrollos, suele ocurrir que las personas que se encargan de la vista no tengan conocimientos muy amplios de programación y con éste patrón se pueden encargar sólo de la vista sin preocuparse de los demás, con lo cual ayuda a la organización de equipos en los desarrollos y al trabajo en paralelo.

Ventajas del patrón MVC

- Menor acoplamiento.
  a. Desacopla las vistas de los modelos.
  b. Desacopla los modelos de la forma en que se muestran e ingresan los datos.
- Mayor cohesión.
  a. Cada elemento del patrón esta altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).
- Las vistas proveen mayor flexibilidad y agilidad.
  a. Se puede crear múltiples vistas de un modelo.
  b. Las vistas pueden anidarse.
  c. Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
  d. Se puede sincronizar las vistas.
  e. Las vistas pueden concentrarse en diferentes aspectos del modelo.
- Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales
  a. Una vista para cada dispositivo que puede variar según sus capacidades.
  b. Una vista para la Web y otra para aplicaciones de escritorio.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad.

El patrón MVC en nuestra aplicación

Para **diseñar las clases de nuestro sistema de riego** nos hemos basado en el patrón MVC. No lo hemos aplicado estrictamente pero sí nos hemos ajustado a sus condiciones en la medida que nos ha sido posible.

En nuestra aplicación las **vistas** son las pestañas (PestañaDia, PestañaMes y PestañaInstantánea). En ellas se realiza la comunicación con el usuario: se recogen las entradas y se muestran las salidas.

El **modelo** son funciones implementadas principalmente en las clases generadas por el Xfuzzy (paquete lógica) que son los que realizan la inferencia: el verdadero trabajo con los datos, también hay funciones del modelo en la vista y una pequeña parte en el controlador.

El **controlador** es el hilo que se ejecuta periódicamente y que llama a las funciones del modelo para trabajar con los datos. En las pestañas diaria e instantánea ocurre de esta manera. En la pestaña mensual el control se encuentra en sí misma puesto que su funcionalidad es muy sencilla (mostrar resultados de un mes) y no hay mucha interacción con el usuario.

Se puede ejemplificar la aplicación de este patrón de la siguiente forma: Se muestra inicialmente la pestaña instantánea con todas las opciones para realizar su funcionalidad (**vista**). Periódicamente se llama al *hilo* (**controlador**) que se encarga de tomar los datos que se han introducido por el usuario. Este controlador llamará a las funciones del **modelo** que obtendrán los resultados y repintará la vista para mostrar los cambios del sistema.

Como se puede observar es el hilo (controlador) el que realiza el control de que se hace con los datos que llegan (a que función del modelo se llama) y que se pinta (que se muestra en la vista). [10]

5.3. Implementación

En esta parte se explica la realización de la implementación física del sistema.

5.3.1. Implementación del controlador borroso. XFUZZY

Para implementar el controlador borroso mediante reglas se ha utilizado el XFUZZY. Explicamos a continuación su funcionamiento y cómo lo hemos aplicado.

5.3.1.1. Introducción

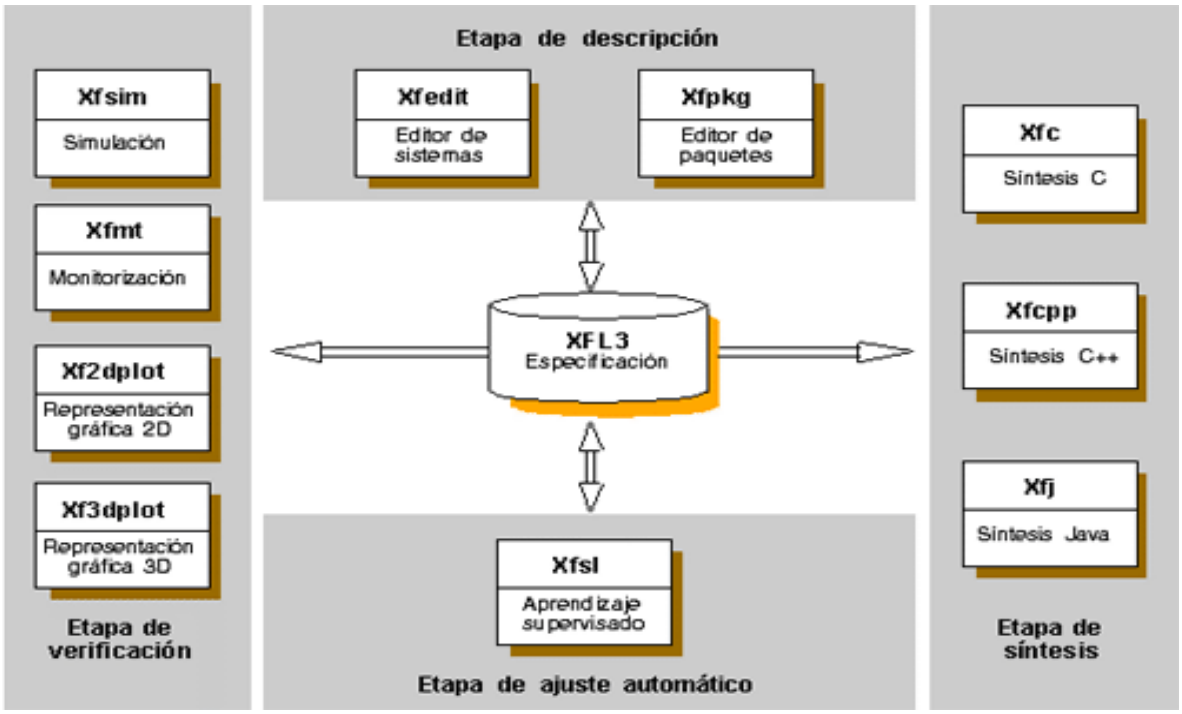
XFUZZY es un entorno de desarrollo para sistemas de inferencia basados en lógica borrosa. El entorno está compuesto por un amplio conjunto de herramientas que cubren las diferentes etapas del diseño de estos sistemas. Estas herramientas comparten una descripción común en un lenguaje de descripción formal llamado XFL3. Las principales novedades de esta versión son el uso de este nuevo lenguaje, más potente y flexible que en versiones anteriores, la inclusión de nuevas herramientas, y la posibilidad de ejecutar el entorno en cualquier sistema operativo, ya que ha sido programado totalmente en lenguaje Java.

Características principales:

- Facilita el diseño de un sistema borroso desde su descripción hasta su implementación.
- Permite el desarrollo de sistemas complejos
- Flexibilidad para permitir al usuario extender el conjunto de funciones disponibles

Herramientas:

- Descripción
- Verificación
- Síntesis
- Definición de operadores
- Generación de gráficas 2D y 3D
- Aprendizaje



La etapa de descripción incluye herramientas gráficas para la definición del sistema difuso. La etapa de verificación está compuesta por herramientas de simulación, monitorización y representación gráfica del comportamiento del sistema. La etapa de ajuste facilita la aplicación de algoritmos de aprendizaje. Finalmente, la etapa de síntesis incluye herramientas para generar descripciones en lenguajes de alto nivel para implementaciones software o hardware.

El nexo entre todas las herramientas es el uso de un lenguaje de especificación común, XFL3, que extiende las capacidades de XFL, el lenguaje definido en la versión 2.0 de Xfuzzy. XFL3 es un lenguaje flexible y potente, que permite expresar relaciones muy complejas entre variables difusas por medio de bases de reglas jerárquicas y conectivas, modificadores lingüísticos, funciones de pertenencia y métodos de defuzzyficación definidos por el usuario.

Las diferentes herramientas pueden ser ejecutadas como programas independientes. El entorno integra a todas ellas bajo una interfaz grafica que facilita el proceso de diseño.

5.3.1.2. Implementación de nuestro sistema mediante XFuzzy

Dada la complejidad de la modelación se opta por considerar tanto para variables de salida como de entrada conjuntos borrosos:

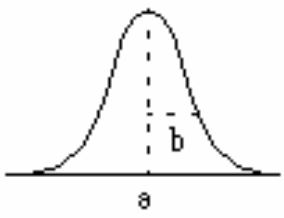
ENTRADAS DEL SISTEMA

- Temperatura ambiente (en grados)
- Humedad relativa de la tierra (en %)
- Estación del año (para obtener radiación solar)
- Nubosidad (hace variar la radiación solar)

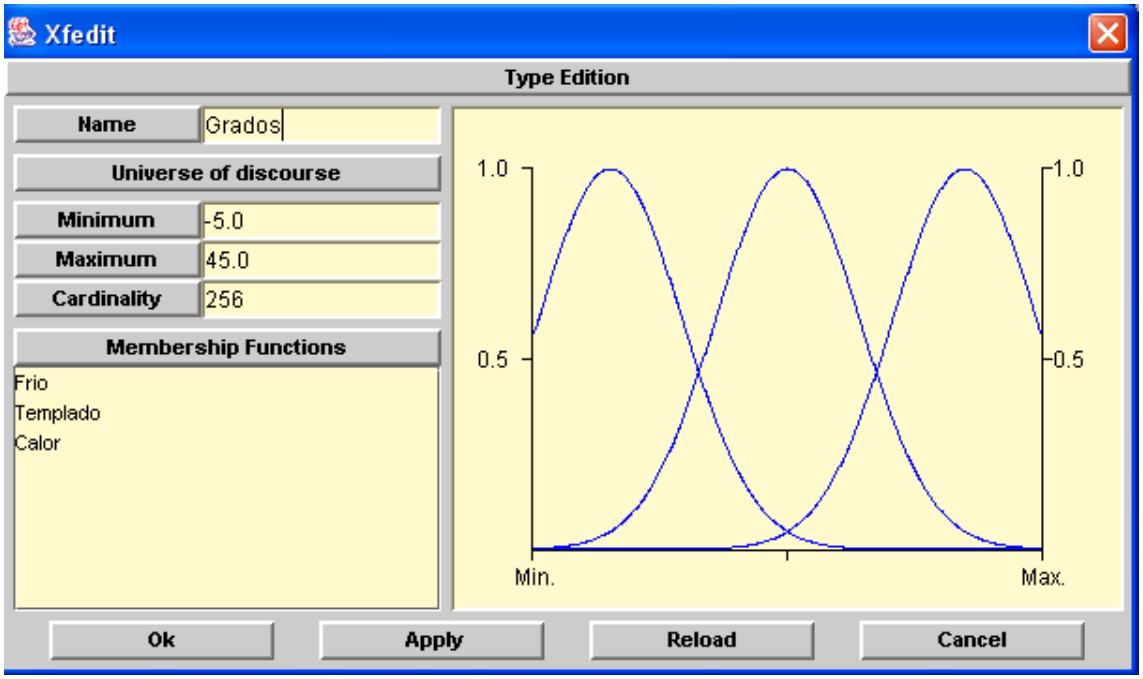
SALIDAS DEL SISTEMA

- Tiempo de riego (caudal constante)

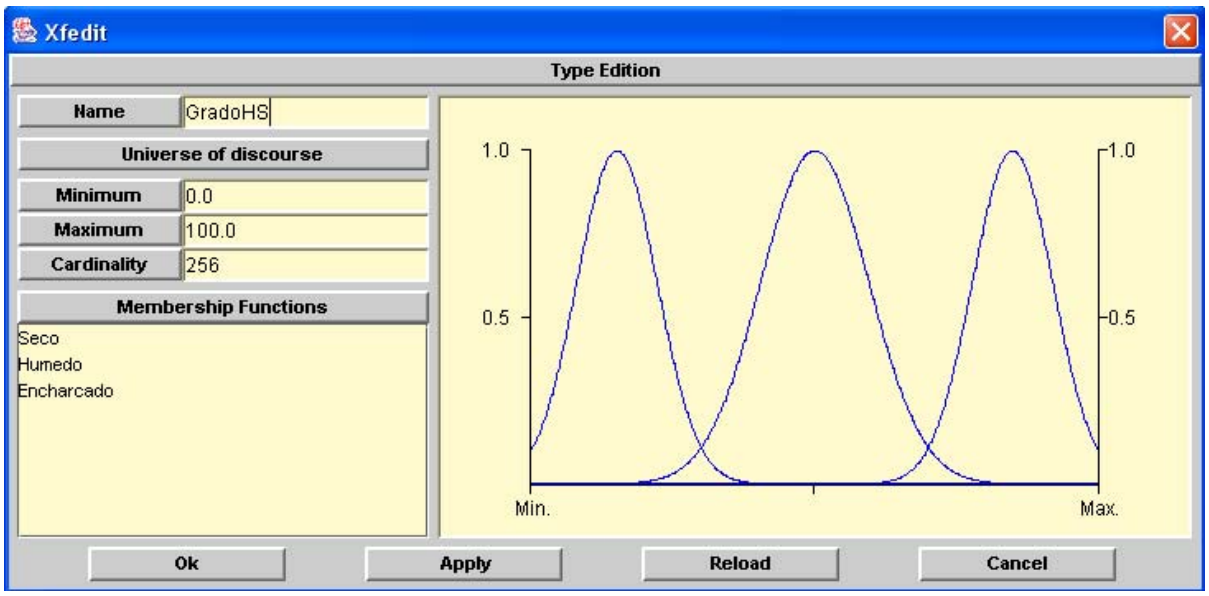
Los tipos de cada una de las variables son *fuzzy-sets*. Estos conjuntos borrosos contienen funciones de pertenencia con forma de campana uniformemente distribuidas a lo largo del universo de discurso:



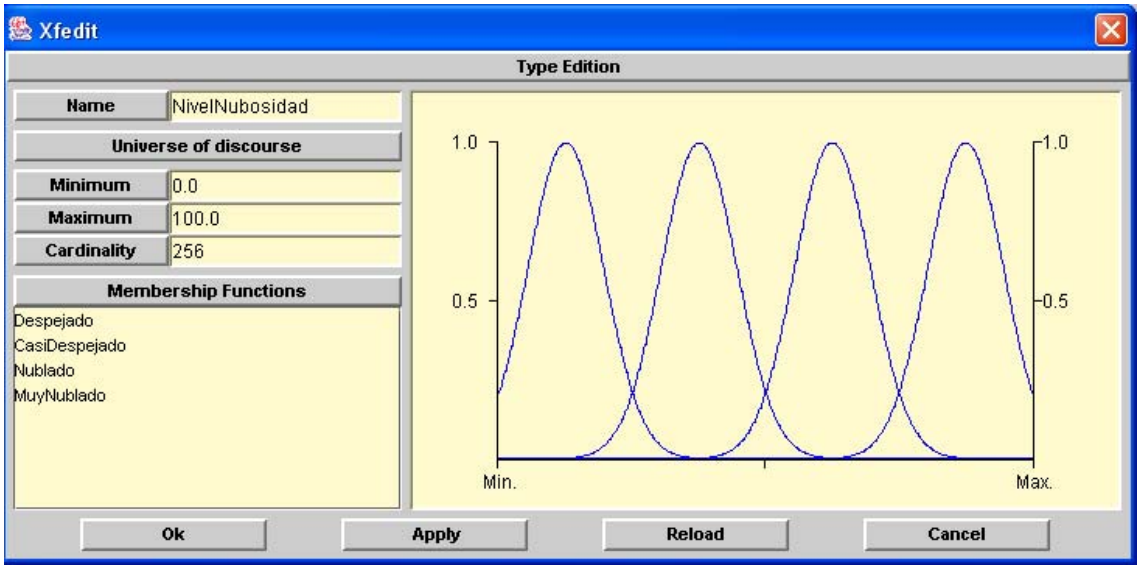
En el caso de la temperatura se medirá en grados, cuyos valores oscilan entre –5 y 45, definiendo así los diferentes conjuntos borrosos: fría, templado y calor.



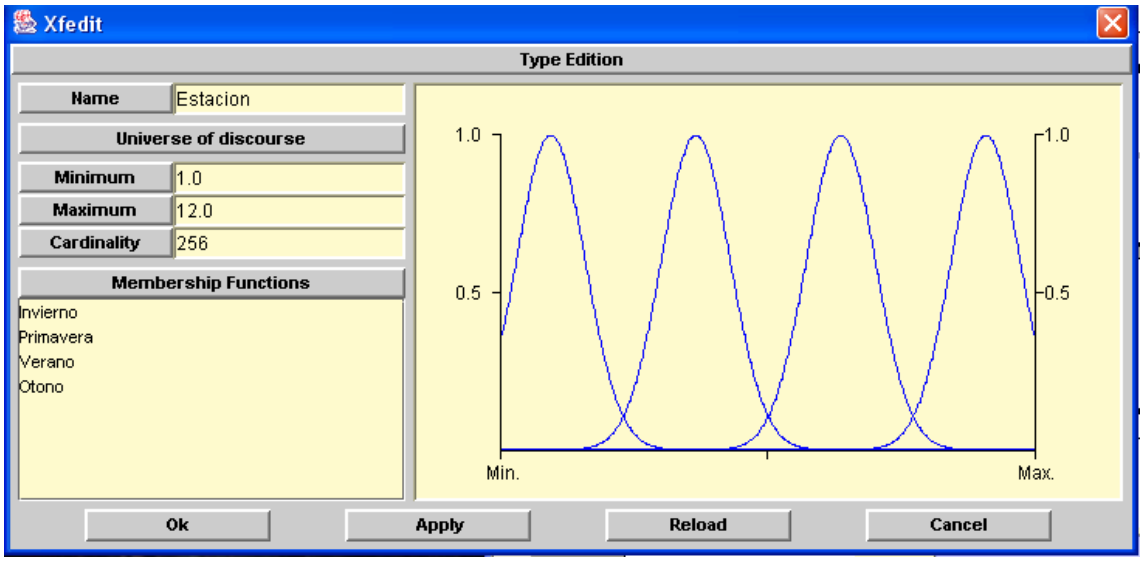
La humedad relativa del suelo y la nubosidad se definirán a través de porcentajes. Para la humedad relativa definimos los conjuntos: seco, húmedo y encharcado.



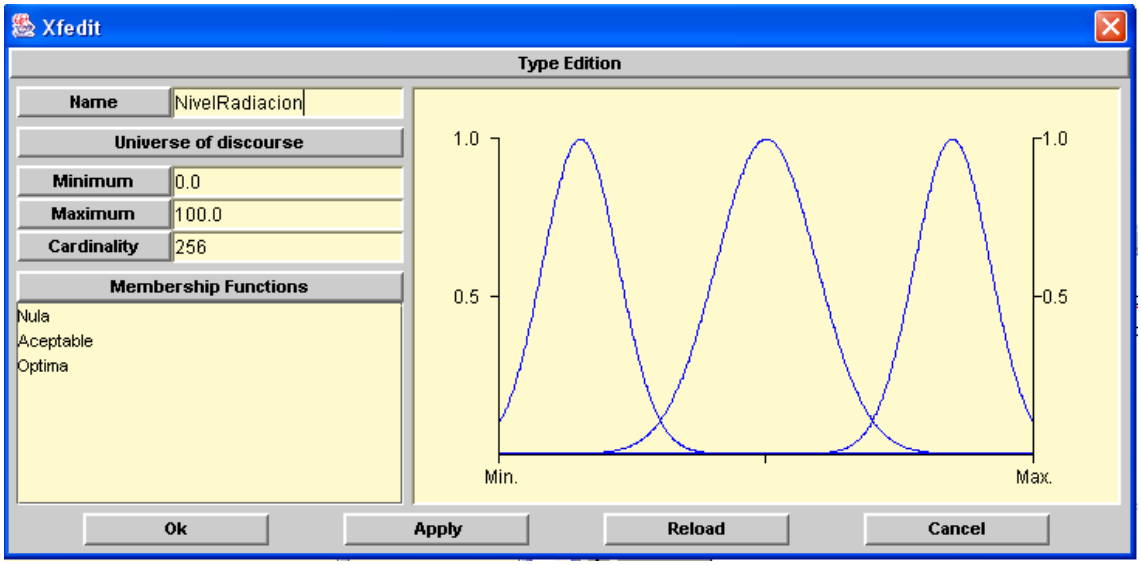
La nubosidad la definimos como el nivel de nubosidad que hay ese día, pudiendo ser: despejado, casiDespejado, Nublado y MuyNublado.



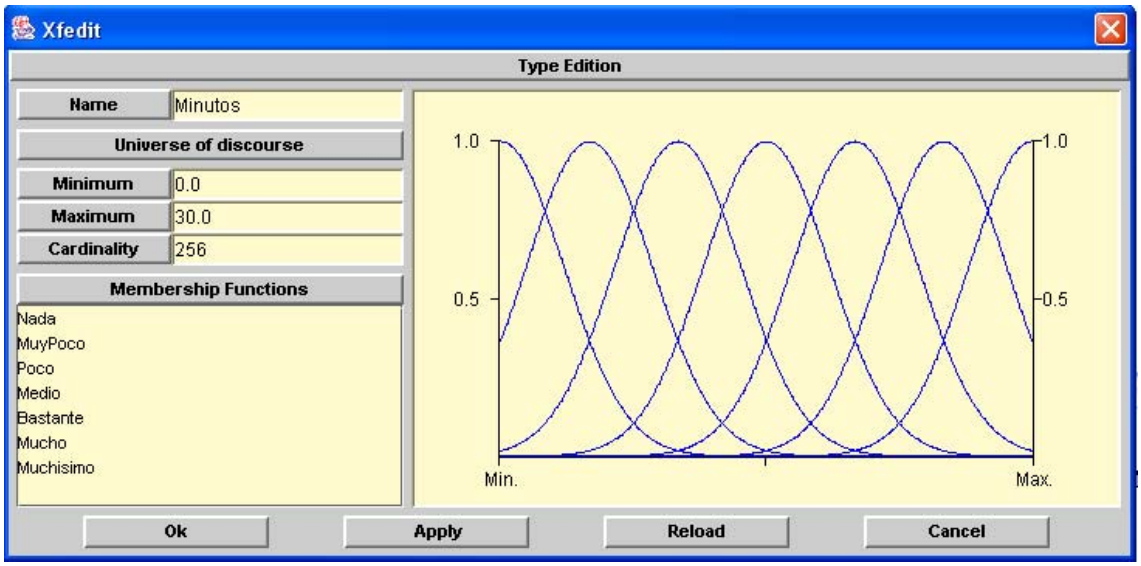
La estación la clasificará el mes en el que estemos, teniendo definidos como conjuntos borrosos las cuatro estaciones:



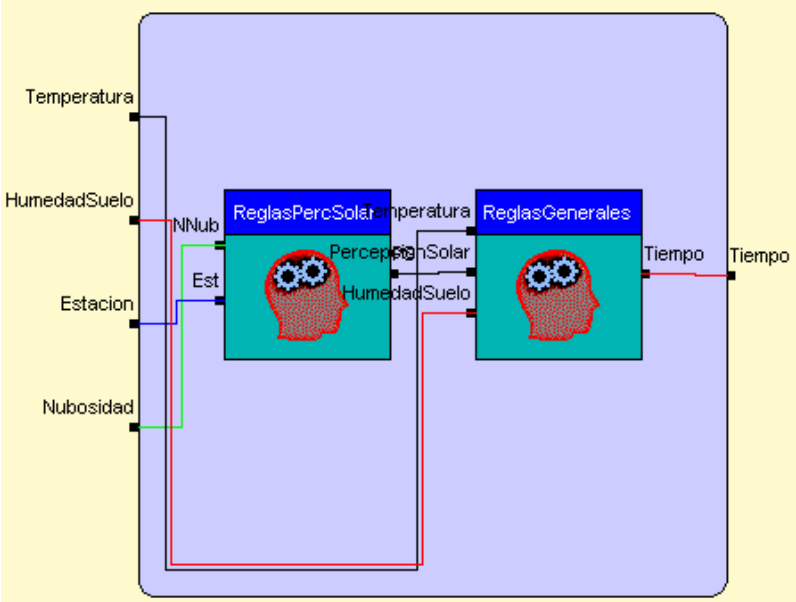
Del motor de inferencia ReglasPercSolar obtenemos la variable de entrada, la percepción solar. Ésta se define con cuatro conjuntos borrosos que describen el nivel de radiación: nula, aceptable y óptima.



El tiempo de regadío se define en intervalos de tiempo, minutos. El tiempo de riego va de entre cero y treinta minutos, pudiendo ser, Nada, MuyPoco, Poco, Medio, Bastante, Mucho y Muchísimo.



Una vez definidos los conjuntos borrosos comenzamos el diseño de los dos motores de inferencia, uno para la variable percepción solar, y otro para la variable final de salida, tiempo. La inferencia se realiza en base a un sistema de reglas definido. La variable percepción solar se introduce en el motor general de la aplicación como variable de entrada fuzzy que proviene de otro motor.

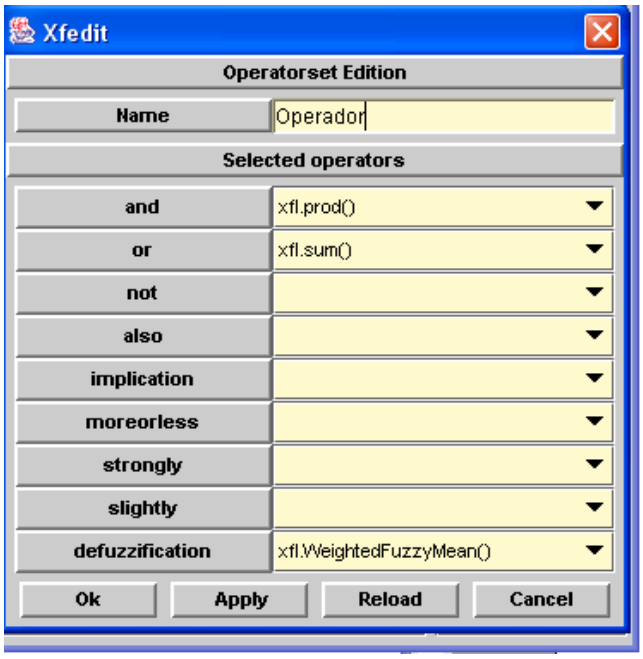


El sistema de reglas se modela con reglas del tipo IF THEN ELSE. De esta forma obtenemos el comportamiento deseado para este sistema. Con estas reglas realizamos la inferencia del sistema y ajustamos el comportamiento del controlador.

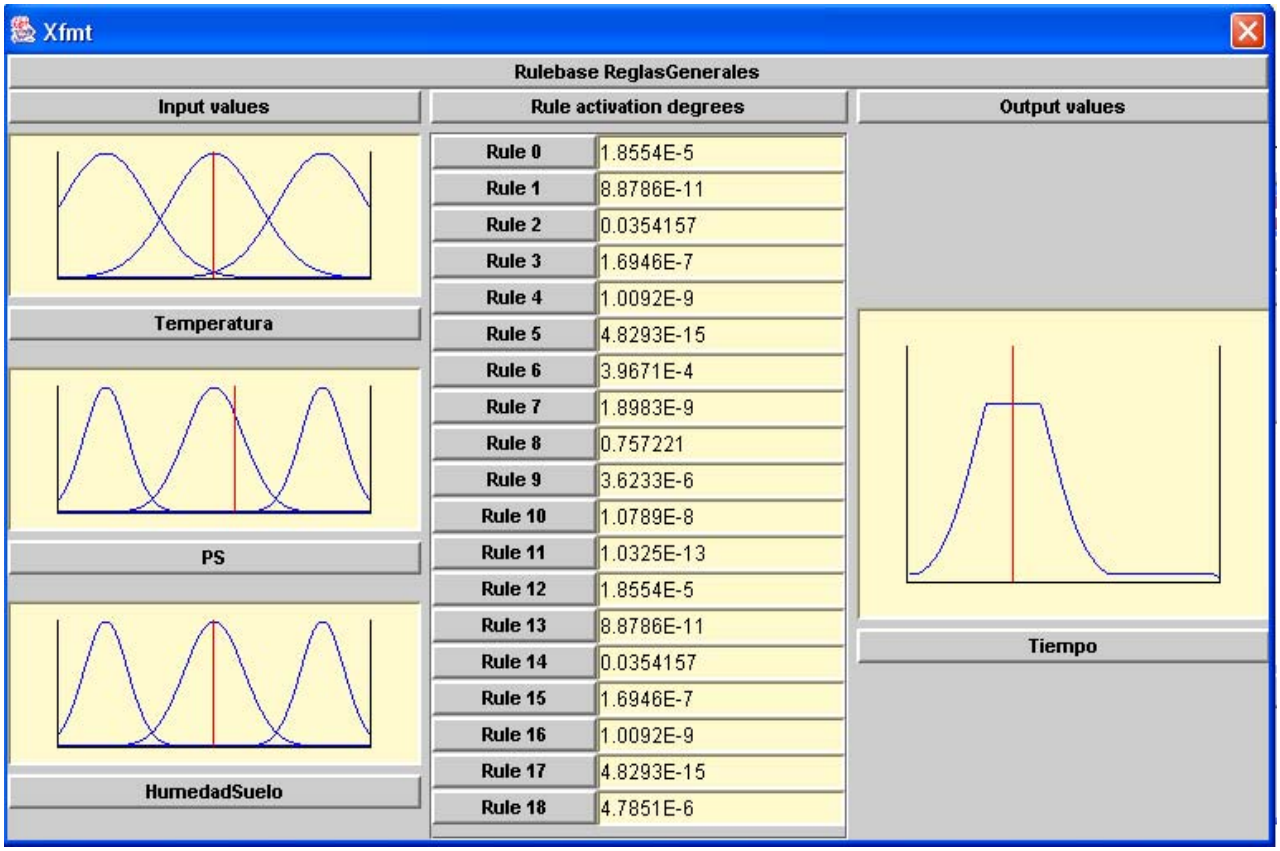
Rule			Premise		Conclusion
0	1.0	if	(Temperatura <= Frio & PS >= Optima & HumedadSuelo ...	->	Tiempo = Poco
1	1.0	if	(Temperatura <= Frio & PS >= Optima & HumedadSuelo ...	->	Tiempo = Medio
2	1.0	if	(Temperatura <= Frio & PS == Aceptable & HumedadSu...	->	Tiempo = MuyPoco
3	1.0	if	(Temperatura <= Frio & PS == Aceptable & HumedadSu...	->	Tiempo = Poco
4	1.0	if	(Temperatura <= Frio & PS <= Nula & HumedadSuelo ==...	->	Tiempo = Nada
5	1.0	if	(Temperatura <= Frio & PS <= Nula & HumedadSuelo <=...	->	Tiempo = MuyPoco
6	1.0	if	(Temperatura == Templado & PS >= Optima & Humedad...	->	Tiempo = Medio
7	1.0	if	(Temperatura == Templado & PS >= Optima & Humedad...	->	Tiempo = Bastante
8	1.0	if	(Temperatura == Templado & PS == Aceptable & Hume...	->	Tiempo = Poco
9	1.0	if	(Temperatura == Templado & PS == Aceptable & Hume...	->	Tiempo = Medio
10	0.5	if	(Temperatura == Templado & PS <= Nula & HumedadSu...	->	Tiempo = MuyPoco
11	1.0	if	(Temperatura == Templado & PS <= Nula & HumedadSu...	->	Tiempo = Poco
12	1.0	if	(Temperatura >= Calor & PS >= Optima & HumedadSuel...	->	Tiempo = Mucho
13	1.0	if	(Temperatura >= Calor & PS >= Optima & HumedadSuel...	->	Tiempo = Muchisimo
14	1.0	if	(Temperatura >= Calor & PS == Aceptable & HumedadS...	->	Tiempo = Bastante
15	1.0	if	(Temperatura >= Calor & PS == Aceptable & HumedadS...	->	Tiempo = Mucho
16	1.0	if	(Temperatura >= Calor & PS <= Nula & HumedadSuelo =...	->	Tiempo = Medio
17	1.0	if	(Temperatura >= Calor & PS <= Nula & HumedadSuelo <...	->	Tiempo = Bastante
18	1.0	if	(HumedadSuelo >= Encharcado)	->	Tiempo = Nada
*					

En estas reglas se ve el uso de los diferentes operadores, ya que dependiendo de su definición los resultados de cada regla variarán.

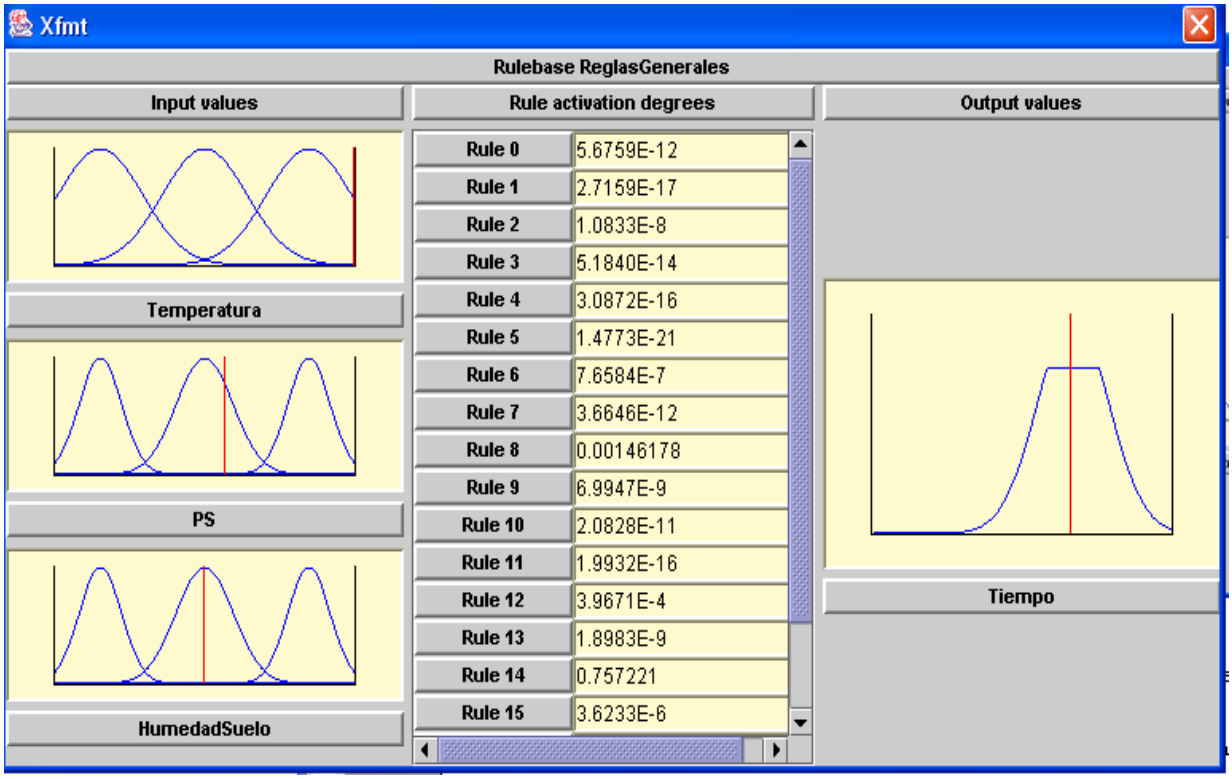
Por ejemplo, en el caso de la lógica del producto, la definición de los operadores y del tipo defuzzyficación tendría este aspecto:



Al ejecutar las reglas con unas determinadas variables de entrada y la definición de una de las lógicas, vemos que la evaluación de las reglas varía.

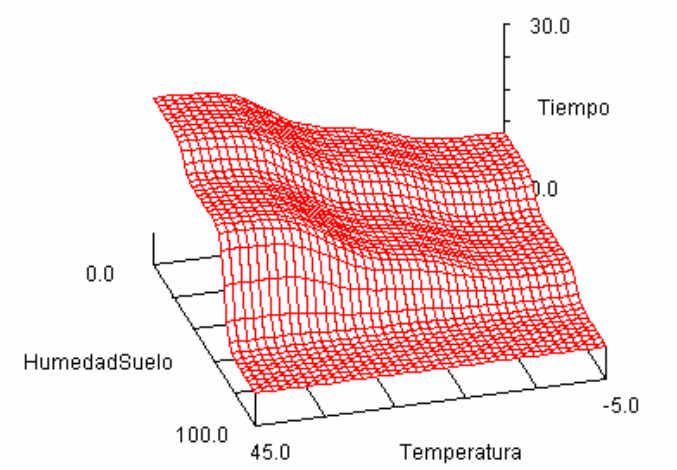


Para cada regla se muestra el grado de pertenencia al conjunto que se le asigna a la variable de salida de esa regla. De este modo, en la regla 1 se asigna la variable de salida al conjunto del tiempo Medio, y vemos que después de ejecutar la inferencia da como resultado un grado de pertenencia a ese conjunto de 8.8786E-11. Como se puede observar la salida pertenecerá al conjunto que marca la regla 8, que es Poco, ya que es la regla que mayor grado de pertenencia genera, para unas entradas de temperatura 20 ºC, humedad del suelo 50 %, estación 6,5 (entre primavera y verano) y una nubosidad del 50%

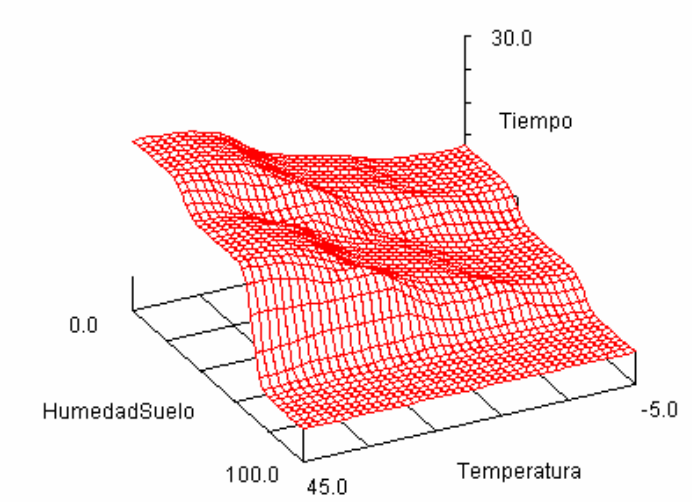


Si cambiamos las variables de entrada aumentando la temperatura a 45 ºC, vemos que ahora la salida pertenece en mayor grado al conjunto asignado en la regla 14, que es el conjunto borroso de tiempo, Medio.

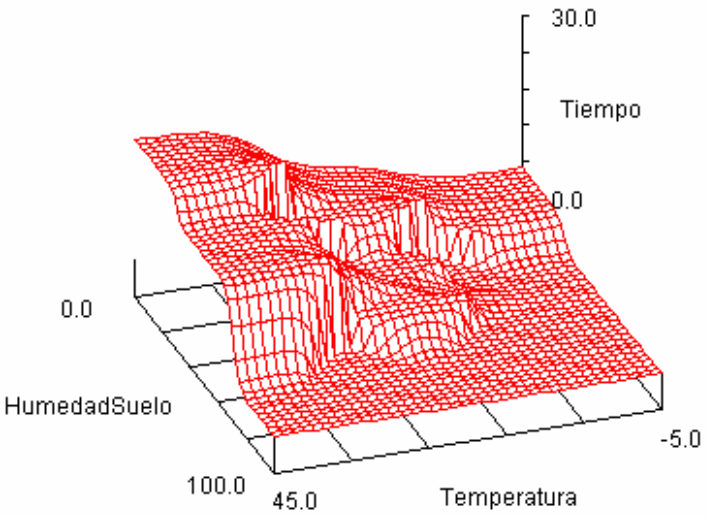
Una vez de definida la base de reglas, realizada la inferencia y habiendo pasado por la defuzzyficación, para ver los resultados estudiamos las gráficas en 3D que nos facilita el propio programa Xfuzzy. Según la familia que usemos para la definición de operadores, obtendremos un resultado u otro.



Zadeh:



Lukasiewicz:



Dichos resultados se obtienen a partir de unos mismos datos de entrada. Se puede observar que son discretamente diferentes. Los resultados proporcionados por Zadeh y el producto tienen en cuenta de forma parcial, todas las variables, sin embargo la lógica definida por Lukasiewicz tiende a 0 ó a 1, perdiendo información.

5.3.2. Implementación de la aplicación mediante Java

Esta parte implementa un **interfaz software** con el controlador borroso que permite visualizar resultados instantáneos, diarios y mensuales; muestra una animación, resultados y gráficas.

Para implementar el sistema se ha utilizado **Borland JBuilder** con el **JDK 1.5**. También se ha creado un proyecto en Eclipse, ya que éste es de libre distribución por lo que el código será más accesible.

Cabe destacar que para conseguir un **ahorro óptimo de agua** en el riego, objetivo del desarrollo del sistema, se ha incluido en ésta parte una matización del tiempo de riego que da la clase que describe el sistema difuso según la cantidad de agua disponible en un depósito del que se supone que regará. Con esto queremos simular lo que ocurre cuando el agua de las reservas pasa un cierto límite y se imponen restricciones en el riego.

Como detalles de implementación, destacar que todas las inferencias son realizadas por las clases Java generadas por la herramienta XFuzzy.
Aparte de la interfaz gráfica, diseñada para visualizar de manera precisa la evolución del entorno, se han tomado algunas consideraciones a la hora de modelar ciertos aspectos. Debido a que es un sistema software, hemos tenido que simular algunos parámetros climatológicos, como es la humedad relativa de la tierra, dado que a falta de sensor hardware, eran necesarios estos datos. Para ello, y de manera empírica, se ha calibrado el ajuste del encharcamiento de la tierra para obtener uno datos bastante aproximados.

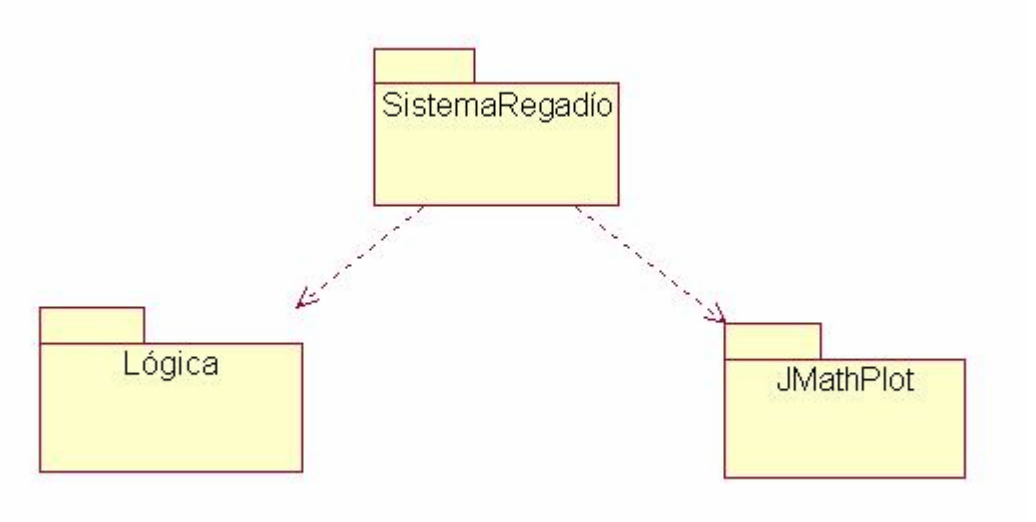
Además, el profesor de hidráulica y riego de la E.T.S.I. de Agrónomos nos comentó que la simulación de la tierra depende de una gran cantidad de parámetros, y que el modelado

de la misma era objetivo de tesis doctorales, por lo que desistimos de realizar un modelo matemático, y optamos por un modelo simple empírico.

Del mismo modo, y para conseguir visualizar cambios climatológicos, en la simulación diaria, se ha optado por realizar una pequeña evolución de tendencias de temperaturas y precipitación, para poder observar la actividad en el sistema sin necesidad de tener que manipular las variables.

5.3.2.1. Diagrama de paquetes de la Aplicación

Nuestra Aplicación se divide en tres paquetes fundamentales, que son **SistemaRegadio**, **lógica** y **JMathPlot**. La interacción de los paquetes se realiza de acuerdo al siguiente diagrama.



El principal paquete de la aplicación es SistemaRegadío, el cual contiene todo lo necesario para la simulación, ya que incluye clases modeladoras de componentes gráficos, relojes para el control de las horas de riego o para llevar la cuenta de los minutos ya regados.

El paquete SistemaRegadío también contiene las principales clases que supervisan a modo de controlador la ejecución de la aplicación, mediante las tareas programadas HiloEventosReloj y la clase TareaMinuto, que gestiona el control una vez se activa el riego.

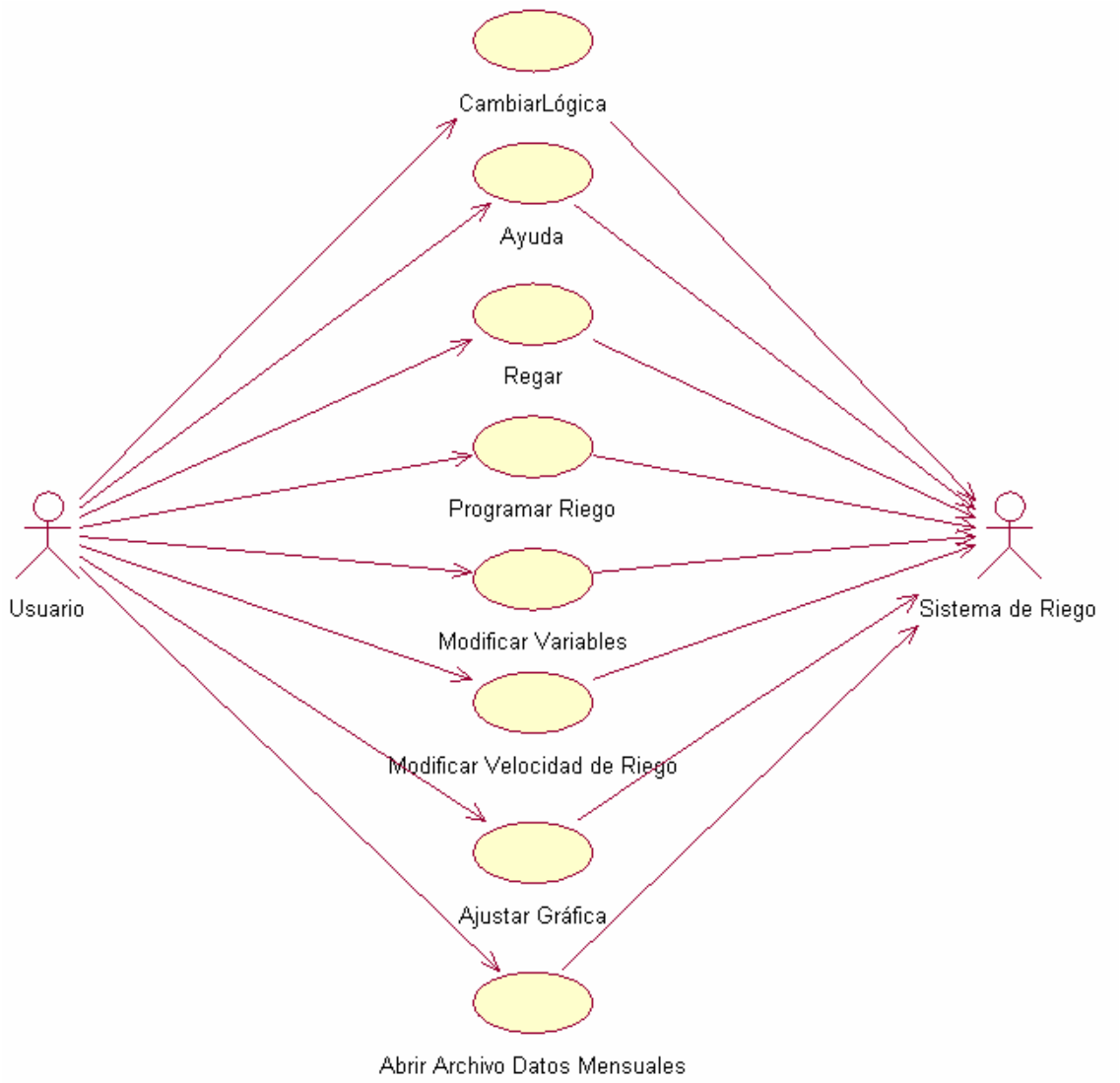
El paquete lógica contiene los interfaces generados por la herramienta XFuzzy, así como las clases que implementan la inferencia del riego mediante las lógicas del Producto, Lukasiewicz o Zadeh.

El otro paquete que utiliza la aplicación es el paquete JMathPlot, de libre distribución en http://sourceforge.net. Mediante este paquete, se permite una visualización más

adecuada, tanto en 2D, como en 3D de las posibles variaciones de la salida del sistema (tiempo de riego) en función de la temperatura y humedad del suelo (factores determinantes) y también nos permite visualizar mensualmente la evolución del tiempo de riego mediante datos obtenidos de fichero.

5.3.2.2. Diagrama de Casos de Uso de la Aplicación

A continuación se muestra el diagrama de Casos de Uso de la aplicación, donde quedan reflejadas el conjunto de interacciones que el usuario puede tener con el sistema, ya sea en la Simulación Instantánea, Diaria o Mensual. Estas interacciones se explican con más detenimiento en el manual del usuario.



i) El paquete lógica

El paquete lógica consta de un conjunto de clases e interfaces generados por el XFUZZY a partir de las especificaciones explicadas en el apartado anterior.

En él quedan determinados los fuzzy-sets, la fuzzyficación, la inferencia a través de reglas y la defuzzyficación.

Clases Generadas por XFuzzy

XFJ: Compilador de XFL a Java

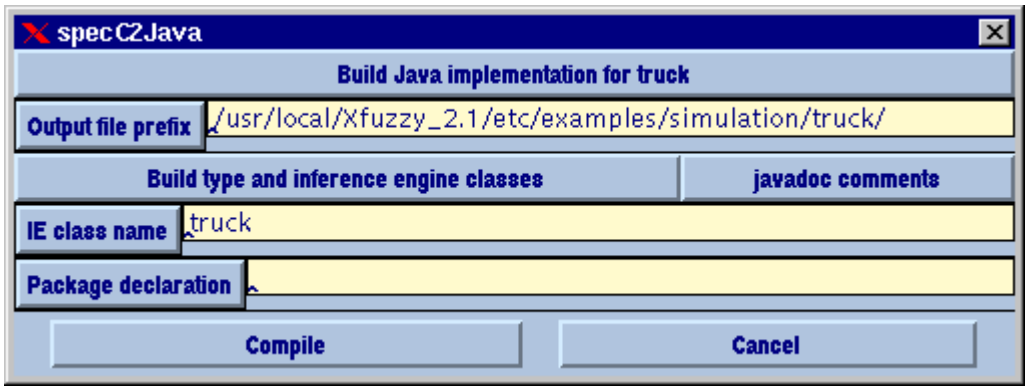
El compilador de XFL a Java, **xfj**, es un programa capaz de generar un conjunto de ficheros fuente Java que implementan los tipos y la base de reglas definidos para un sistema difuso mediante un fichero fuente XFL. Las características del lenguaje destino permiten en este caso que la implementación software del sistema difuso se corresponda más directamente con la estructura de la definición de acuerdo con XFL. El resultado de ejecutar **xfj** son varios ficheros Java, uno para cada una de las clases que constituyen la salida del compilador. Las clases Java que produce **xfj** se corresponden, por un lado, con los tipos definidos en la especificación XFL (cada tipo da lugar a una clase) y, por otro, con la definición del comportamiento definida por la base de reglas: se genera una clase adicional que implementa la base de reglas. [6]

Xfj no permite seleccionar el tipo de aritmética empleada en la representación interna de los valores difusos. Dado que Java garantiza la portabilidad total de sus representaciones numéricas entre diferentes plataformas, toda la aritmética se realiza en doble precisión, utilizando el tipo Java nativo *double*. Sin embargo, el tratamiento de los tipos es según el tipo base XFL de los mismos: en los tipos con tipo base *integer* se efectúan operaciones de redondeo que no se aplican sobre los tipos cuyo tipo base sea real.

Xfj permite emplear las facilidades de autodocumentación que ofrece el lenguaje Java (a través de la herramienta *javadoc*) para ofrecer facilidades adicionales de identificación

de las clases que produce. **Xfj** debe disponer de una especificación de cómo se implementan las operaciones difusas. Si el fichero fuente no contiene directivas que seleccionen operaciones concretas, se emplean un conjunto de operaciones difusas:

- T-norma _sd_min (mínimo).
- T-conorma _sd_max (máximo).
- Negación _sd_not (complemento a 1).
- Función de implicación _sd_min (mínimo).
- Método de defuzzificación _sd_CoA (Centro de Área).



La ventana de síntesis proporciona una interfaz para definir el tipo de dato de las variables difusas (el objetivo de la construcción "typedef FUZZY"), el nombre de la función que implementa el motor de inferencia y el prefijo usado para construir el nombre del fichero de salida, es decir, para asignar un prefijo común a los ficheros Java de salida (*Output file prefix*), para proporcionar el nombre de la clase que implementa la base de reglas del sistema (*IE class name*) y para declarar el paquete al que pertenecen las clases generadas por el compilador (*Package declaration*). Además de los campos de texto, el interfaz ofrece dos botones que permiten controlar las clases Java que se desea generar, y si se incluyen o no comentarios para la herramienta *javadoc* en los ficheros de salida. El interfaz incluye también dos botones de control: uno para llevar a cabo la traducción de la especificación XFL a Java (*Compile*) y otro para cerrar la ventana (*Cancel*).

Explicación de las clases generadas por el xfuzzy

El Xfuzzy genera tres interfaces (**FuzzyInferenceEngine**, **FuzzySingleton**, **MembershipFunction**) y las clases que se indiquen al sistema para implementar la interfaz FuzzyInferenceEngine.

El fichero *FuzzyInferenceEngine.java* describe una interfaz Java que define un sistema de inferencia difuso general. Esta interfaz define cuatro métodos para implementar el proceso de inferencia con valores crisp y difusos.

```
public interface FuzzyInferenceEngine {
 public double[] crispInference(double[] input);
 public double[] crispInference(MembershipFunction[]
input);
 public MembershipFunction[] fuzzyInference(double[]
input);
 public MembershipFunction[]
fuzzyInference(MembershipFunction[] input);

}
```

El fichero *MembershipFunction.java* contiene la descripción de una interfaz usada para describir un número difuso. Contiene sólo un método, llamado *compute*, que calcula el grado de pertenencia para cada valor del universo de discurso del número difuso.

```
public interface MembershipFunction {
 public double compute(double x);

}
```

La clase *FuzzySingleton* implementa la interfaz *MembershipFunction*, que representa un valor crisp como un número difuso.

```
public class FuzzySingleton implements MembershipFunction {
 private double value;

 public FuzzySingleton(double value) { this.value = value;
}
 public double getValue() { return this.value; }
 public double compute(double x) { return (x==value? 1.0:
0.0); }

}
```

Finalmente, el fichero *systemname.java* contiene la clase que describe el sistema difuso. Esta clase es una implementación de la interfaz *FuzzyInferenceEngine*. Por tanto, los métodos públicos que implementan la inferencia son los de la interfaz (*crispInference* y *fuzzyInference*). En nuestro caso hemos utilizado esta estructura para implementar las tres lógicas explicadas anteriormente: producto (SistemaRegadio_Producto), zadeh (SistemaRegadio_Zadeh) y lukasiewickz (SistemaRegadio_luka) de forma que permitamos al usuario comprobar los resultados obtenidos mediante las tres posibilidades para poder ver las diferencias y similitudes.

La **implementación de la clase que describe el sistema difuso** (SistemaRegadio_*logica*.java) contiene varias clases abstractas y concretas: para implementar los comparadores, operadores, conjuntos, reglas e inferencia. A continuación se detallan dichas clases:

Clases abstractas:

- **InnerMembershipFunction**: Pertenencia de una variable de entrada a un conjunto. Con funciones de comparación: mayor o igual, menor o igual, aproximadamente igual, muy igual,...Para utilizarlas al implementar las reglas de xfuzzy.

- **InnerOperatorset**: Conjunto de operadores.

Clases concretas:

- **InnerConclusion**: Representar la conclusión de una regla borrosa. Utiliza la clase anterior de los operadores y la que describe un número difuso (*MembershipFunction*) para implementar el método *compute*.
- **OutputMembershipFunction**: implementa la interfaz *MembershipFunction* con todas sus funcionalidades, entre ellas el método compute.
- **MF_xfl_bell**: Extiende la clase *InnerMembershipFunction* hace una implementación de los comparadores para la lógica que sea (explicados en apartados anteriores.)
- **OP_Operador**: Extiende la clase *InnerOperatorset* implementando los operadores para la lógica concreta.

Clases que representan los tipos de las variables borrosas, indicando los límites, el paso entre distintos valores y los conjuntos borrosos que los representan:

- **TP_Grados**.
- **TP_NivelRadiacion**.
- **TP_GradoHS**.
- **TP_Minutos**.
- **TP_Estacion**.
- **TP_NivelNubosidad**.

Clases que implementan las reglas:

- **RL_ReglasGenerales**: Implementa en Java las reglas hechas con el xfuzzy para la salida general (la que da el tiempo de riego a partir de la percepción solar, la humedad y la temperatura).
- **RL_ReglasPercSolar**: Implementa en Java las clases hechas en el xfuzzy para calcular la percepción solar a partir de la nubosidad y la estación del año. La salida de esta clase será una entrada para la anterior.

Estas clases se utilizan para implementar las funciones que dan la inferencia, que es lo que se utilizará en el sistema final. Estas funciones son las siguientes:

- **CrispInference**: Realiza la inferencia devolviendo un valor nítido. La entrada puede ser un vector de valores nítidos (reales de doble precisión) o borrosos.
- **FuzzyInference:** Realiza la inferencia devolviendo un valor borroso. La entrada puede ser un vector de valores nítidos (reales de doble precisión) o borrosos.

Estas funciones realizan, utilizando las clases explicadas anteriormente, la fuzzyficación (*FuzzySingleton*) y la defuzzyficación (*OutputMembershipFunction*).

Uso de las clases generadas por el XFuzzy

El uso de las clases que genera el xfuzzy es muy sencillo. No es necesario comprender toda la estructura de clases ni el código que las compone ya que sólo se es necesario utilizar las funciones **CrispInference** y **FuzzyInference**, según si la salida será nítida o borrosa.

Para implementar nuestro sistema hemos utilizado la función que toma como entradas valores nítidos y devuelve una salida nítida (CrispInference). También hemos utilizado FuzzyInferenceEngine para declararnos los sistemas fuzzy en nuestras clases. De esta forma permitimos la herencia y podemos cambiar la lógica dinámicamente en el sistema.

ii) El paquete SistemaRegadío

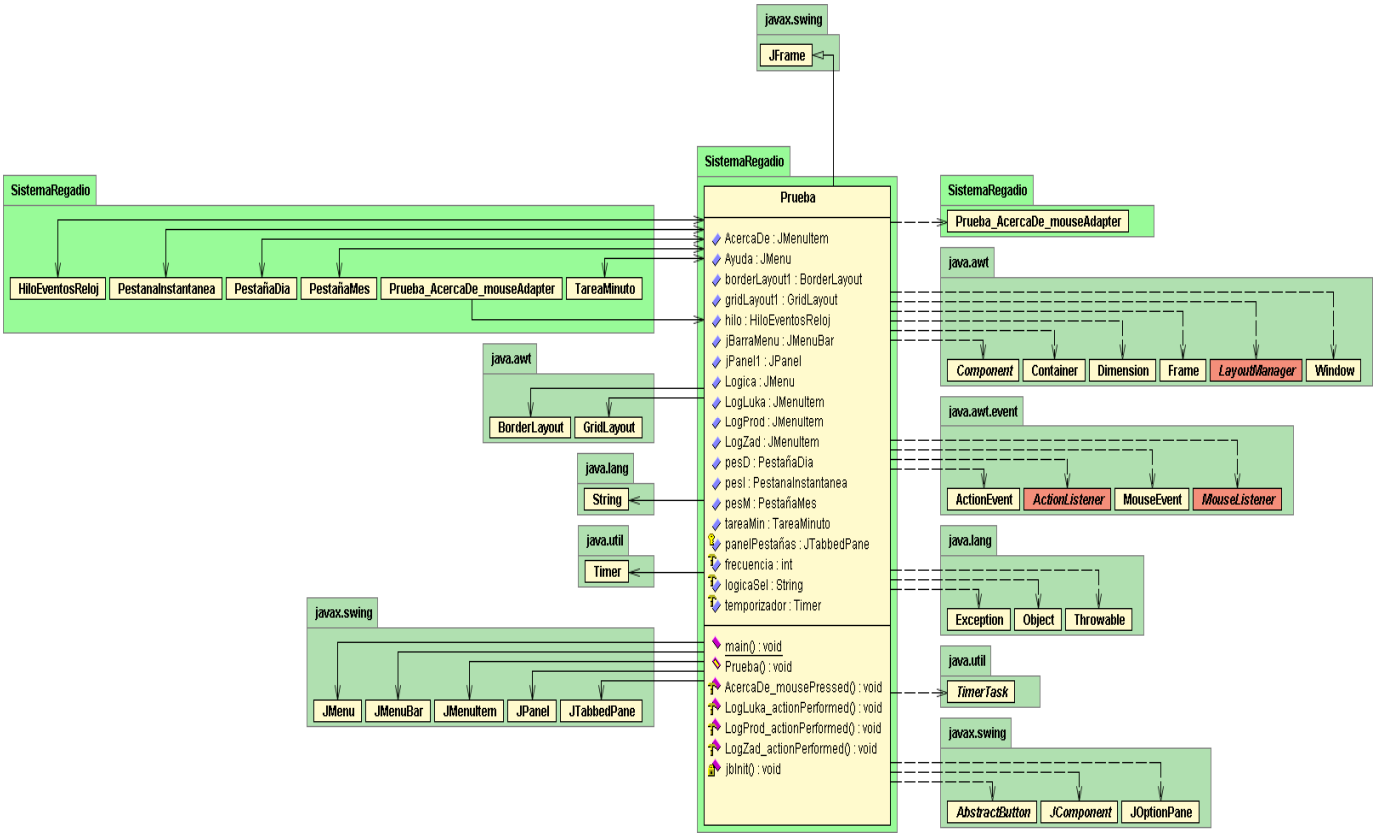
Este paquete está integrado por doce clases distintas que acaban interactuando en una aplicación principal, que extiende de JFrame para poder visualizar la ventana. Las clases se diferencian también en que unas **controlan el aspecto gráfico** (la vista según el patrón MVC), mediante la construcción de paneles externos parametrizados, que evitan la sobrecarga de los paneles principales que muestran la aplicación. Cabe destacar también la existencia de dos tareas programadas, HiloEventosReloj y TareaMinuto, que

a modo de **controlador**, coordinan la ejecución de la aplicación, ya que está orientada a eventos de reloj.

Por orden de importancia se detallan las clases implementadas:

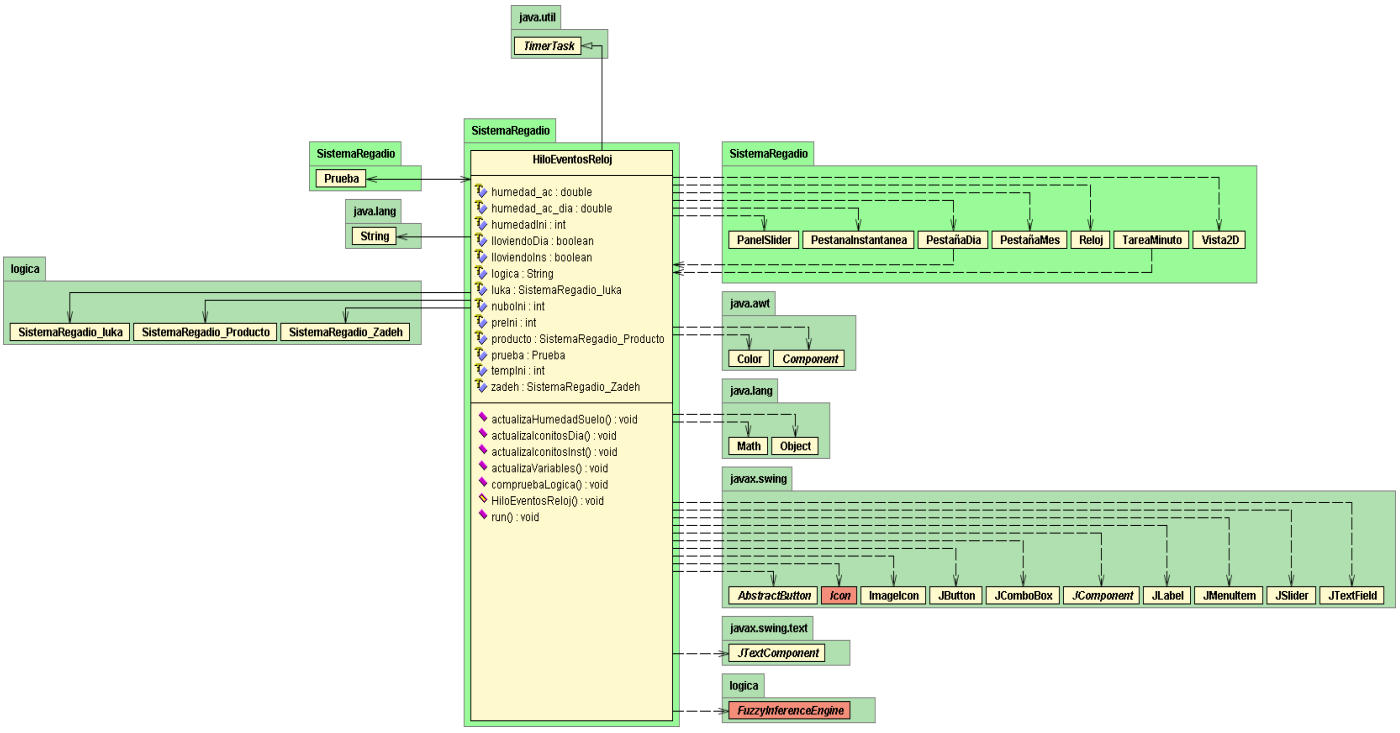
- **PRUEBA**

Es la clase encargada de "crear" la aplicación, posee el método **main()** y es la clase que crea el panel de pestañas, el hilo de eventos de reloj, que se disparará cada décima de segundo. También crea un menú, utilizado sobretodo para poder cambiar el tipo de la lógica borrosa que se esté utilizando. Como detalle, decir que se ha fijado el tamaño del Frame a 1024 x 768 no redimensionable, para evitar problemas de repintado de imágenes de Swing. El diagrama UML de la clase es el siguiente:



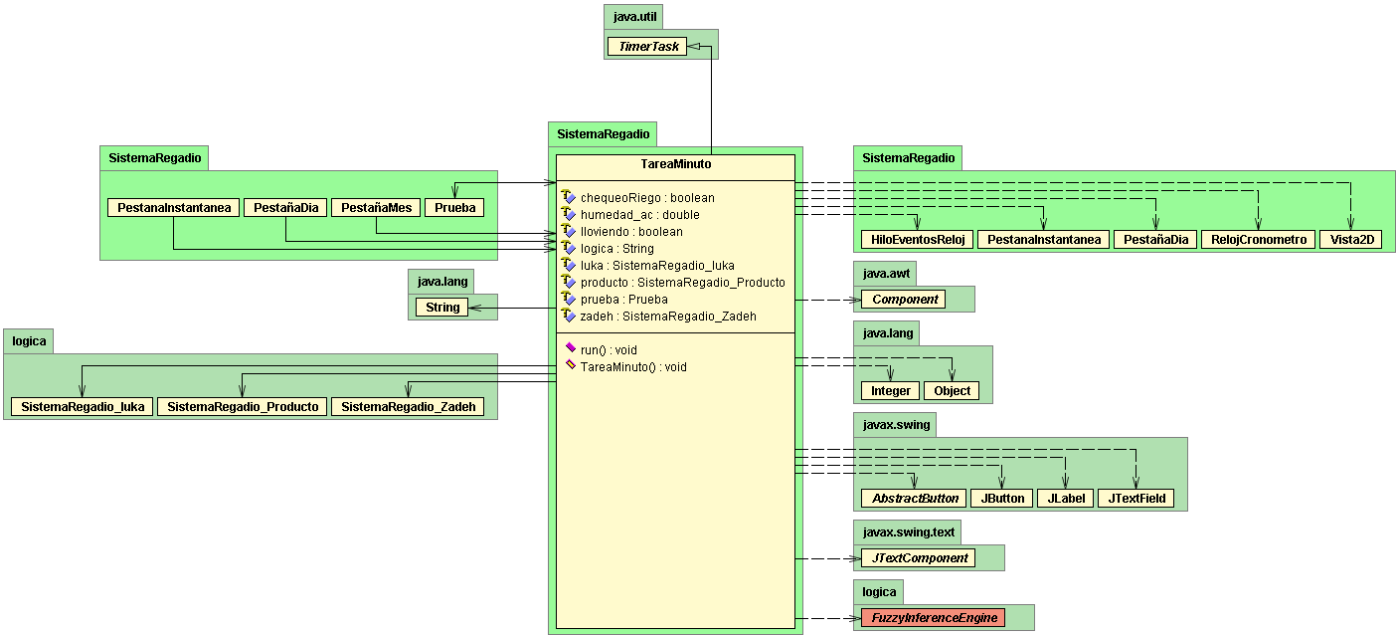
- **hiloEventosReloj**

Esta es la clase que **controla la ejecución** a modo de controlador. Se encarga de refrescar los valores tomados de las variables de entrada, ajustando los iconos para el aspecto visual. Se encarga de controlar cuando se comienza un riego, ya sea en modo diario o instantáneo. Controla el aumento de humedad del suelo, debido a la lluvia o al propio riego (para ellos se ha utilizado un modelo muy sencillo de aumento de la humedad en el tiempo, puesto que hacer un modelo más completo habría sido objeto de otro proyecto). Genera condiciones climáticas según la tendencia señalada (pestaña diaria). También controla el cambio de imágenes de la Web Cam, según las condiciones climáticas que se den en el momento. El diagrama UML de la clase es el siguiente:



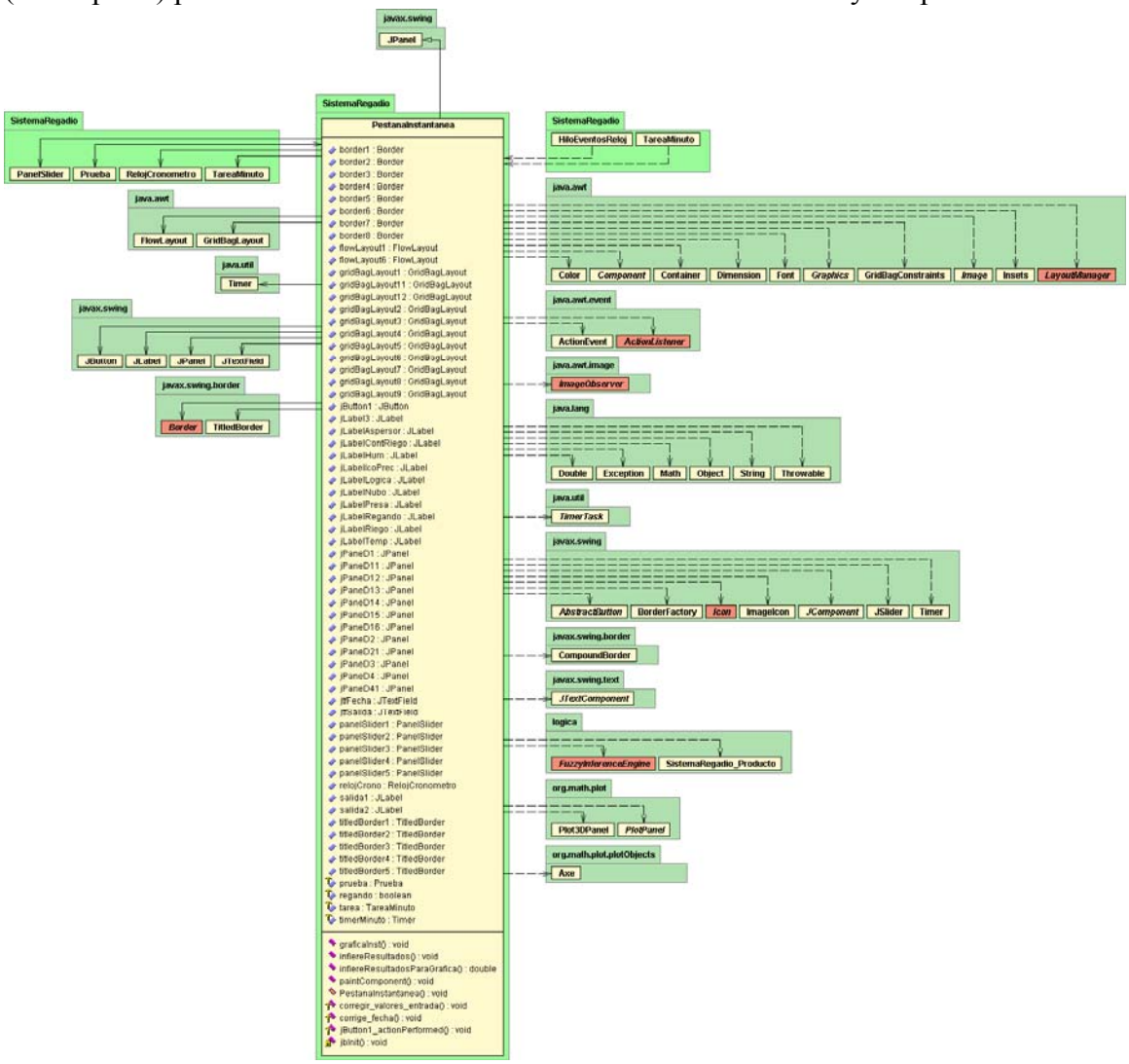
- **tareaMinuto**

Esta clase es otra tarea programada que se ejecuta cada segundo una vez haya comenzado un riego. De esta forma, se evita sobrecargar al sistema por el otro hilo, y se permite el poder variar los tiempos deseados para la obtención de inferencias una vez que se está regando, para poder precisar el tiempo de riego ante unas condiciones climáticas cambiantes o inestables. El diagrama UML de la clase es el siguiente:



- **pestañaInstantánea**

Esta clase está implementada en una pestaña del panel de pestañas. Con esta clase se hace un **seguimiento** y una **simulación** de una **inferencia del sistema**, según la lógica seleccionada. En cada instante, podemos conocer cual es el tiempo de riego estimado por el sistema, para unos valores de variables de entrada. También esta clase permite la simulación de un riego on-line, visualizando como evolucionaría el tiempo del mismo en caso de que se vean afectadas condiciones medioambientales. En caso de detectar el sistema que el tiempo transcurrido supera al inferido en cada segundo, detiene el riego. En caso contrario continúa regando. Si el sistema infiere que hay que regar cero minutos, lo detecta y no riega. Para mejorar la comprensión, se incluye una gráfica 3D (JMathplot*) para evaluar valores de salida en función de humedad y temperatura.



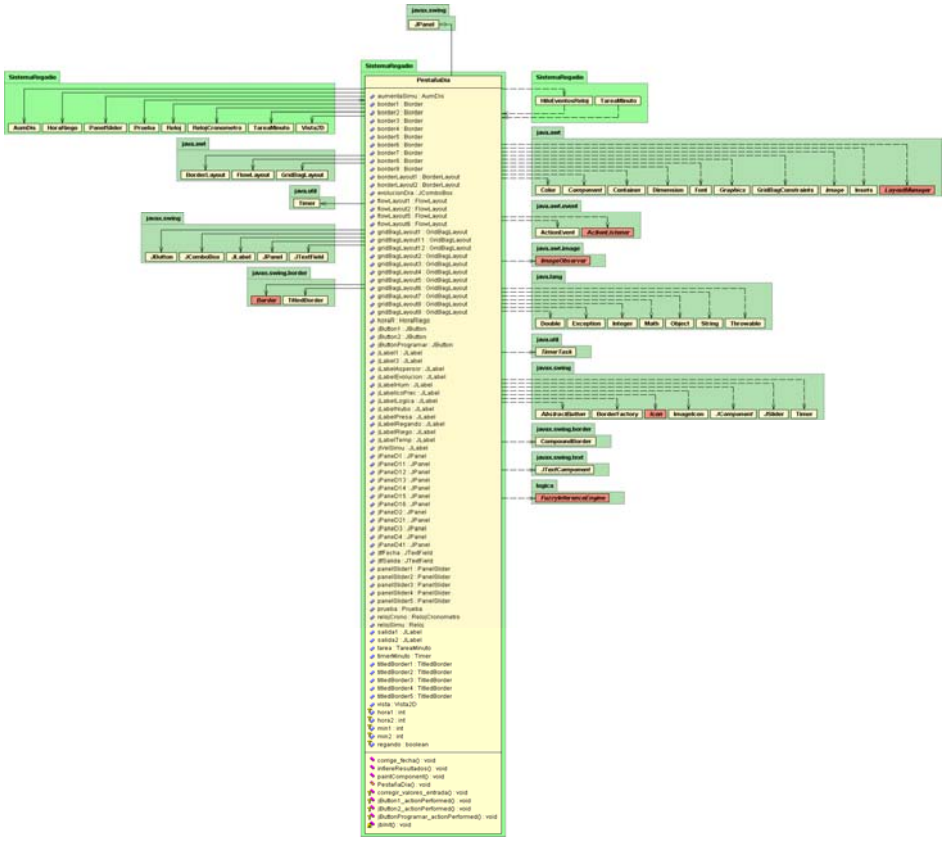
- **pestañaDia**

Esta clase realiza un **seguimiento** de una **simulación** de riego a lo largo de **un día**. Para ello consta de un reloj de simulación, que avanza en tramos de minuto para poder observar de manera rápida la evolución a lo largo de 24 horas.

Tiene toda la funcionalidad de la instantánea, a diferencia de que incluye un panel de programación de riego, donde el usuario puede introducir las dos horas del día para regar, por lo que el sistema, cuando detecte que es la hora, inferirá el tiempo óptimo de riego y solamente regará cuanto sea necesario.

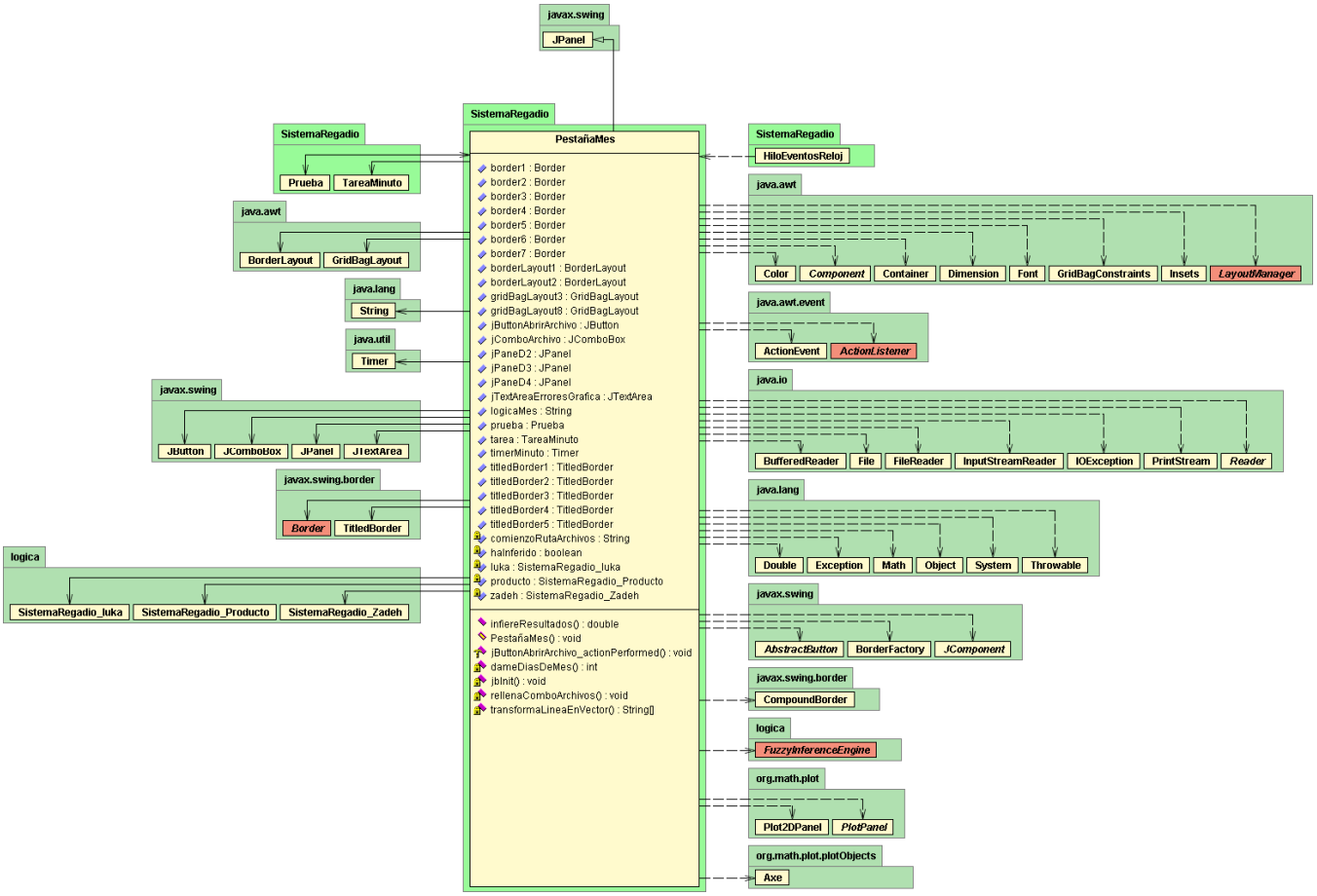
Dado que estamos simulando condiciones climatológicas, por falta de sensores, hemos incluido unas posibles evoluciones del día, para evitar que el usuario esté actualizando continuamente el valor de las variables de entrada.

Se muestra una WebCam, en la cual se queda reflejada la situación climatológica actual.



- **pestañaMes**

En esta clase se permite la **lectura desde fichero de datos meteorológicos** obtenidos de la estación meteorológica de la facultad de Ciencias Físicas y realizar una visualización de la evolución del tiempo de riego a lo largo de un mes. También se muestran resultados comparativos frente a sistemas de riegos convencionales. El diagrama UML es el siguiente:



- **Reloj**

La clase reloj nos proporciona un reloj disparado por un Timer() configurable, que permite su lanzamiento parametrizado. Ha sufrido alguna modificación respecto a un reloj convencional, ya que hemos prescindido del segundero, ya que pretendemos agilizar el tiempo de simulación sin el mismo.

- **RelojCronómetro**

Es una extensión de la clase reloj, solo que como se usa sólo para contar minutos, pues se ha prescindido del segundero y del dígito de hora. También para darle un mejor aspecto, se han cambiado los dígitos de color para distinguirlo del reloj normal.

- **Vista2D**

Es la clase que se encarga de visualizar imágenes no incluidas en un JLabel. Hace uso de un atributo Graphics que se encarga de visualizar una imagen en formato *.jpeg.

- **panelSlider**

Es una clase creada a modo de patrón, para evitar la creación de un exceso de atributos en las pestañas. Contiene un JSlider, una etiqueta y una variable que indica el valor del jSlider.

- **HoraRiego**

Es una clase que crea un panel que tomará los datos de la hora y minuto de riego de los dos posibles riegos disponibles en un día. Al igual que la clase anterior, ha sido creada para evitar la saturación de atributos y replicación al usarlos en varias pestañas

- **AumDis**

Es una clase que crea un panel con un jTextField, dos botones y un JLabel que sirven para permitir introducir algún dato al usuario, y poder modificarlo incrementalmente mediante el + y -. También dispone de una variable interna que va controlando el valor del jTextField.

- **jMathPlot**

JMathPlot es una librería de libre distribución (http://jmathtools.sourceforge.net/jmathplot-tutorials.php) que permite realizar, de forma sencilla y rápida, gráficas en dos y tres dimensiones en Java.

Para introducir los datos a representar en la gráfica se utilizan vectores de *double*, de diferente dimensión según sea necesario
Se muestra la gráfica en un panel de forma que se puede insertar fácilmente en cualquier interfaz Java y se pueden utilizar los atributos y métodos propios de los paneles, así como otros adicionales para cambiar el color, poner etiquetas, etc. Dicho panel no sólo muestra la gráfica sino que permite cambiar la escala, ver los datos a partir de los que se dibuja la gráfica, girarla (en el caso de la de 3 dimensiones) y guardarla en un archivo.

El paquete se distribuye en forma de .jar para poder incluirlo en cualquier proyecto. También está disponible el .zip con el código fuente.

5.4. Pruebas

Las pruebas del sistema nos han servido para poder ajustar determinados aspectos visuales, facilitando el manejo al usuario.

Uno de las decisiones que tuvimos que tomar es la de bloquear el tipo de lógica seleccionada, al menos mientras se está regando.

Para evitar introducir valores erróneos y tener que realizar un control exhaustivo de errores, optamos por introducir barras de selección (JSlider) en vez de cuadros de texto. De esta forma, establecemos unos rangos de datos y éstos solo varían en dicho rango.

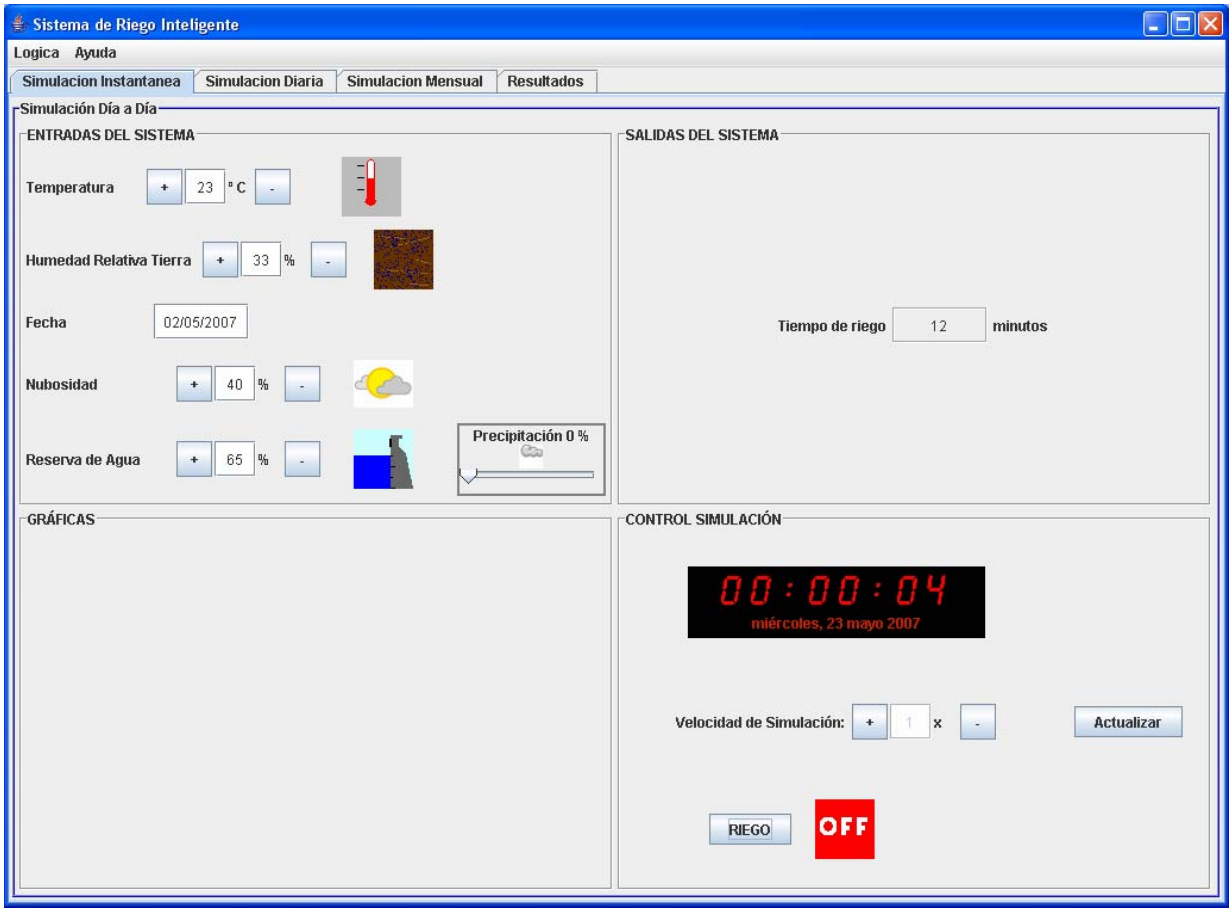
Para mejorar la visualización, decidimos introducir iconos gráficos activos (cambian con el estado de las variables de entrada), por lo que de esta forma se mejoró el aspecto visual.

Se introdujo un contador de minutos de riego, que llevase la cuenta del tiempo de ejecución del riego. De esta forma, al igualar o superar el tiempo inferido, el sistema detiene el riego.

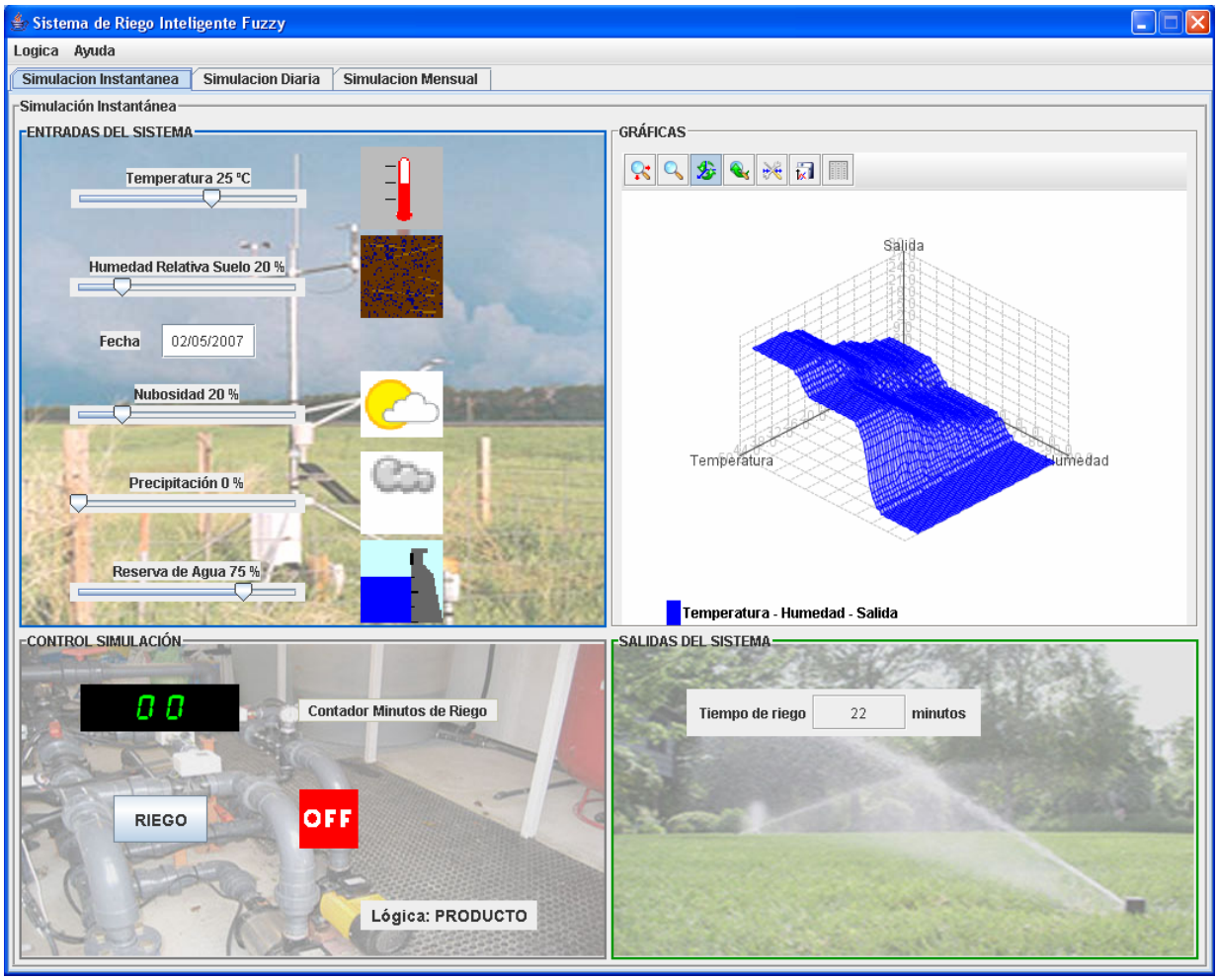
5.4.1. Ejecución de las pruebas

Las pruebas se han realizado de manera incremental, prestando especial atención al realismo, resultados e interfaz gráfica.

Pasamos de una apariencia como esta:



A esta otra:



Las pruebas incrementales no han ayudado a ajustar los conjuntos borrosos y el sistema de reglas de inferencia.

5.4.2. Resultados

Para comparar nuestro sistema con los sistemas tradicionales suponemos que en ambos se usan los siguientes aspersores:

Aspersor para parque 80
Círculo completo o parcial
Rosca macho de ½"
Fabricado en latón con 1 boquilla
Tornillo difusor y deflector
Caudal operativo: 450 a 720 l/h
Presión operativa: 1,5 a 4 kg/cm²
Diámetro regado: 20 a 24 m.

Las ganancias de riego se podrían observar en minutos de riego, pero como lo más importante es contemplar el ahorro de agua se mostrarán los cálculos en esos términos (12 litros el minuto).

Según los datos obtenidos de las entrevistas con un experto jardinero del Ayuntamiento de Madrid tenemos los siguientes datos para el tiempo de riego por estaciones y por día:

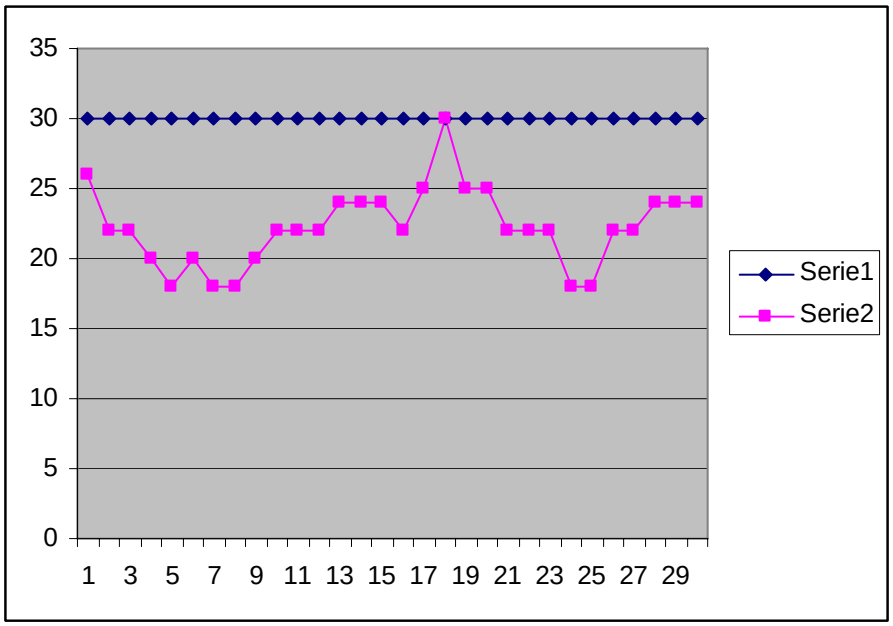
Primavera: ~20 minutos.
Verano: ~30 minutos.
Otoño: ~ 10 minutos.
Invierno: depende de las lluvias.

En nuestro sistema después de varias ejecuciones utilizando datos reales sacados del instituto de meteorología y de la facultad de agrónomos y sacando la media hemos obtenido los siguientes valores:

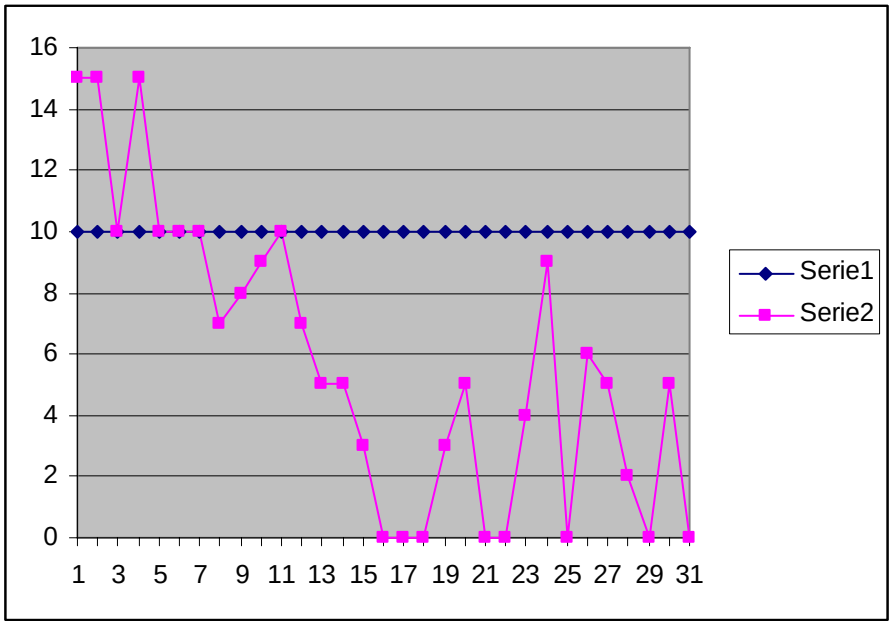
Primavera: 5 – 15 minutos -> ahorro 10 minutos aprox.
Verano: 20 – 25 minutos. -> ahorro 7 minutos aprox.
Otoño: 5 – 10 minutos -> no hay ahorro.
Invierno: 0 – 5 minutos -> no hay ahorro.

A continuación mostramos gráficamente los datos con los resultados. La Serie1 se corresponde con el riego tradicional y la Serie 2 se corresponde con el resultado de nuestra aplicación.

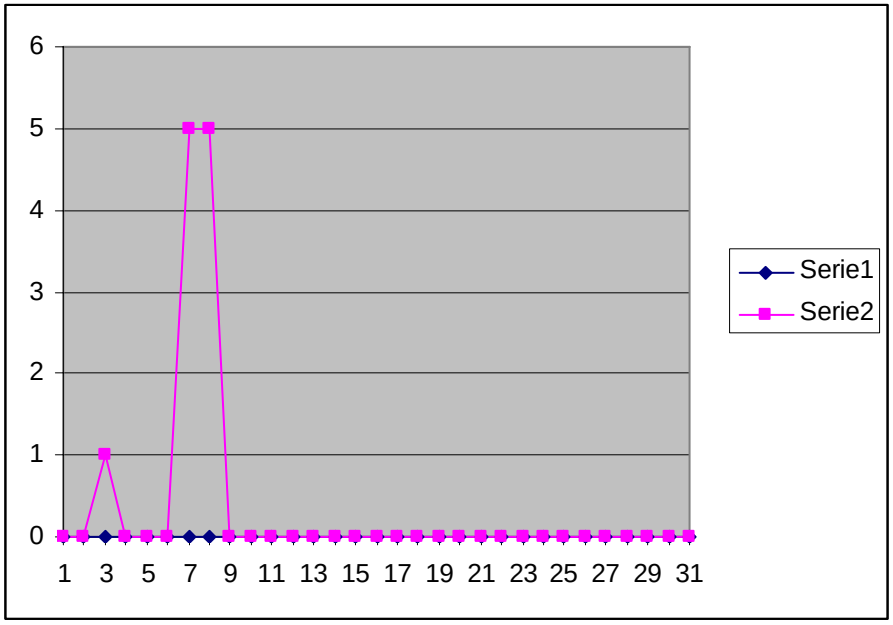
Verano:



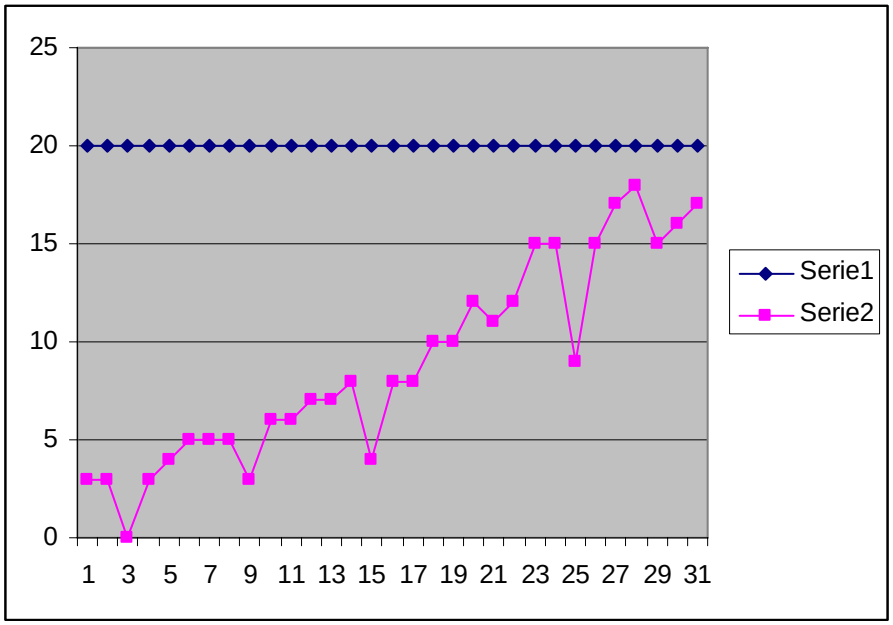
Otoño:



Invierno:



Primavera:



En estas gráficas se puede ver perfectamente cómo la respuesta del sistema está adaptada a las condiciones de forma completamente dinámica. Por ejemplo, los días que riega cero minutos suele ser porque había mucha humedad muy probablemente debida a la lluvia. Es un riego mucho más exacto puesto que no se basa en algo fijo sino que responde a los cambios. Si un día de otoño hace temperatura de verano, por ejemplo, regará más que lo estimado para el otoño, puesto que la planta tendrá unas necesidades parecidas a las de verano, aunque no regará tanto como en verano porque al ser otoño se supone una tendencia de menos necesidad de agua que en verano. Sin embargo, con un riego tradicional, el sistema regaría 10 minutos aunque hiciera mucho calor, lo que podría perjudicar a las plantas.

En este caso se observa un ahorro de agua que parece pequeño pero que es considerable a lo largo del año.

Para los datos probados sólo se observa ahorro en los meses de verano y primavera, así que sólo se contemplan datos para ellos (para las otras estaciones las diferencias no son significativas).

Los meses de **verano** son 3, con unos 31 días cada uno y regando una vez al día (en ambos casos: riego tradicional y nuestro sistema), serían 96 riegos. Si en cada riego se ahorran 7 minutos, es decir, 84 litros, en todo el verano se ahorran 8064 litros por cada aspersor de este tipo.

Los meses de **primavera** son otros 3. Para redondear suponemos meses de 31 días. Como en el caso anterior se supone que se riega una vez al día serían igualmente unos 96 riegos. Como en cada riego se ahorran 10 minutos son 120 litros. En los 96 riegos el ahorro total es de 11520 litros.

Con lo cual, para los datos comprobados se consigue un ahorro anual de 19584 litros **por cada aspersor**, es decir, unos 20000 litros al año. Teniendo en cuenta que un jardín doméstico tiene del orden de 5 a 10 aspersores, estaríamos hablando de un ahorro de 200000 litros de agua (200 m^3), el equivalente a una piscina de 16 metros de largo x 6 de ancho y 2 metros de profundidad media.

Además, la principal ventaja de nuestro sistema y la que nos produce el ahorro, es que responde frente a los cambios instantáneos meteorológicos, como son las precipitaciones. Además, si una temporada de otoño (por ejemplo 15 días) hace mucho calor, las plantas evaporarán más agua y el riego de 10 minutos no sería suficiente. Nuestro sistema regaría un poco más manteniendo a la planta en óptimas condiciones. Por otra parte, si por ejemplo en primavera hace varios días de frío o nublados, nuestro sistema regará menos de forma que se ahorrará agua sin perjudicar a la planta.

En este apartado se compara nuestro sistema con el riego tradicional. Lo más adecuado sería compararlo con otros sistemas de riego inteligentes existentes en la actualidad, pero lamentablemente no hemos podido obtener suficientes datos de otros sistemas como para poder hacer buenas comparaciones.

Lo más que hemos podido conseguir es hablar con algunos jardineros de El Retiro de Madrid que nos han mostrado su descontento con un sistema de riego inteligente que se está utilizando en la actualidad.

En cualquier caso, no hemos encontrado datos de ningún sistema de riego inteligente que utilice control borroso, con lo que el nuestro, en ese sentido, introduce una gran innovación. Las ventajas explicadas en apartados anteriores sobre control inteligente borroso se pueden entender como ventajas de nuestra aplicación sobre otros sistemas inteligentes.

6. CONCLUSIONES

Consideramos que hemos hecho una gran aportación al riego inteligente ya que no hemos encontrado otro sistema de riego inteligente que utilice control borroso.

El sistema de control funciona perfectamente de acuerdo a las especificaciones del sistema. De poder ser implementado en la realidad, con sensores, podría conseguir un ahorro considerable de agua, como se puede ver en los resultados de la pruebas, con lo que hemos logrado el objetivo primordial del desarrollo.

Al ser un sistema complejo que tiene muchas consideraciones hemos aprendido mucho con su desarrollo.

Además hemos aumentado nuestro conocimiento sobre *ingeniería de sistemas basados en conocimiento* realizando las fases del estudio del conocimiento.

También hemos mejorado nuestros conocimientos de programación en Java, manejo de nuevas librerías, interfaces gráficas, tareas programadas, etc.

Y destacar como más importante, los conocimientos adquiridos en ***lógica borrosa***: hemos aprendido los conceptos teóricos, su utilidad, sus ventajas, su aplicación práctica,… Además, hemos tenido la oportunidad de utilizar la herramienta XFuzzy, la cuál nos ha facilitado mucho el desarrollo.

7. FUTURAS AMPLIACIONES

Desde nuestro punto de vista se podrían hacer tres ampliaciones para mejorar el sistema.

La **primera ampliación**, y la más sencilla, sería incluir una forma de parametrizar el sistema con los valores fijos del cultivo, el tipo de suelo, número de aspersores, etc. Como hemos explicado en la primera parte de esta memoria, es determinante en el tiempo de riego, pero es algo fijo, no varía con el tiempo. Hemos realizado el sistema para un suelo medio, pero se podrían incluir modificaciones para que pueda **configurarse** el sistema ajustándose a ciertas características.

La **segunda ampliación** que se nos ocurre sería incluir **aprendizaje**. La herramienta utilizada para las reglas, el XFUZZY, permite aprendizaje, así que probablemente se podría llevar a cabo modularmente en ésa parte del sistema manteniendo la parte Java tal y como está. El incluir aprendizaje supondrá, muy probablemente, hacer una estudio del conocimiento como el que hemos hecho para este sistema para incluir o modificar las variables que tenemos adecuándolas a los nuevos requerimientos.

La **tercera ampliación** posible es realizar una implementación **hardware**. Para ellos habría que adquirir sensores (de temperatura, luminosidad y humedad), conectores, etc., en definitiva, todo el hardware necesario utilizar el sistema en un cultivo real. Habría que estudiar la forma de conectarlo todo y ajustarlo a las distintas condiciones. Consideramos que ésta ampliación se podría llevar a cabo en otro proyecto de fin de carrera.

8. BIBLIOGRAFÍA

[1] Dubois D, Prade H (1980) Fuzzy Sets and Systems. Theory and its Applications. Academic Press, New York

[2] Pradera A; Trillas E, Cubillo S (2000) On modus ponens generating functions. Internat. J. Uncertain. Fuzziness Knowledge Based Systems 8, 1, pp. 7-19.

[3] B. Schweizer, A. Sklar. Probabilistic metric spaces. North-Holland, Amsterdam, NL, 1983.

[4] E. Trillas, C. Alsina and J. M. Terricabras. *Introducción a la Lógica Borrosa*. Editorial Ariel. 1995.

[5] Trillas E. and L. Valverde (1985). On mode and implication in approximate reasoning. *Approximate reasoning in expert systems.* Eds. M. M. Gupta. North-Holland. pp. 157-166.

[6] Xfuzzy. http://www.imse.cnm.es

[7] L.A. Zadeh. Fuzzy sets. Inform. and Control 8, 338–353, 1965.

[8] L. A. Zadeh. Similarity relations and fuzzy orderings, Inform. Sci. 3, 177–200, 1971.

[9] G. Pajares, M. Santos; Inteligencia Artificial e Ingeniería del Conocimiento; RA-MA, 2005

[10] Buschmann, Frank et al.: Pattern Oriented Software Architecture, Volume 1: A System of Patterns, Willey & Sons, 1996.

[11] "Estimating water requirements of landscape plantings. The landscape coefficiente method." Laurence R. Costello, Nelda P. Mayheny y James R. Clark. Cooperative Extension University of California, Division of Agricultura and Natural Resources. 1991

9. APÉNDICE

9.1. Manual de usuario

SISTEMA DE RIEGO INTELIGENTE BORROSO





MANUAL DEL USUARIO E INSTALACIÓN

Instalación

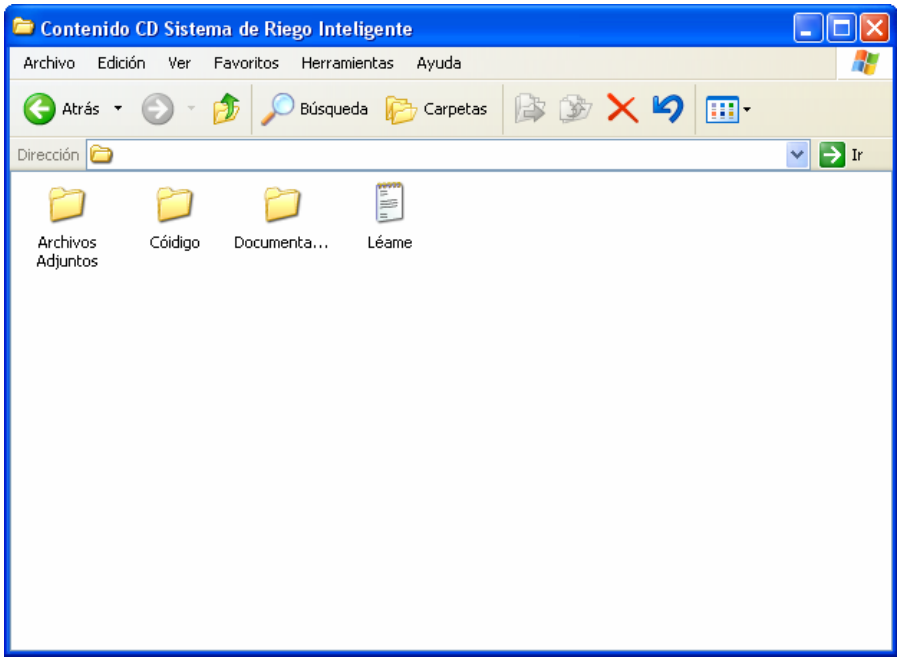
El CD adjunto contiene el Proyecto "Sistema de Riego Inteligente", realizado en el curso académico 2006-2007. El contenido del CD está dividido en: Código, Documentación y Archivos Adjuntos.

En la carpeta Documentación podrá encontrar la memoria del proyecto, con todo su proceso de introducción, investigación y desarrollo.

El código Java del proyecto se encuentra en la carpeta "Código". Por mayor comodidad, se encuentra en dos proyectos para ejecutar en dos entornos de programación distintos, en Borland JBuilder y en Eclipse.

En la carpeta "Archivos Adjuntos" podrá encontrar la JDK necesaria para la ejecución de la aplicación, así como otro tipo de software necesario para la ejecución de la misma.

Es importante que sepa, que este proyecto está dirigido a un determinado tipo de profesionales del sector agrónomo, y dada la complejidad matemática, se recomienda leer antes toda la documentación adjunta.



Contenido del CD

Instalación del Software

Para instalar el software de Simulación, es recomendable copiar a un disco duro uno de los proyectos incluidos en la carpeta código. El usuario debe comprobar que dispone del entorno de programación instalado, así como de la Máquina Virtual de Java, en su Kit de Desarrollo (JDK) con versión 1.5 o superior.

En caso de no tenerla instalada, puede encontrarla en la carpeta "Archivos Adjuntos", o bien descargársela de la página http://java.sun.com.

Manual del Usuario

Inicio de la Aplicación

Para ejecutar la aplicación es necesario abrir el Eclipse, indicando la ruta del "Workspace" deseada, y una vez indicada, ejecutar la clase "Prueba.java".

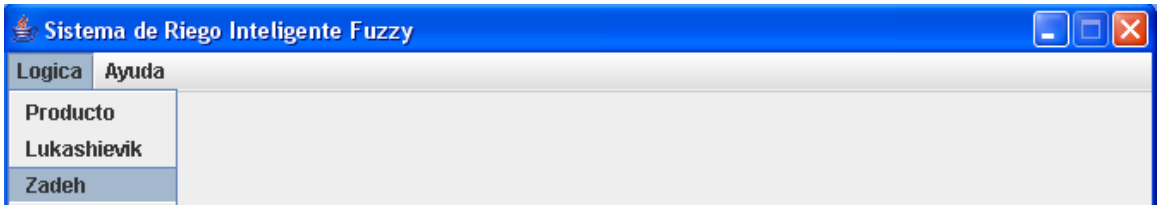
En caso de usar Borland JBuilder, basta con abrir el proyecto y ejecutar la clase "Prueba.java".

Nota: La aplicación está optimizada para una resolución de 1024 x 768 píxeles por pulgada, por lo que es recomendable ajustar la resolución de su monitor a una resolución similar o superior.

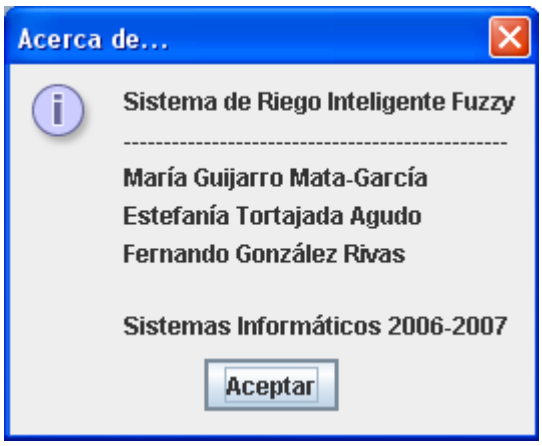
A continuación se describen los modos de funcionamiento de la aplicación: Simulación Instantánea, Diaria o Mensual.

Común a los tres tipos de simulación, es el menú situado en la parte superior, donde se permite modificar el tipo de lógica borrosa deseada para las inferencias de los tiempos de riego.

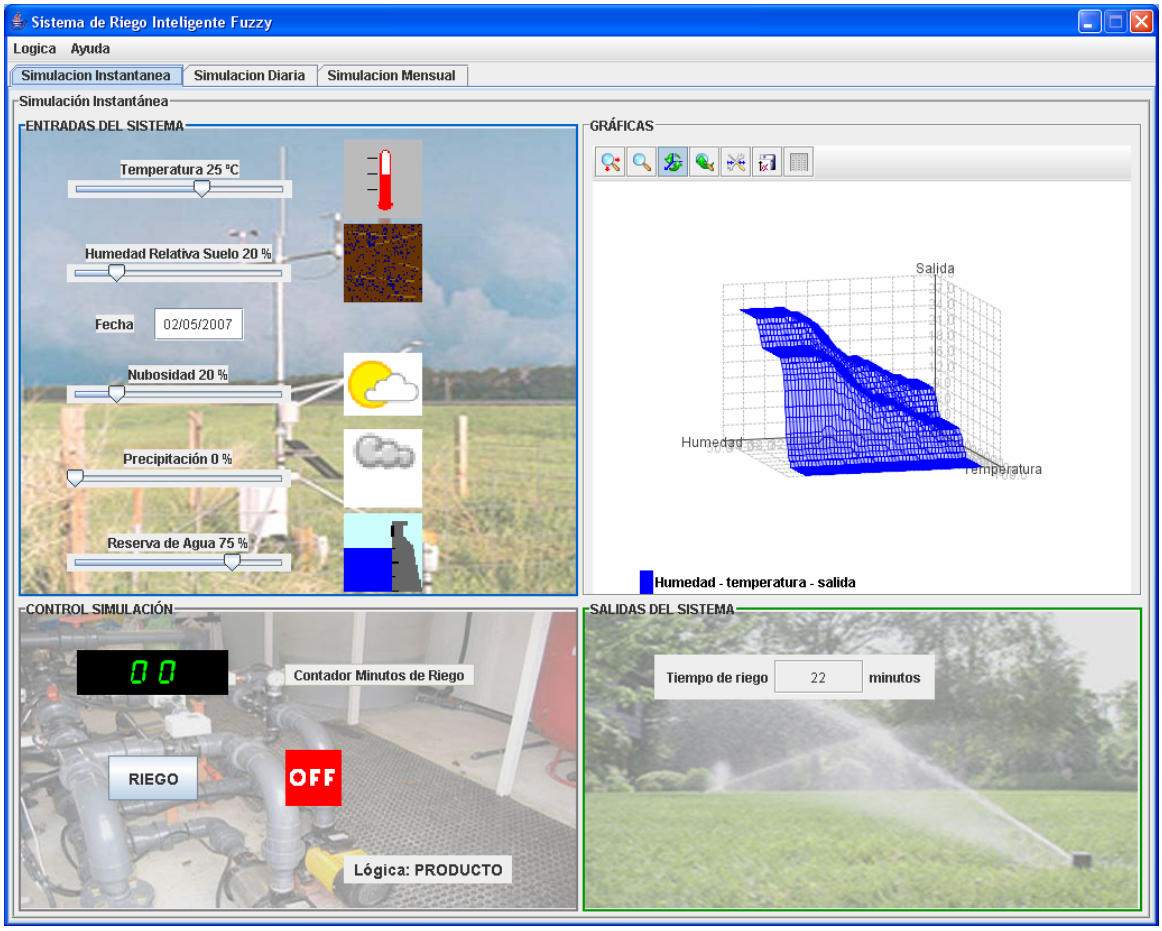
Como aclaración, destacar que no se pueden modificar dichos tipos de lógica, una vez que hay un riego en ejecución, ya sea en una pestaña o en otra.



También en el menú se encuentra disponible en Ayuda, un "Acerca de" que informa al usuario de los autores del proyecto:

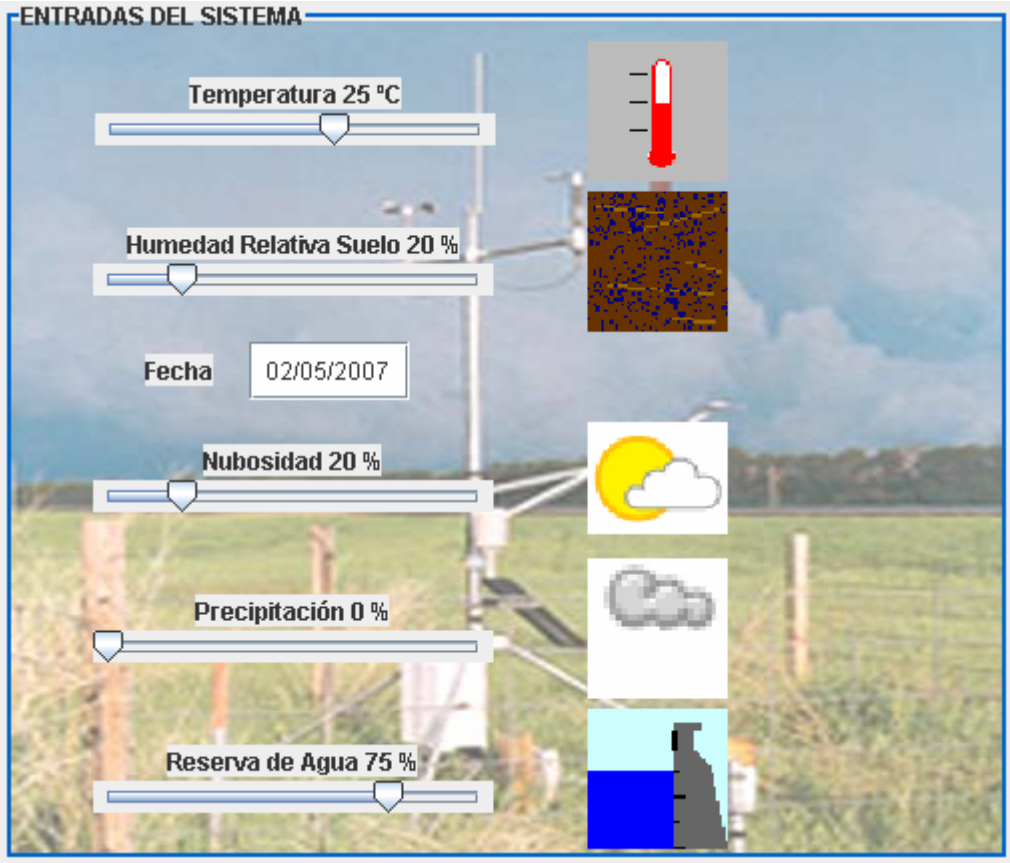


Simulación Instantánea



La simulación instantánea recrea las condiciones atmosféricas, en teoría recogidas por sensores hardware situados en el exterior, que en la práctica son variables configuradas por el usuario.

Como se puede observar, está dividida en cuatro paneles: Entradas del Sistema, Control de Simulación, Gráficas y Salidas del Sistema.



Las entradas del sistema permiten al usuario configurar, mediante JSliders, la temperatura ambiente deseada, humedad relativa del suelo inicial, una fecha determinada, un porcentaje de nubosidad, de precipitación, y un tanto por ciento de reserva acuífera disponible para el riego, para que, en función de todas estas condiciones, el sistema pueda inferir el tiempo de riego óptimo para la planta.

Comentar que una vez que un riego ha comenzado o el nivel de precipitación es superior a cero, la humedad relativa del suelo quedará bloqueada, ya que se está actualizando su valor en función de estimaciones de agua procedentes de la lluvia, del riego, o de ambas.

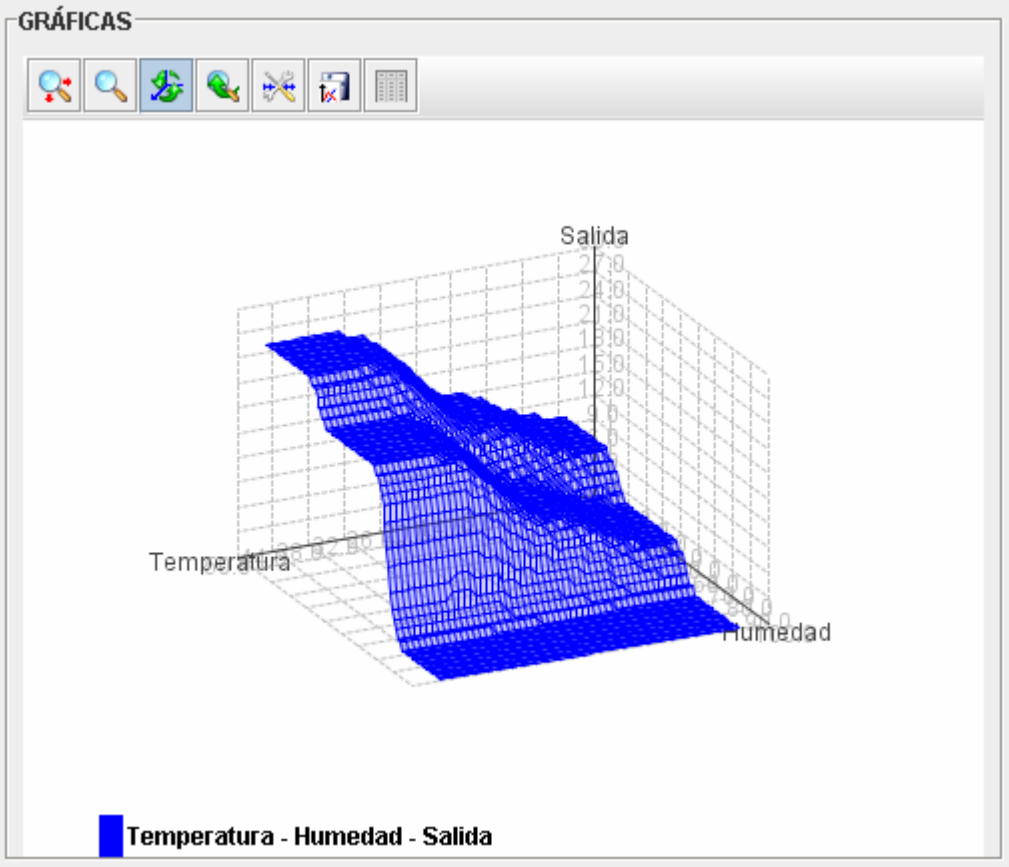
El Control de Simulación, permite al usuario activar el riego de manera instantánea cuando el considere oportuno. Una vez activado el riego, se visualizará mediante el Contador de Minutos, el número de minutos regados en función de la inferencia.

El número de minutos estimados se encuentra en el panel de Salidas del Sistema, por lo que una vez alcanzados o superados ese número de minutos, el sistema detendrá el riego instantáneamente.

A modo informativo, en este panel se indica el tipo de lógica seleccionada por el menú.

En el panel "Salidas del Sistema" se muestra al usuario, como ya se ha comentado, el tiempo de riego inferido por el Sistema de Riego *Fuzzy*, dada las condiciones ambientales prefijadas. En caso de encontrarse regando, se visualizará un aspersor y un cuadro informativo en parpadeo continuo.

En el panel de Gráficas, se encuentra una visualización 3D del modelo de inferencia seleccionado en el menú "Lógica", con los correspondientes valores de la salida en función de la temperatura y de la humedad relativa, factores determinantes en el tiempo estimado de riego



Este panel, es un Frame JMathPlot, con lo que se permite realizar distintos movimientos en la gráfica, así como moverse, hacer zoom, guardar datos, o visualizar valores de las variables.

Simulación Mensual

La simulación mensual, permite exprimir la funcionalidad de la simulación instantánea ampliándola a un uso a lo largo de un día completo, permitiendo la programación de hasta dos horarios de riego, simulación de una posible evolución climatológica, y visualización externa climática mediante una Web Cam.

Como novedades, respecto a la Simulación Instantánea, cabe destacar en el panel de Entradas del sistema, la posibilidad de seleccionar una posible evolución o tendencia climatológica del día, por lo que el sistema simulará el aumento/disminución de ciertas variables.

Las distintas opciones de tendencia en la evolución del día supondrán un aumento o disminución de ciertas variables. Por ejemplo, una evolución a calurosa, influirá en un aumento de la temperatura y una disminución de la humedad relativa debido a la evaporación.

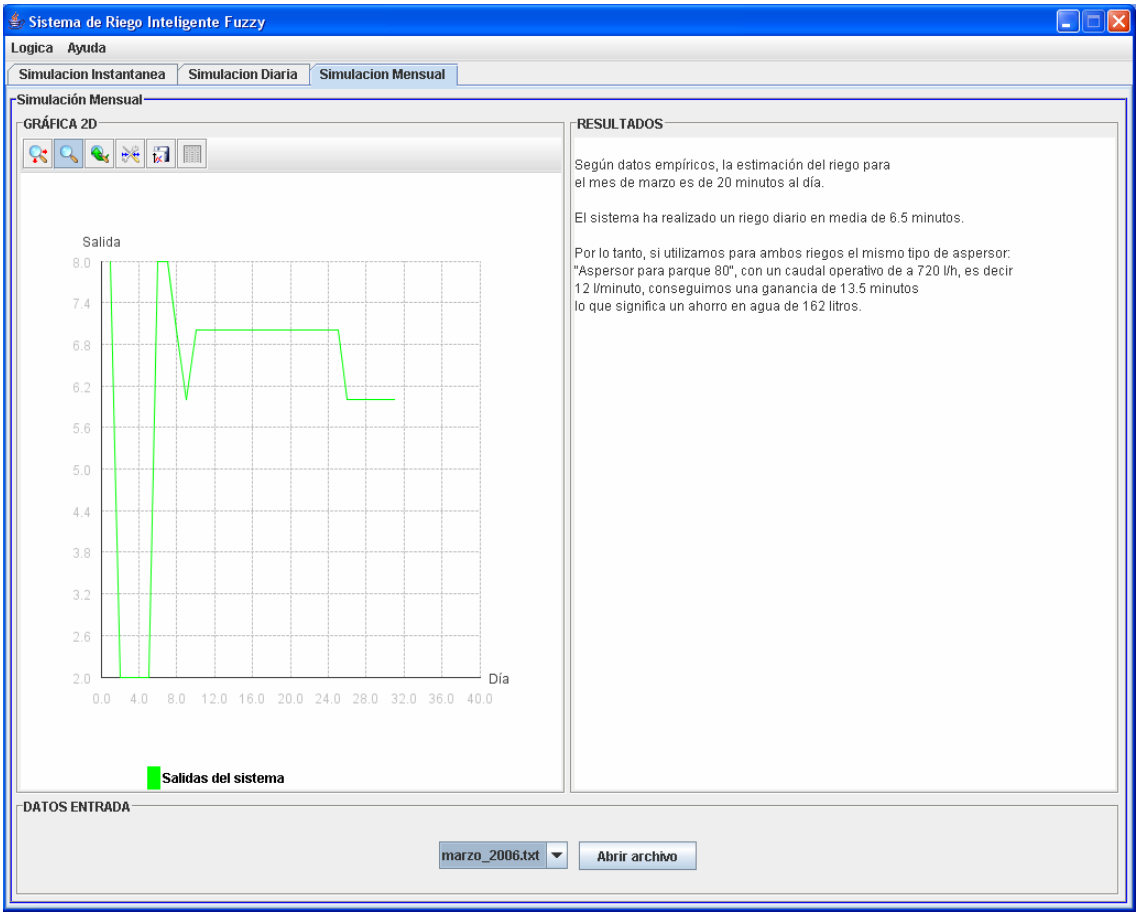
Como se puede ver, ahora el Control de Simulación dispone de un reloj en horas y minutos, un contador de riego, y un panel de programación de horas de riego. Una vez que el reloj llegue a cierta hora programada, el riego se conectará, realizando inferencias periódicamente para optimizar el tiempo de riego.

También se puede observar que se permite variar la velocidad de simulación para poder visualizar de manera más rápida el transcurso del día.

En el panel Web Cam, se permite visualizar a modo de ventana, las condiciones climatológicas, así como el estado del césped para que se pueda visualizar el comportamiento del sistema a modo de supervisor.

Según evolucione el clima, se podrá visualizar los efectos en este panel.

Por último, el panel de Simulación Mensual, permite leer datos meteorológicos desde fichero y realizar una inferencia mensual con todos ellos mostrando la evolución del tiempo de riego a lo largo de un mes y permitiendo visualizar resultados comparativos del Sistema de Riego Inteligente Borroso frente a un control humano clásico.



Como se puede ver, permite seleccionar un fichero localizado en la carpeta “archivos” de la carpeta del código del proyecto. En la gráfica 2D se permite, modificar y ajustar características visuales suministradas por el paquete JMathPlot.

9.2. Instalación XFUZZY

1. Descargarse de la página:
   https://sdlc4a.sun.com/ECom/EComActionServlet/DownloadPage:~:com.sun.sunit.sdlc.content.DownloadPageInfo;jsessionid=3D1583B44E30C72C0D33B9FF64D6DD16;jsessionid=3D1583B44E30C72C0D33B9FF64D6DD16

   La versión para Windows de java JRE 1.3.x

2. Descargarse de la página:
   http://www.imse.cnm.es/Xfuzzy/Xfuzzy_3.0/download_sp.html

   la clase install.class

3. Instalar el java JRE 1.3.x.
4. En la carpeta bin de java jre, copiar la clase install.class
5. Ejecutar en la línea de comandos dentro de la carpeta donde este install.class, en la carpeta bin de java

   Java install install.class

6. Instalar xfuzzy en la carpeta que viene por defecto, es decir, en la carpeta bin de java jre.
7. ir a la nueva carpeta bin, generada al instalar xfuzzy y ejecutar xfuzzy.bat, que ya es el programa.