Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

INTEGRACIÓN DE OPERADORES DE AGREGACIÓN Y MODELOS DE DECISIÓN EN EL SISTEMA DE SOPORTE A LA DECISIÓN FLINTSTONES

Alumno: Jesús Alejandro Benítez Pedrero

Tutor: Prof. Dr. Luis Martínez López
Dpto: Informática

Febrero, 2017

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Dr. D. Luis Martínez López, tutor del Trabajo Fin de Grado titulado: Integración de operadores de agregación y modelos de decisión en el sistema de soporte a la decisión Flintstones, que presenta D. Jesús Alejandro Benítez Pedrero, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Febrero de 2017

El Alumno

El Tutor

D. Jesús Alejandro Benítez Pedrero

Dr. D. Luis Martínez López

Agradecimientos

Esto va dirigido a todas aquellas personas que han aportado su granito de arena para que este proyecto se culmine como finalización de toda mi carrera universitaria en la universidad de Jaén.

Quiero agradecer a mis padres y mi hermano por todo el esfuerzo que han realizado, ya que gracias a su sacrificio diario y nunca dudar de mí he conseguido llegar a este punto.

También he de agradecer la enorme paciencia que tanto Luis Martínez López como Álvaro Labella Romero han tenido conmigo, puesto que gracias a su ayuda se ha llegado a este punto no solo aportando sus conocimientos también aportando su apoyo y revisiones.

Y por último a toda mi familia y amigos que también siempre están ahí aportando consejos y apoyo.

Muchas Gracias a Todos

Índice

Capítulo 1: Introducción

1.1. Introducción al trabajo

La toma de decisiones es una tarea fundamental a la vez que compleja, debido a que tenemos una serie de alternativas y deberemos escoger cual es la mejor dependiendo de las circunstancias en la que nos encontremos en ese momento. Aunque muchas decisiones que tomamos habitualmente son decisiones donde toda la información se conoce de antemano, también nos encontramos con situaciones donde no conocemos toda la información necesaria para elaborar una buena decisión. Para ello existen distintas teorías, herramientas y software que dan soporte a la decisión en aquellas situaciones donde no conocemos toda la información o bien la información es vaga o incierta, como por ejemplo Flintstones. Flintstones es un sistema de soporte a la decisión que nos permite estudiar y analizar problemas de toma de decisión. Este tipo de herramientas sirven de apoyo para las personas que realizan el proceso de toma de decisión, dividiendo este proceso en fases o etapas, resultando más intuitivo para así utilizar la selección de una alternativa u otra. La toma de decisión final siempre la realizará una persona o un grupo de personas.

Flintstones funciona recopilando la opinión de uno o varios **expertos** en función de varios **criterios** para determinar cuál es la mejor **alternativa** u opción de un conjunto de alternativas que define el problema en cuestión. Además dependiendo del modelo de decisión utilizado la solución puede ser diferente; para agrupar las distintas opiniones de los expertos en problemas de decisión en grupo se utilizan operadores de agregación que dependiendo del objetivo perseguido puede obtener resultados diversos.

Herramientas de este tipo hacen que la toma de decisiones con información vaga o incierta, sea más fácil de llevar a cabo.

La finalidad de este trabajo fin de grado es aprender el uso de la tecnología Eclipse RCP, tecnología utilizada en Flintstones, para desarrollar e integrar en él nuevas funcionalidades y operadores de agregación.

1.2. Propósito del trabajo

En este trabajo, se diseñarán, implementarán e integrarán diferentes operadores de agregación y métodos de resolución de la toma de decisión en un sistema de soporte a la decisión para dominios lingüísticos difusos (Flintstones), empleando una metodología de programación basada en componentes con estructura OSGI Equinox [92] en un entorno de programación Eclipse RCP [93].

El principal propósito de este trabajo es aumentar la funcionalidad de Flintstones mediante la integración de nuevos operadores de agregación, esto hará que el sistema adquiera más diversidad a la hora de dar soporte a las decisiones en ambientes con información difusa o incierta. Teniendo en cuenta que este tipo de herramientas nos ayudan a tomar la decisión, en última instancia una persona será la encargada de tomar la decisión.

1.3. Objetivos del trabajo

1. Búsqueda y revisión bibliográfica.

2. Estudio y análisis de posibles operadores de agregación y métodos de resolución que puedan ser incorporados a la herramienta Flintstones.

3. Estudio y análisis de la metodología de programación basada en componentes orientada a una estructura OSGI (Equinox).

4. Análisis, diseño, implementación e integración de los distintos operadores de agregación.

5. Redacción de la memoria de proyecto.

1.4. Planificación temporal

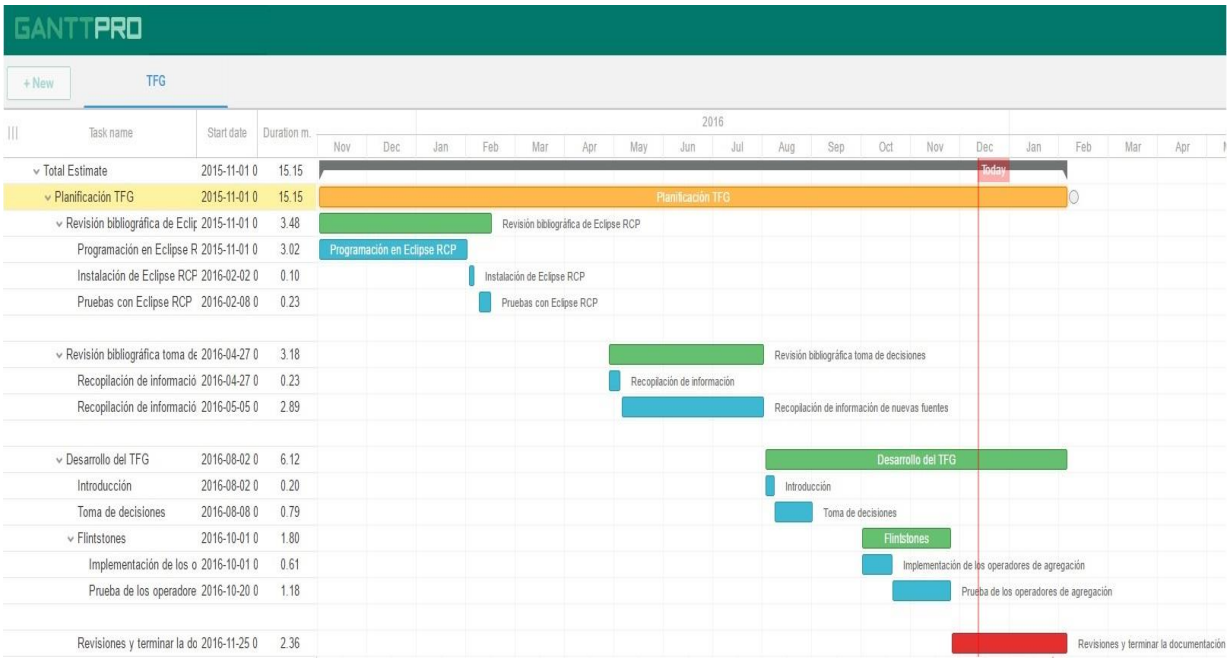

Figura 1.1 Diagrama de Gantt

En la Figura 1.1 tenemos la descomposición de tareas llevadas a cabo para este trabajo; así como su planificación temporal a lo largo del curso académico, desde que se asignó este trabajo.

El trabajo se divide en varias fases como podemos observar. En primera fase se llevará a cabo una revisión bibliográfica de Eclipse RCP, en la segunda fase también se realizará una revisión bibliográfica pero en este caso sobre toma de decisiones.

La fase de desarrollo del TFG nos da una imagen global de cómo va a quedar el trabajo una vez implementado. Esta fase se adapta a la fase de diseño, y es una guía

completa de los pasos a seguir para la consecución de los objetivos en el tiempo predicho con los recursos estimados.

La implementación y las pruebas están íntimamente ligadas, para conseguir cumplir los requisitos no funcionales que se le pueden exigir a este tipo de trabajos.

La memoria es el proceso por el cual se especifican todas las tareas llevadas a cabo para la consecución correcta del proyecto, será una documentación correcta y precisa de todo lo realizado.

Siempre tenemos que tener en cuenta que los cambios que se produzcan en la última fase que observamos en la Figura 1.1, la fase de revisión repercutirán en el tiempo de desarrollo del proyecto, que aunque resulten tediosas son habituales y necesarias en este tipo de trabajo.

1.5. Estructura del trabajo

A continuación, se especifican los capítulos que contendrá este trabajo fin de grado, así como una breve descripción del contenido de cada uno de ellos:

El primer capítulo es una introducción al trabajo fin de grado, en la que hemos presentado la justificación, propósito y objetivos que se desean alcanzar en este trabajo, así como su planificación temporal.

El segundo capítulo introduce los términos teóricos necesarios para entender y ampliar Flintstones, como son la Toma de Decisión lingüística, modelos de computación lingüística y el modelo 2-tupla entre otros.

El tercer capítulo presenta los sistemas de soporte a la decisión, Eclipse RCP, y se explica la arquitectura y tecnología de Flintstones, sus componentes y como se puede añadir nuevas funcionalidades mediante puntos de extensión

El cuarto capítulo es el capítulo dedicado al proceso de ingeniería de software y a la implementación software de los operadores de agregación vistos en el capítulo segundo.

El quinto capítulo es el capítulo dedicado a las conclusiones obtenidas después de llevar a cabo todo el proyecto.

El sexto capítulo este capítulo se especificará toda la bibliografía utilizada para el desarrollo de este trabajo.

Anexo 1 guía paso a paso para crear un nuevo operador de agregación mediante el uso de los puntos de extensión en Flintstones.

Anexo2 Instalación de la versión Flintstones con los operadores de agregación ya integrados y un ejemplo de funcionamiento de un operador de agregación.

Capítulo 2: Toma de decisión

2.1. Introducción a la toma de decisión

La Toma de Decisiones es un proceso habitual el cual se lleva a cabo casi a diario por cualquier ser humano, pero esto no significa que sea un proceso simple. Ya que a menudo puede llegar a ser un proceso complejo. Algunos autores argumentan que la toma de decisiones en situaciones complejas es una característica fundamental que diferencia al género humano de los animales [2], ya que la toma de decisiones hace uso del razonamiento y pensamiento e incluso en algunos casos de experiencias anteriores. La toma de decisiones se compone de una persona o varias que se enfrentan a un problema, en el cual se tiene un conjunto de posibles alternativas u opciones y debe de seleccionarse una como solución al problema. Dependiendo de la información que tengamos sobre el problema la resolución del mismo puede ser un proceso simple o en caso contrario bastante complejo.

La toma de decisión se aplica en distintas disciplinas, tales como, la Psicología [75, 76], la Ingeniería [77, 78, 79], la inteligencia artificial [80, 81, 82], etc. Los diferentes y extensos campos de aplicación donde se puede aplicar la toma de decisión han dado lugar a la existencia de diferentes modelos de toma de decisiones surgiendo así la Teoría de Decisión [83].

Un problema clásico de decisión se constituye de los siguientes elementos básicos:

1. Un conjunto de alternativas o decisiones posibles.
2. Un conjunto de estados de la naturaleza que definen el contexto de definición del problema.
3. Un conjunto de valores de utilidad, cada uno de los cuales está asociado a un par formado por una alternativa y un estado de la naturaleza.
4. Una función que establece las preferencias del experto sobre los posibles resultados.

Con los elementos básicos de un problema de decisión podemos hacer un gráfico como el que se muestra en la Figura 2.1, que representa un esquema de resolución de un problema paso a paso de forma intuitiva.

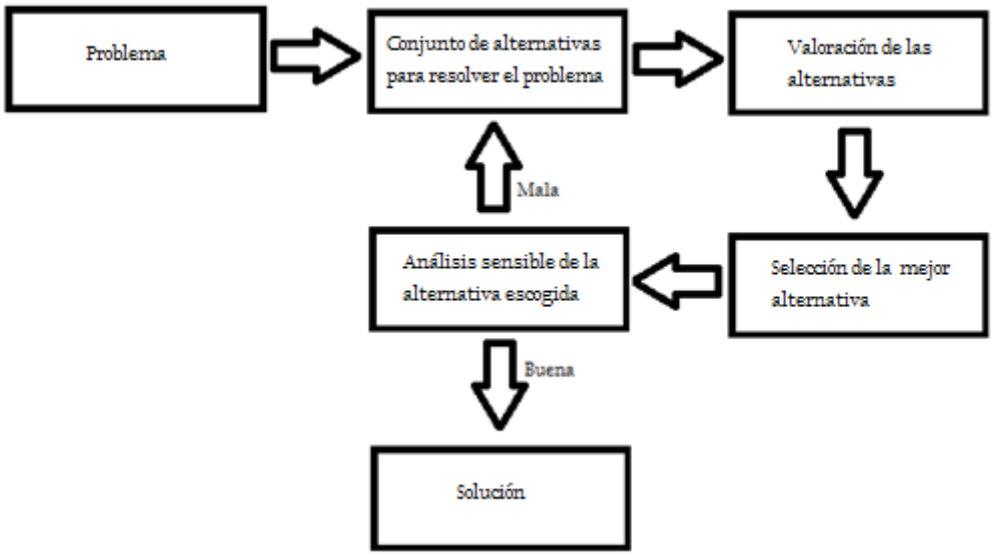

Figura 2.1 Resolución de un problema de toma de decisión de forma intuitiva.

A continuación se hará una descripción de cada etapa desde se plantea el problema de decisión hasta que se llega a su resolución.

En la primera etapa es donde se establece el problema, el cual hay que afrontar para poder solucionarlo, esta etapa es fundamental, ya que hay que tener muy claro cuál es el problema para poder identificarlo correctamente y sentar las bases para escoger la mejor solución. Para esto debemos fijar los objetivos a conseguir para formular correctamente el problema.

En la siguiente etapa se definen las alternativas que pueden solucionar el problema. En esta etapa hay que tener en cuenta que alternativas son viables y cuáles no.

En la tercera etapa consideramos las ventajas e inconvenientes de cada una de las alternativas posibles. Para evaluar las alternativas se usan dos tipos de factores:

- **Factores Cuantitativos**: Suelen ser valores numéricos asociados a las alternativas, como puede ser por ejemplo kilómetros en un problema de distancias.
- **Factores Cualitativos**: No suelen ser fáciles de medir numéricamente, están asociados a opiniones o percepciones como puede ser la comodidad o el confort.

En la cuarta etapa se selecciona la mejor alternativa teniendo en cuenta si se necesita tener una solución aceptable, buena o la mejor de todas.

En la quinta etapa se comprueba si la alternativa escogida en la anterior etapa es fiable y robusta, en caso de que lo sea se pasa a la siguiente etapa, en caso de que sea rechazada, se vuelve otra vez a la segunda etapa y vuelve a realizarse todo el proceso desde ese punto.

En la última etapa se selecciona la mejor alternativa para la resolución del problema.

Además de tener en cuenta los elementos básicos de un problema de decisión hay que tener en cuenta factores que influyen también en este tipo de problemas:

- **Información** que se tiene del problema.
- **Conocimiento** obtenido anteriormente de problemas similares.
- **Experiencia** de resolver diferentes problemas a lo largo de los años.
- **Análisis** de todos elementos que intervienen en el problema.
- **Juicio** para evaluar la mejor alternativa.

2.2. Proceso de toma de decisión

El proceso de Toma de Decisión es un proceso complejo debido a la necesidad de analizar detalladamente las ventajas e inconvenientes asociados a cada alternativa del problema.

Anteriormente se explicó de forma intuitiva cómo se soluciona un problema de decisión. Ahora explicaremos como se realizan estas etapas en el proceso de toma de decisión:

1. Identificar la decisión y los objetos del problema.
2. Identificar los criterios del problema.
3. Recoger la información.
4. Valorar las diferentes alternativas.
5. Selección de la mejor opción.
6. Se prueba la decisión mediante su aplicación.
7. Evaluación de los resultados mediante el análisis sensible.

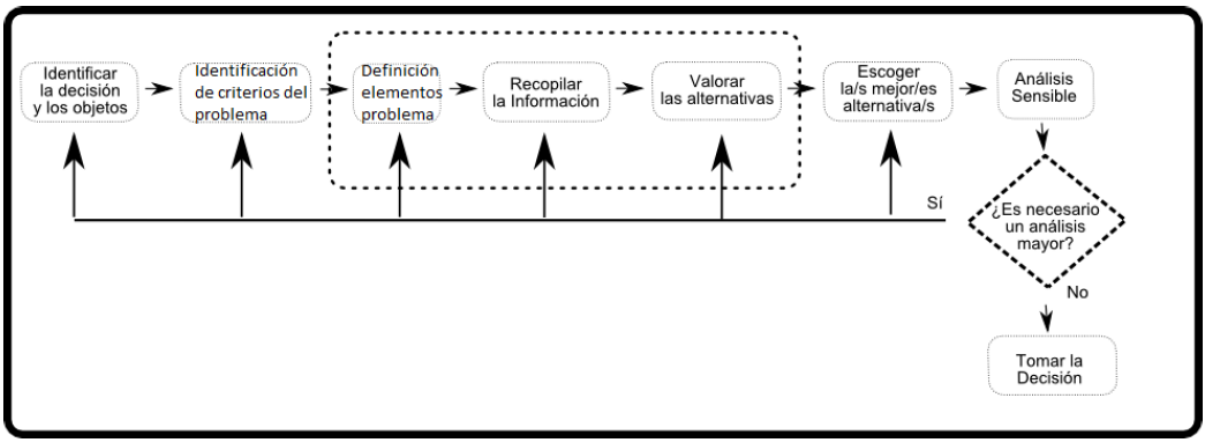

Figura 2.2 Etapas del proceso de toma de decisión.

Estas 7 etapas anteriores las podemos resumir en 5 fases (Figura 2.3) como en [4], esas fases son:

1. ***Inteligencia***: Se observa la realidad, se identifican los problemas, las alternativas y los objetivos a alcanzar en el proceso de resolución.
2. ***Modelado***: Se construye un modelo que define un marco que establece la estructura del problema, las preferencias, la incertidumbre, etc.

3. ***Recopilación de información***: Se obtiene la información, el conocimiento y las preferencias proporcionadas para que se puede tomar una o varias decisiones (dependiendo del problema) de acuerdo con el modelo previamente definido.
4. ***Análisis***: Analiza y agrega la información recopilada de acuerdo con los objetivos y limitaciones, y los resultados de los informes que debe considerarse en la fase de selección.
5. ***Selección***: De acuerdo con los resultados obtenidos en la etapa de análisis se realiza un proceso de explotación en el cual se puede elegir la alternativa o alternativas para la resolución del problema.

Figura 2.3 Proceso de toma de decisión.

En este proceso de toma de decisión tenemos que tener en cuenta además una serie de procesos cognitivos que son:

- Observación: Trata la forma de precisar los objetos o elementos que intervienen en el problema.
- Comparación: Prestar atención en objetos o elementos para así encontrar sus relaciones, similitudes o diferencias.
- Codificación: Convierte un código de formulación a valores numéricos, lingüísticos, etc.
- Organización: Evaluar las distintas alternativas, determinando las fases necesarias y el mejor desarrollo de acción.
- Clasificación: Ordenar atendiendo a algún tipo de criterio un conjunto de objetos o elementos en clases o categorías.
- Resolución: Aplicación de las decisiones escogidas, en esta fase se muestra el resultado que se ha obtenido.
- Evaluación: Se realiza un análisis de los resultados que se han obtenido mediante el proceso de toma de decisión y se alcanza la solución más acorde.
- Retroalimentación: Después de la fase de evaluación, se debe realizar un análisis para así determinar la mejor solución de cara al futuro.

2.3. Clasificación de los problemas de toma de decisión

Los problemas de toma de decisión se clasifican de diferentes formas dependiendo de distintos criterios. En este caso los vamos a clasificar atendiendo al número de personas que toman la decisión y al ambiente o contexto en el que se define el problema.

Según el número de criterios que se utilicen en el problema tenemos problemas de un único criterio o problemas multicriterio de toma de decisión.

En los problemas de un único criterio, cada alternativa es definida por un único valor. Sea $X = \{x_1, x_2,..., x_n\}$ el conjunto de alternativas del problema. De forma que su representación se muestra en la Tabla 2.1

Alternativa	Valoración
$\mathbf{X_1}$	V_1
...	…
$\mathbf{X_n}$	V_n

Tabla 2.1 Esquema de un problema de toma de decisión con un único criterio

Cada entrada x_i le asignamos una valoración v_i, dependiendo del tipo de problema el dominio de cada v_i puede ser numérico, lingüístico, etc.

En los problemas multicriterio, en vez de tener un solo criterio para tomar la decisión, hablamos de varios criterios. Los criterios en un problema de decisión multicriterio deben ser finitos, para así poder abordarlo.

Sea $X = \{x_1, x_2,..., x_n\}$ y $C = \{c_1, c_2,..., c_n\}$ el conjunto de alternativas y el conjunto de criterios respectivamente de un problema concreto. De forma que su representación en la Tabla 2.2.

Alternativa	Criterios			
(x_i)	c_1	c_2	...	c_n
$\mathbf{x_1}$	y_{11}	y_{12}	...	y_{1n}
...	...	...	...	...
$\mathbf{x_n}$	y_{n1}	y_{n2}	...	y_{nh}

Tabla 2.2 Esquema general de un problema de toma de decisión multicriterio

Cada entrada y_{ij}, indica la preferencia de la alternativa x_i, respecto del criterio, c_j. Los problemas de toma de decisión de un único criterio son menos complejos que los problemas multicriterio.

Los problemas de toma de decisión de un experto se conocen como toma de decisiones individual, en los casos de que sean varios expertos nos encontramos con problemas en grupo.

En los casos de un solo experto, en él recaerá la tarea de valorar las diferentes alternativas según sus preferencias. En este caso, tiene el inconveniente que solo un experto es el que se ha encargado de valorar las diferentes alternativas obteniendo una solución respecto de una sola opinión.

En los casos de múltiples expertos, cada experto expresará sus preferencias dando lugar a un conjunto de valoraciones de las diferentes alternativas. Después se realizará un proceso de agregación para que todos los resultados obtenidos se unifiquen en un resultado global.

1. La ventaja de estos casos es que la información es más completa, y esto conlleva a que se incremente la aceptación de la solución.
2. La desventaja es el incremento tiempo y puede producirse el abandono de algún miembro del grupo o que se le exima su responsabilidad.

Los problemas de decisión no sólo se clasifican dependiendo del número de criterios o expertos que toman la decisión también depende de factores como el ambiente en el que se define.

El ambiente, situación o contexto en los cuales se tiene que tomar la decisión, se puede clasificar de la siguiente forma:

- Ambientes de certidumbre: son cuando conocemos todos los elementos y/o factores que intervienen en el problema con exactitud y precisión.
- Ambientes de riesgo: son cuando algunos de los elementos y/o factores que intervienen en el problema no podemos tenerlos con exactitud ni precisión, ya que son elementos y/o factores probabilísticos.
- Ambientes de incertidumbre: son cuando la información que tenemos sobre los elementos y/o factores que intervienen en el problema es vaga o incierta.

El último caso es bastante común en los problemas de toma de decisiones complejos del mundo real de cualquiera de las disciplinas científicas relacionadas con la toma de decisión. Por lo tanto, para resolver los problemas de toma de decisiones en ambientes de incertidumbre, la teoría de la decisión clásica ofrece modelos probabilísticos para gestionar la incertidumbre para este tipo de problemas. En muchos de ellos, sin embargo, es fácil observar que distintos aspectos de incertidumbre tienen un carácter no probabilístico, ya que están relacionados con la imprecisión y la vaguedad de significados expresados, o bien por expertos o bien por los encargados de tomar las decisiones con su conocimiento acerca del problema. En tales situaciones, el uso de descriptores lingüísticos, para expresar el conocimiento y las preferencias sobre las

alternativas o criterios, se utiliza a menudo por los expertos en el problema de toma de decisiones, adquiriendo así el concepto de la Toma de Decisiones Lingüística para indicar que la información utilizada en el problema de toma de decisiones es información lingüística. Los descriptores lingüísticos son los que representan juicios subjetivos en lugar de valores de probabilidad. Por lo tanto, para el modelado y gestión de la incertidumbre inherente en estos juicios y la imprecisión que presentan existen diferentes metodologías y enfoques. Sin embargo, este trabajo se centrará principalmente en el uso de la lógica difusa, teoría de conjuntos difusos, y el enfoque lingüístico difuso para facilitar el modelado y la gestión de la incertidumbre en toma de decisiones lingüística y proporcionar de manera directa la representación de la información lingüística por medio de variables lingüísticas.

2.4. Modelado de preferencias

El modelado de preferencias es el tipo de información que utilizan los expertos para evaluar una alternativa u otra, o bien para dar preferencia a una alternativa sobre otra. Este proceso se realiza en base al conocimiento, experiencia y creencia que cada experto tiene sobre el problema el cual está afrontando. Por lo tanto en cualquier área o disciplina en la cual nos encontremos este proceso siempre se realiza.

El dominio de expresión de preferencia es el dominio que utilizan los expertos para expresar sus preferencias. Cuando los dominios de expresión de preferencias son los mismos para todos los expertos hablamos de problemas de contexto homogéneos y cuando los dominios de expresión son diferentes, es decir los expertos utilizan dominios de información diferentes, hablamos de problemas de contexto heterogéneos.

En los problemas de decisión definidos en contextos heterogéneos, lo normal es que los expertos expresen sus preferencias en un modelo de representación que conozcan o bien que sea cercano sus disciplinas o campos de trabajo. Por ejemplo, cuando hablamos de expertos de áreas técnicas lo normal es que expresen sus preferencias mediante valores numéricos, ya que es el modelo de representación con el que se sienten más cómodos. Sin embargo, expertos que no pertenecen a disciplinas técnicas o cuando la información disponible es vaga o incierta, se puede preferir utilizar un lenguaje natural o bien palabras o etiquetas lingüísticas. También se puede dar el caso, que los expertos no tengan suficiente conocimiento para asignar valores numéricos exactos y por lo tanto se recurra a valores intervalares.

Cuando adaptamos el modelado de preferencias al contexto del problema, los expertos se sienten más cómodos y seguros expresando su información. Por lo tanto, cuando tenemos la solución, ésta tiene más garantías de éxito que otras soluciones en las cuales no se ha utilizado el modelado de preferencias adecuado.

El modelado de preferencias es la forma en la que los expertos expresan sus preferencias, es una fase dentro de la toma de decisión en la que podemos distinguir:

1. La estructura de información utilizada por los expertos para la representación de sus preferencias.
2. El dominio de la información en el que se expresan las preferencias sobre el conjunto de alternativas al problema.

Elegir un dominio u otro que se adecue a la resolución del problema puede deberse a varios motivos que son los siguientes:

1. Naturaleza cuantitativa o cualitativa de la información, cuando la información es cuantitativa se utilizan dominios numéricos, ya que son más adecuados, mientras que si la información que tenemos es cualitativa lo más adecuado es utilizar dominios lingüísticos.
2. Pertenencia de los expertos a diferentes áreas de conocimiento, los expertos adaptará el dominio de información a sus áreas de trabajo, para así poder expresar mejor sus opiniones.
3. Expertos con diferente grado de conocimiento sobre el problema, aquí los expertos con experiencia adaptaran mejor unos dominios que otros en base a sus experiencias anteriores con problemas similares.

Los tres tipos de dominio de información más habituales para expresar preferencias son:

1. Dominio numérico, donde se expresa las preferencias con valores numéricos exactos. Hay dos variantes que son:
   a. Numérico binario.
   b. Numérico normalizado en el intervalo [0,1].
2. Dominio intevalar, dado que existe incertidumbre, porque la información es vaga o incierta, existen modelos intervalares capaces de recoger dicha incertidumbre de forma eficaz.
3. Dominio lingüístico, en este caso también existe incertidumbre, y normalmente también la naturaleza de la información es cualitativa, por lo tanto se utilizan términos lingüísticos.

2.5. Toma de decisiones lingüística

Los problemas de toma de decisión a los que nos enfrentamos en la vida diaria pueden ser muy complejos, ya que no tenemos información suficiente, o ésta es vaga e imprecisa; por lo tanto este tipo de problemas no se adaptan correctamente a los modelos clásicos y necesitamos recurrir a modelos de incertidumbre para así poder afrontar este tipo de problemas. Existen diferentes modelos que podemos utilizar para adaptar nuestro problema a este tipo de información, como son los intervalos, grados de creencias, etc.

El uso de información lingüística es bastante común en problemas de toma de decisiones bajo incertidumbre, a consecuencia de esto en la teoría de la decisión se origina el concepto de toma de decisiones lingüística. Yager [6, 7] señaló que una de las razones de porque la información lingüística es tan útil en estas situaciones de decisión es porque la experiencia muestra que:

"*Una estrategia realista para la toma de decisiones bajo un ambiente con un alto grado de incertidumbre consiste en describir la información incierta por medio de lenguaje natural, ya que en muchas situaciones reales de decisión los valores que se utilizan para la evaluación de valoraciones y evaluar la importancia de alternativas / criterios de un problema de toma de decisiones se hace a partir de una escala lingüística que facilita al decisor la expresión de las valoraciones.*"

En problemas de toma de decisiones con ambiente de incertidumbre, el uso de descriptores lingüísticos es una herramienta útil, sencilla y natural para representar las preferencias debido al contexto del problema al cual nos enfrentamos. Para modelar y gestionar la incertidumbre inherente y la vaguedad de descriptores lingüísticos, el enfoque lingüístico difuso [8, 9], basado en la teoría de conjuntos difusos ha sido ampliamente utilizado [10]. Por lo tanto la toma de decisiones lingüística podría utilizar el enfoque lingüístico difuso en su proceso de resolución siempre que su representación difusa fuera adecuada para situaciones de decisión.

2.5.1. Enfoque lingüístico difuso

Un enfoque común para modelar la información lingüística es el Enfoque Lingüístico Difuso [9] que utiliza la teoría de conjuntos difusos [11] para gestionar la incertidumbre y el modelo de información lingüística utilizando el concepto de variable lingüística.

Zadeh [9] introdujo el concepto de variable lingüística como "*una variable cuyos valores no son números, sino palabras o frases en un lenguaje natural o artificial*." Un valor lingüístico no es tan preciso como un valor numérico pero para problemas con incertidumbre utilizar un lenguaje natural puede ser más adecuado para las personas que usar números. Por lo tanto, el uso de valores lingüísticos en este tipo de problemas con ambiente de incertidumbre es habitual. Formalmente una variable lingüística se define como sigue.

Definición [12]. Una variable lingüística se caracteriza por una quíntupla (*H*, *T (H)*, *U*, *G*, *M*) en la que *H* es el nombre de la variable; *T (H)* (o simplemente *T*) denota el conjunto de términos de *H*, es decir, el conjunto de nombres de valores lingüísticos de *H*, con cada valor difuso indicado genéricamente por *X* que se definen en el universo del discurso *U* asociado; *G* es una regla sintáctica (que por lo general toma la forma de una gramática) para generar los nombres de los valores de *H*; y *M* es una regla semántica para asociar su significado con cada uno de *H*, *M(X)*, es un subconjunto difuso de *U*.

Para el uso de variables lingüísticas debemos de seleccionar aquellos descriptores lingüísticos que mejor se adapten al conjunto de términos del problema en el cual nos encontramos, incluyendo el análisis de su nivel de incertidumbre (granularidad), y su sintaxis y semántica. La primera, comúnmente se nota como $g + 1$, determina el nivel de discriminación entre los diferentes cargos de la incertidumbre modeladas por los descriptores lingüísticos en el conjunto de términos lingüísticos, $S = \{S_0 ... S_g\}$. Alta granularidad significa un alto nivel de discriminación, sin embargo, baja granularidad significa un bajo nivel de discriminación. Esta selección depende del problema en el Enfoque Lingüístico Difuso. Por otro lado, la selección de la semántica de sintaxis adecuada es crucial para determinar la validez del Enfoque Lingüístico Difuso. Existen diferentes enfoques para elegir los descriptores lingüísticos y diferentes maneras de definir su semántica lingüística. A continuación podemos observar un ejemplo (Figura 2.4) de una semántica lingüística [9, 13, 14].

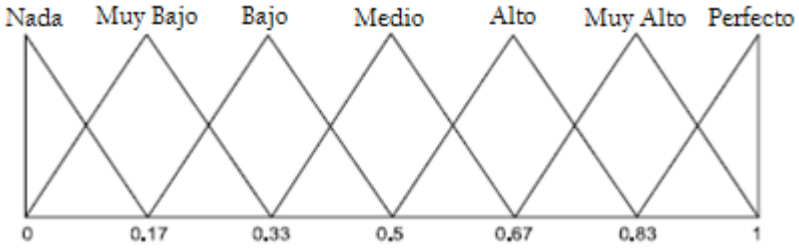

Figura 2.4 Un conjunto de siete estados con su semántica.

Los principales enfoques para seleccionar los descriptores lingüísticos son:

1. Enfoque de estructura ordenada: Se define el término lingüístico establecido por medio de una estructura ordenada que proporcionan el conjunto de términos, S, distribuidos en una escala en la que se define un orden total [14]. Por ejemplo, un conjunto de siete términos S, se podría definir como (Figura 2.4).

   $S = \{S_0$: nada (n), S_1: muy bajo (mb), S_2: bajo (b), S_3: medio (m), S_4: alto (a), S_5: muy alto (ma), S_6: perfecto (p)$\}$.

   Por lo general, en estos casos, la existencia de los siguientes operadores es necesaria

   Un operador de negación: $Neg(S_i) = S_j$ de tal manera que $j = g-i(g+1$ es la cardinalidad).

   Un operador de maximización: $max(S_i, S_j) = S_i$ sí $S_i \geq S_j$

   Un operador de minimización: $min(S_i, S_j) = S_i$ sí $S_i \leq S_j$

2. Enfoque de la gramática libre de contexto: Se define el término lingüístico establecido por medio de una gramática libre de contexto, *G*, de tal manera que los términos lingüísticos son frases generadas por *G* [8, 9, 15]. Una gramática *G* es una 4-tupla (V_N, V_T, I, P), Siendo V_N el conjunto de símbolos no terminales, V_T el conjunto de símbolos terminales, I el símbolo de partida, y P las normas de producción que puede definirse en está gramática libre de contexto. Por ejemplo, entre el V_T y V_N términos primarios como por ejemplo {pobre, regular o bueno} y con los delimitadores {no, muy} que se pueden encontrar en esta gramática. Por lo tanto la elección de I para ser el símbolo inicial, de cualquiera de los términos, y el uso de un conjunto P de términos lingüísticos da lugar a un conjunto S = {peor, muy bueno, no bueno,...} que se pueden generar con este tipo de gramáticas.

De acuerdo con la definición anterior, las variables lingüísticas asocian un significado con la sintaxis de los términos lingüísticos. Existen tres tipos de posibles definiciones de semántica para un conjunto de términos lingüísticos que son:

1. Semántica basada en funciones de pertenencia y una regla semántica: Este enfoque supone que el significado de cada término lingüístico se da por medio de un subconjunto difuso definido en el intervalo [0,1], que se describe por funciones de pertenencia. Este enfoque se utiliza cuando la semántica de los descriptores lingüísticos se generan por medio de una

gramática generativa. Por lo tanto, se establece por medio de dos elementos [8, 9]:

a. Los conjuntos difusos primarios asociados con los términos lingüísticos primarios.
b. La regla semántica *M* para generar conjuntos difuso no primarios en el conjunto difuso.

2. Semántica basada en una estructura ordenada del conjunto de términos lingüísticos: Este enfoque presenta la semántica de la estructura definida sobre el conjunto de términos lingüísticos. Esto sucede cuando los usuarios proporcionan sus evaluaciones mediante el uso de un conjunto ordenado de términos lingüísticos. Bajo esta semántica se acercan a la distribución de los términos lingüísticos en la escala [0,1] puede ser igualmente informativa simétrica [14] o no simétrica [16, 17].

3. Semántica mixta: En este enfoque todos los términos lingüísticos se consideran términos primarios. Asume elementos de los enfoques anteriores, es decir, una estructura ordenada de los términos lingüísticos primarios y los conjuntos difusos de semántica de los términos lingüísticos. Al igual que en la semántica sobre la base de estructura ordenada, conjuntos lingüístico de términos ordenados, se asume que se distribuyen en una escala suponiendo que cada término lingüístico es igualmente informativo. Por otra parte, como en la semántica sobre la base de la estructura ordenada del conjunto de términos lingüísticos, se define la semántica de los términos lingüísticos primarios por medio de los conjuntos difusos [18].

Por lo tanto, la semántica de los términos está representada por números difusos, que se describen por funciones de pertenencia. Las evaluaciones lingüísticas dadas por los usuarios son simples aproximaciones. Una manera de caracterizar un número difuso es utilizar una representación basada en parámetros de su función de pertenencia [19], en la sección 2.5.3 se explica el término función de pertenencia. Algunos autores consideran que las funciones de pertenencia paramétricas (trapezoidales, triangulares) son lo suficientemente buenas para capturar la vaguedad de estas evaluaciones lingüísticas [20]. La representación trapezoidal se consigue por la 4-tupla (a, b, d, c) en la que b y d indican el intervalo en el que el valor de asociación es 1, con a y c que indica los límites izquierdo y derecho de dominio definición de la función de pertenencia trapezoidal. Un caso particular de este tipo de representación son las evaluaciones lingüísticas cuyas funciones de pertenencia son triangulares, esto es $b = d$, por tanto, la representación de este tipo de función de pertenencia de 3-tuplas (a, b, c).

2.5.2. Proceso de toma de decisión lingüística

El proceso de toma de decisiones lingüística lo utilizamos cuando nos enfrentamos a problemas con incertidumbre y siempre que sea conveniente, pero tenemos que tener en cuenta que este proceso de resolución de problemas cambia respecto a un proceso de toma de decisión clásico, por lo tanto a continuación se comparan para ver sus diferencias.

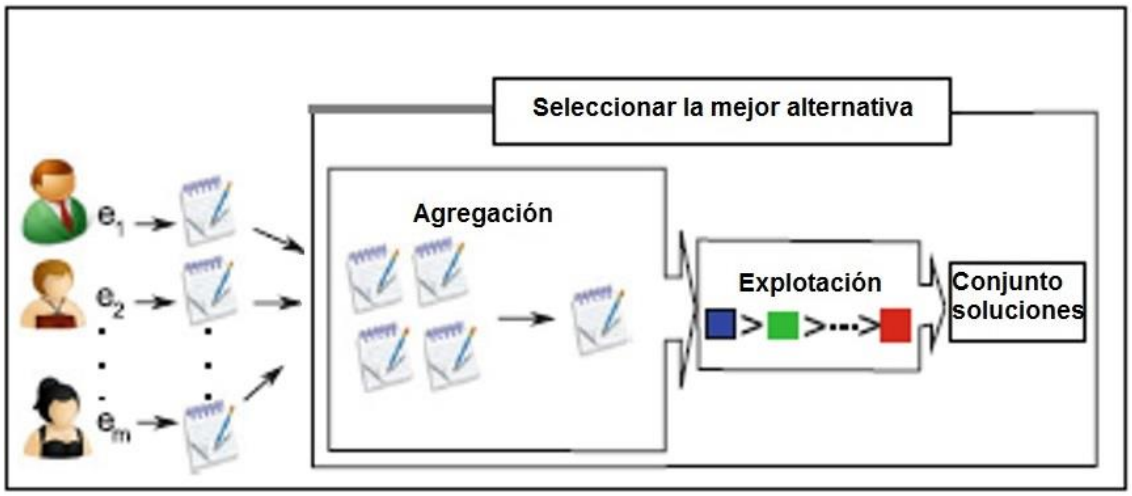

Figura 2.5 Esquema del proceso clásico de toma de decisiones.

Como podemos apreciar en la Figura 2.5 el proceso de toma de decisiones clásico consiste en dos fases principales [26]:

1. Una fase de agregación que se agregan los valores proporcionados por los expertos para obtener una evaluación colectiva de las alternativas posibles.
2. Una fase de explotación de las evaluaciones colectivas para clasificar, ordenar, o escoger la mejor opción u opciones entre las alternativas posibles.

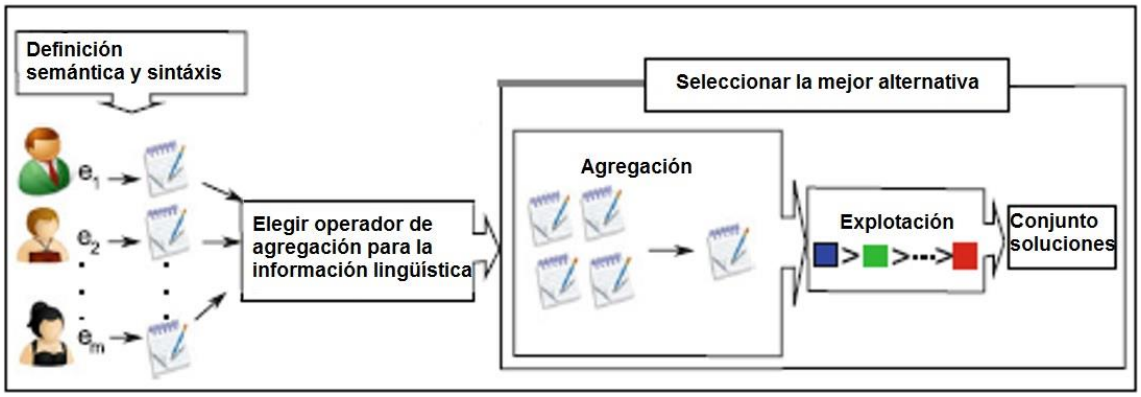

Figura 2.6 Esquema del proceso de toma de decisiones lingüística.

El proceso de toma de decisiones clásico se ve modificado cuando se hace uso de la información lingüística, como podemos apreciar (Figura 2.6) mediante la introducción de dos nuevos pasos [27]:

1. La elección del conjunto de términos lingüísticos con su semántica. Establece el dominio de expresión lingüística en la que los expertos ofrecen sus evaluaciones lingüísticas sobre las alternativas de acuerdo con sus conocimientos.
2. La elección del operador de agregación de información lingüística. Un operador de agregación lingüístico adecuado se elige para la agregación de las evaluaciones lingüísticas. La idoneidad del operador depende de cada problema de decisión individual, también podemos tener distintas

soluciones a un problema utilizando distintos tipos de operadores de agregación, ya que nos puede interesar en algún problema la diversidad de soluciones más que la calidad de la misma.

3. Fase de agregación. Se obtiene evaluaciones colectivas lingüísticas mediante la agregación de las evaluaciones lingüísticas proporcionadas por los expertos utilizando el operador de agregación lingüístico adecuado, aunque no siempre tengamos la certeza de que el operador usado sea el más adecuado, ya que a no ser que nos enfrentemos a problemas similares donde la experiencia y/o conocimiento del experto nos sirva de utilidad, no podemos asegurar que el operador seleccionado sea el más adecuado.
4. Fase de explotación. Se obtiene un ranking para elegir la mejor alternativa o alternativas a partir de las evaluaciones colectivas lingüísticas.

Después de ver el proceso de toma de decisión lingüística observamos que es necesario el uso de modelos computacionales lingüísticos para poder tratar con la información lingüística para así poder solucionar problemas con incertidumbre.

La metodología que destaca sobre otras para trabajar con información lingüística es la metodología de computación por palabras.

El concepto de la computación por lo general implica procesos de cálculo, ya sea por medios matemáticos con números y símbolos o bien mediante un ordenador. Cuando este concepto lo asociamos con las personas, hablamos de razonamiento en vez de proceso computacional, este razonamiento se expresa mediante palabras o lo que es lo mismo se expresa mediante información lingüística [28]. Por lo tanto, la Computación con Palabras aplica la misma visión de sus procesos computacionales cuyo objetivo es obtener resultados lingüísticos a partir de entradas lingüísticas.

Dado que las palabras tienen una representación difusa cuando son utilizados para hacer procesos de computación, el paradigma de la computación con palabras fue definido como una rama de la lógica difusa por Zadeh [28] en la que computación con palabras se definió como "*una metodología en la que las palabras se utilizan en lugar de números para realizar procesos de computación y de razonamiento.*" Más tarde Zadeh [29] añadió que "*Computación con palabras es una metodología en la que los objetos de cómputo son palabras y proposiciones extraídas de un lenguaje natural*".

Es necesario aclarar qué la computación con palabras tiene sentido como metodología para resolver problemas complejos en los que los objetos lingüísticos son objeto de cálculo. Zadeh [30] ofrece su punto de vista sobre la utilidad de la computación con palabras para resolver estos problemas:

"*Los seres humanos tienen muchas capacidades notables. Entre ellos hay dos que se destacan en importancia. En primer lugar, la capacidad de conversar, comunicarse, razonar y tomar decisiones racionales en un ambiente de imprecisión, incertidumbre y con información incompleta. Y en segundo lugar, la capacidad de realizar una amplia variedad de tareas físicas y mentales sin ningún tipo de medidas y ni cálculo. En gran medida, la computación con palabras se inspira en estas capacidades.*"

De los párrafos anteriores se puede deducir que la computación con palabras se basa en tres principales razones [30]:

1. Gran parte del conocimiento humano está descrito lingüísticamente.
2. Las palabras son menos precisas que los números, por lo tanto, la Computación con Palabras podría ser una poderosa herramienta para manejar información imprecisa.
3. La precisión tiene un costo. Si hay tolerancia a la imprecisión en un problema, ésta puede ser explotada mediante el uso de las palabras en lugar de números.

Es importante indicar que los fundamentos de la computación con palabras se establecieron mucho antes de la propia metodología de computación con palabras [9, 31]. En dichos trabajos se definieron los conceptos de variable lingüística, granulo, restricción difusa, propagación de restricciones difusas, etc. Por lo que, el papel fundamental de la lógica difusa en computación con palabras se vuelve aún más claro.

La computación con palabras se basa en la capacidad humana para realizar diferentes tareas sin necesidad de ninguna medida numérica precisa y tal capacidad se sustenta en la capacidad del cerebro para manipular diferentes percepciones, por lo general imprecisas, inciertas, o parciales, que desempeñan un papel clave en los procesos de decisión que necesitan conocimiento o preferencias subjetivas para tomar una decisión por medio de procesos de computación y razonamiento [32].

Por lo tanto, un aspecto crucial de la computación con palabras consiste en que sus procesos, deben obtener entradas lingüísticas para proporcionar resultados comprensibles mediante información lingüística; es decir, que la computación con palabras lleva a cabo un proceso de cálculo a partir de palabras y obtiene resultados expresados lingüísticamente. Esta idea, se muestra en la Figura 2.7, y se ha desarrollado en la toma de decisión desde principios de los 80, cuando los diferentes investigadores, incluyendo Tong y Bonissone [33], Schmucker [34], y Yager [6] comenzaron a proponer diferentes esquemas de computación para trabajar con información lingüística en toma de decisiones.

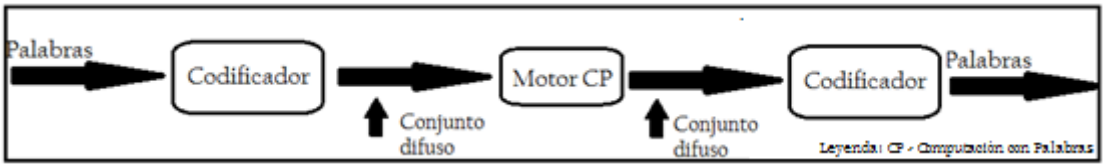

Figura 2.7 Esquema de computación con palabras.

Estos esquemas son bastante similares y tienen una estructura en la que la información lingüística de entrada debe ser asignada a los modelos de conjuntos difusos y los resultados deben ser expresados en información lingüística fácilmente comprensible para las personas.

Yager señaló la importancia de los procesos de traducción y retraducción en la computación con palabras [35, 36] como podemos ver en la Figura 2.8. Primero traduce las entradas lingüísticas manipulándolas en un formato máquina basado en herramientas difusas en el que los cálculos se llevan a cabo, y a continuación se hace una conversión de

los resultados de computación en información lingüística de nuevo para facilitar la comprensión humana (que es uno de los principales objetivos de la computación por palabras).

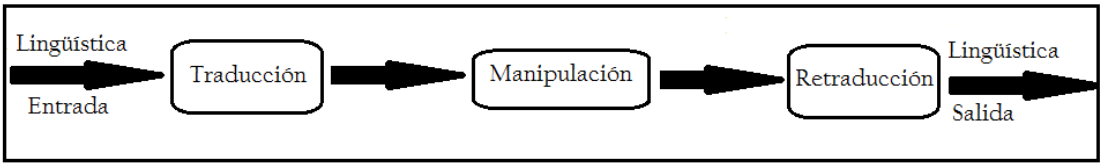

Figura 2.8 Esquema de computación con palabras de Yager.

En consecuencia, diferentes modelos de computación lingüísticos se han desarrollado y aplicado como base de cálculo para la computación con palabras en toma de decisiones lingüística [37, 38, 39, 40, 41]. Los modelos computacionales lingüísticos clásicos que han sido ampliamente utilizados en toma de decisiones lingüística se revisarán en la siguiente sección.

2.5.3. Modelos lingüísticos computacionales

Usar información lingüística y el enfoque lingüístico difuso implica la necesidad de realizar procesos de computación con palabras.

Antes de seguir con la revisión de los modelos computacionales lingüísticos clásicos, definiremos los conceptos de función de pertenencia y principio de extensión en la lógica difusa, para así entender mejor los siguientes apartados.

Un conjunto lo podemos definir como una colección de objetos que tienen una serie de características similares o simplemente se desea encapsular formando un solo objeto.

Los conjuntos introducen una noción fundamental de dicotomía. Básicamente cualquier proceso de dicotomización es una clasificación binaria, donde la decisión de aceptar se nota como "1" y la de rechazar como "0". Por lo tanto, una decisión de clasificación puede expresarse a través de una función característica.

Sea A un conjunto en el universo X, la función característica asociada a A, A (x), x ϵ X, se define como:

$$A(x) = \begin{cases} 1, & si\ x \in A \\ 0, & si\ x \notin A. \end{cases} \tag{2.1}$$

La función A: X —› {0, 1}, los objetos del universo X tienen una restricción, ya que tienen un límite bien definido a la hora de ser asignados al conjunto A. Mediante los conjuntos difusos la restricción la podemos suavizar, puesto que en la función característica admiten valores intermedios, y la función característica pasa a denominarse **función de pertenencia**.

Esto permite que la interpretación sea más realista, ya que el mundo real no tiene marcado unos límites. Por ejemplo cuando se habla del grado de satisfacción de un producto, el cual satisface nuestras necesidades pero es mejorable, por lo tanto le asignamos 0,8 que es un número real que se encuentra dentro del intervalo [0, 1], donde cuanto más cercano a 1 sea el grado, mayor será la pertenencia al conjunto y por caso contrario cuanto más cercano sea a 0, menor será la pertenencia al conjunto.

La función de pertenencia no es una función trivial como en los conjuntos clásicos, por lo que hay que definirla. En principio cualquier forma de la función $\mu_{\tilde{A}}: X \longrightarrow [0,1]$, describe una función de pertenencia asociada a un conjunto difuso $\tilde{A}$ que depende no sólo del concepto que representa, sino también del contexto en el que se usa.

A continuación observaremos un ejemplo de una función de pertenencia, tenemos un concepto que es el de *futbolista de élite*, en un contexto donde la edad gracias a la calidad de vida se encuentra en el intervalo [1, 70]. Así que un jugador cuya edad sea mayor o igual a 15 años se puede considerar futbolista de élite y se le asignaría un valor de 1 a su grado de pertenencia al conjunto difuso de futbolistas de élite. Por el contrario si un jugador tiene una edad igual o superior a 35 años no puede considerarse como un futbolista de élite, y de ahí que se le asigne un valor 0 al grado de pertenencia al conjunto difuso de futbolista de élite. La cuantificación del resto de valores puede realizarse mediante una función de pertenencia $\mu_{\tilde{O}}: X \longrightarrow [0,1]$ que caracteriza el conjunto difuso $\tilde{O}$ de futbolistas de élite en el universo $U = [1, 70]$.

$$\mu_{\tilde{O}}(x) = \begin{cases} 0 & x \in [1, 15] \\ 1 & x \in (15, 35) \\ 0 & x \in [35, 70]. \end{cases}$$

Figura 2.9 Ejemplo de función de pertenencia.

El **principio de extensión** es un concepto básico de la Teoría de Conjuntos Difusos utilizado para generalizar conceptos matemáticos no difusos a conjuntos difusos. A lo largo del tiempo han aparecido diferentes formulaciones de este concepto [84, 85] que se puede definir como:

Sea x el producto cartesiano de los universos $X_1,\ldots, X_r$ y sean $\tilde{A}_1,\ldots, \tilde{A}_r$, r conjuntos difusos en $X_1,\ldots, X_r$ respectivamente. Sea f una función definida desde el universo X, ($X = X_1 \times \ldots \times X_r$), al universo Y, $y = f(x_1,\ldots, x_r)$. El principio de extensión nos permite definir un conjunto difuso $\tilde{B}$ en Y, a partir de los conjuntos difusos $\tilde{A}_1,\ldots, \tilde{A}_r$ representando su imagen a partir de la función f, de acuerdo a la siguiente expresión,

$$\tilde{B} = \{(y, \mu_{\tilde{B}}(y))/y = f(x_1, \ldots, x_r), (x_1, \ldots, x_r) \in X\}$$

donde

$$\mu_{\tilde{B}}(y) = \begin{cases} \sup_{(x_1,\ldots,x_r)\in f^{-1}(y)} min\{\mu_{\tilde{A}_1}(x_1), \ldots, \mu_{\tilde{A}_r}(x_r), & si \quad f^{-1}(y) = \emptyset \\ 0, & en\ otro\ caso \end{cases}$$

(2.2)

2.5.3.1. Modelo Computacional Lingüístico Basado en el Principio de Extensión

Este modelo computacional además de denominarse modelo computacional lingüístico basado en el principio de extensión también los podemos encontrar como modelo semántico porque utiliza términos asociadas a la semántica de los términos lingüísticos, para realizar las operaciones con palabras utilizándose en este tipo de modelo la aritmética difusa basada en el principio de extensión [10]. La aritmética difusa proporciona un resultado a la computación difusa sobre un conjunto de n etiquetas lingüísticas en el conjunto de términos, T (H), un número difuso, F, que normalmente no coincide con ninguna etiqueta lingüística en T (H). Este modelo de computación tiene las siguientes características:

- En aquellos problemas donde la precisión es más importante que la interpretabilidad, los resultados se expresan a través de los mismos números difusos utilizando procedimientos de ordenación difusa para obtener una orden final de las alternativas [42, 43].

- Si la interpretación es más importante, para obtener resultados lingüísticos se necesita una función de aproximación, App, que es aplicada a resultados difusos F para obtener una etiqueta en T (H).

$$T(L)^n \xrightarrow{\tilde{F}} F(\mathbb{R}) \xrightarrow{app_1(\cdot)} T(L) \tag{2.3}$$

El proceso de aproximación implica una pérdida de información y la falta de precisión de los resultados como se muestra claramente en la Figura 2.10.

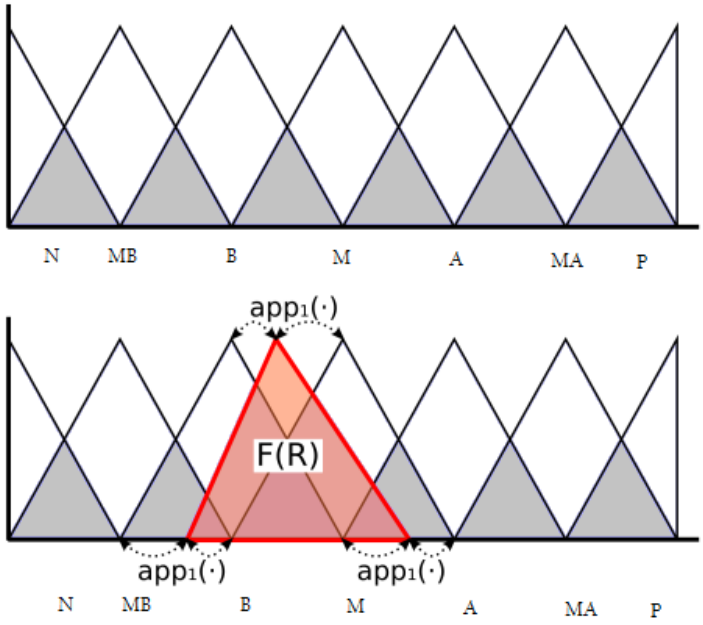

Figura 2.10 Modelo Computacional Lingüístico Basado en el Principio de Extensión.

2.5.3.2. Modelo Computacional Lingüístico Simbólico

Para operar con palabras tenemos un modelo computacional clásico del enfoque lingüístico difuso que es el denominado modelo computacional lingüístico simbólico, el cual utiliza la estructura ordenada del conjunto de términos lingüísticos $S = \{S_0, S_1,...S_g\}$ donde $S_i < S_j$ si $i < j$, para llevar a cabo los procesos de computación.

Utilizando la estructura ordenada del conjunto de términos lingüísticos para llevar a cabo la computación simbólica en tales escalas lingüísticas ordenadas, se proponen los operadores clásicos máximo, mínimo, y negación.

Con este modelo se obtiene resultados lingüísticos de fácil comprensión, pero su exactitud es cuestionable, ya que proporcionan resultados basados en extremos y los valores intermedios no se tienen en cuenta, pudiéndose producir datos anómalos muy altos o muy bajos.

Además de los operadores para modelos simbólicos que acabamos de ver basados en escalas ordinales y operadores Max-min también hay un modelo basado en combinaciones convexas, el cual posee una extensa colección de operadores de agregación mediante el uso de una combinación convexa de etiquetas lingüísticas [44], que actúa directamente sobre los índices de la etiqueta.

El inconveniente de este tipo de modelo simbólico basado en combinaciones convexas es que considera impar la cardinalidad del conjunto de términos lingüísticos y que las etiquetas lingüísticas se colocan simétricamente en torno a un término medio.

Una vez revisados los modelos computacionales lingüísticos clásicos, es conveniente analizar y caracterizar brevemente algunas limitaciones sobre su manera de llevar a cabo las operaciones y su precisión con el fin de caracterizar dichos modelos.

- Computacionales: modelos computacionales basados en el principio de extensión que utilizan aritmética difusa para llevar a cabo sus operaciones de obtención de resultados difusos, esto produce pérdida de información porque el modelo de representación de la información del enfoque lingüístico difuso es discreto en un dominio continuo. Por su parte los modelos simbólicos no operan en valores difusos pero finalmente obtienen un resultado difuso. Por lo tanto, los últimos modelos utilizan una metodología operativa más simple que los anteriores, lo que facilita la computación con la información lingüística.
- Precisión: Los modelos basados en Principio de Extensión obtienen valores precisos difusos como resultado de la computación aritmética difusa pero por lo general no coinciden con los términos lingüísticos iniciales y un proceso de aproximación es necesario para proporcionar salidas lingüísticas. Sin embargo, los modelos simbólicos necesitan un proceso de aproximación de acuerdo con los valores extremos para obtener salidas directamente lingüísticas. De ahí que en los antiguos modelos se pueden obtener resultados más precisos pero no son interpretables lingüísticamente, por lo tanto, ambos modelos necesitan un proceso de aproximación para obtener resultados lingüísticos.

2.5.4. Modelo lingüístico de 2-Tupla

El modelo lingüístico de 2-Tuplas fue presentado en [37] con el objetivo de que la computación con palabras obtuviera mayor precisión, además de poder expresar de forma simbólica cualquier resultado en el universo del discurso. Este modelo se ha utilizado también de una forma ventajosa para el tratamiento de la información lingüística multigranular [45], lingüística no balanceada [47] e información heterogénea (numérica, intervalar y lingüística) [48].

2.5.4.1. Representación

La representación del modelo lingüístico de 2-Tupla se fundamenta en el concepto de traslación simbólica, que a continuación definiremos.

Sea $S = \{S_0,...S_g\}$ un conjunto de términos lingüísticos, y $\beta \in [0, g]$ un valor obtenido por una operación simbólica.

La traslación simbólica de un término lingüístico $S_i \in S = \{S_0,...S_g\}$ es un valor numérico definido en $[-0.5, 0.5)$ que representa la "diferencia de información" entre una cantidad de información $\beta \in [0, g]$ obtenida de una operación simbólica y el índice del término lingüístico más cercano.

A partir de este concepto es donde se desarrolla un nuevo modelo de representación para la información lingüística, el cuál usa como base de representación una 2-Tupla, (S_i,α), donde $S_i \in S = \{S_0,...S_g\}$ representa la etiqueta lingüística y α es un valor numérico que representa la traslación simbólica.

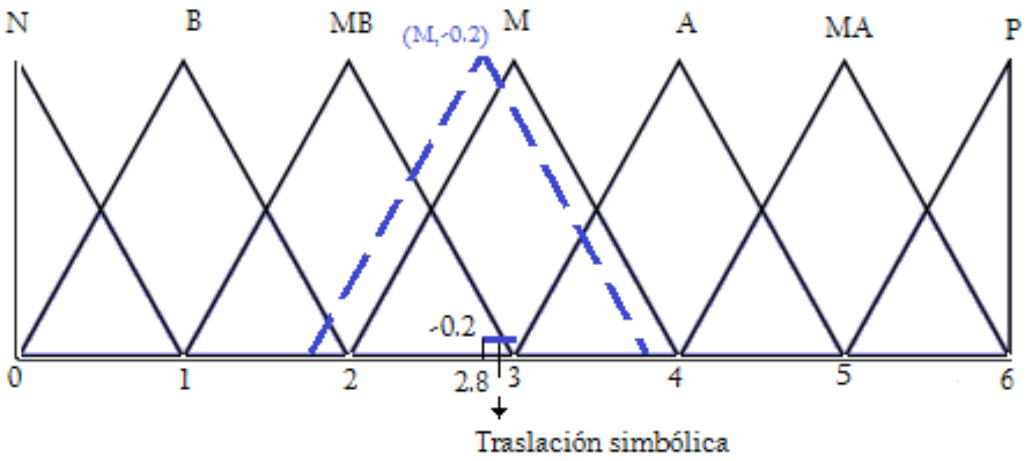

Figura 2.11 Ejemplo de una operación de Traslación Simbólica.

2.5.4.2. Modelo computacional de 2-Tupla

Junto a este modelo de representación de información lingüística, Herrera y Martínez definieron un modelo computacional lingüístico basado en las funciones de transformación Δ y Δ^{-1}. Para realizar transformaciones entre valores numéricos y 2-Tupla con objeto de facilitar los procesos de computación con palabras

La agregación consiste en agrupar varios elementos y transformarlo en uno solo, este elemento será el representante del conjunto de elementos. Por lo tanto, los operadores de agregación agruparán las preferencias sobre las alternativas. Cada una de estas alternativas tendrá una valoración hecha por cada experto, al usar el operador de agregación con todas las valoraciones se tendrá como resultado una única valoración por alternativa.

El uso de operadores de agregación en la toma de decisión multicriterio, da lugar a que este tipo de problemas sean más simples cuando debemos enfrentarnos a ellos; el gran problema al cual nos enfrentamos cuando hacemos uso de los operadores de agregación es la pérdida de información y la calidad de la solución, sobre todo cuando tenemos problemas bajo incertidumbre.

A continuación se revisan algunos operadores de agregación para el modelo 2-Tuplas [37]; los cuales se implementarán posteriormente.

Media aritmética: Sea un conjunto de valores numéricos $X = \{x_0,...x_n\}$, la media aritmética $\bar{x}$ se obtiene dividiendo la suma de todos los valores por su cardinalidad.

$$\overline{x}(x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

(2.4)

Este operador es el más sencillo de entender y además en teoría es el operador que ofrece un equilibrio o conjunto de valores centrales, aunque puede ser engañoso en ciertos casos que nos encontramos con valores anómalos o datos extremos.

Sea $x = \{(r_1,\alpha_1),..., (r_m,\alpha_m)\}$ un conjunto de 2-Tuplas con valores lingüísticos, un operador de media es definido como:

$$\overline{x}^e((r_1, \alpha_1), \ldots, (r_m, \alpha_m)) = \Delta(\frac{1}{n}\sum_{i=1}^{n} \Delta^{-1}(r_i, \alpha_i)) = \Delta(\frac{1}{n}\sum_{i=1}^{n} \beta_i)$$

(2.5)

Media aritmética ponderada: Permite que diferentes valores de x_i tengan diferente importancia, esto es debido a que cada valor de x_i se le asocia un peso w_i, el cual indica la relevancia de un valor sobre otro.

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

(2.6)

Sea $x = \{(r_1,\alpha_1),..., (r_m,\alpha_m)\}$ un conjunto de 2-Tuplas con valores lingüísticos y $w = \{(w_1,\alpha_1),..., (w_m,\alpha_m)\}$ el vector de pesos asociado, un operador de media ponderada es definido como:

$$\overline{x}^e = \Delta(\frac{\sum_{i=1}^{m} \Delta^{-1}(r_i,\alpha_i) \cdot w_i}{\sum_{i=1}^{m} w_i}) = \Delta(\frac{\sum_{i=1}^{m} \beta_i \cdot w_i}{\sum_{i=1}^{m} w_i})$$

(2.7)

OWA: Yager [49] fue el que introdujo este operador, es un operador de agregación con peso, en el cuál, los pesos no están asociados a un valor predeterminado sino que están asociados a una posición determinada.

$$F(a_1, ..., a_n) = \sum_{j=1}^{n} w_j b_j,$$

$$donde\ b_j\ es\ el\ j\text{-}esimo\ mayor\ valor\ del\ conjunto\ A.$$

(2.8)

Sea $x = \{(r_1,\alpha_1),..., (r_m,\alpha_m)\}$ un conjunto de 2-Tuplas con valores lingüísticos y $w = \{(w_1,\alpha_1),..., (w_m,\alpha_m)\}$ el vector de pesos asociado que satisface que:

- $w_i \in [0,1]$.
- $\sum w_i = 1$.
-

Un operador OWA es definido como:

$$F^e((r_1,\alpha_1),\ldots,(r_m,\alpha_m)) = \Delta(\sum_{j=1}^{m} w_j \cdot \beta_j^*),$$

$$siendo\ \beta_j^*\ el\ j\text{-}esimo\ mayor\ valor\ de\ los\ \Delta^{-1}((r_i,\alpha_i)).$$

(2.9)

Media geométrica: muestra un valor típico de un conjunto de números utilizando el producto de sus valores. A menudo se utiliza para comparar los diferentes elementos que tienen diferentes rangos numéricos.

$$\overline{x}(x_1, ..., x_n) = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

(2.10)

Sea $x = \{(s_1,\alpha_1),..., (s_m,\alpha_m)\}$ un conjunto de 2-Tuplas con valores lingüísticos, un operador de media geométrica es definido como:

$$TGA((s_1, \alpha_1), \ldots, (s_n, \alpha_n)) = \Delta((\prod_{i=1}^{n} \beta_i)^{1/n}), \quad \beta_i = \Delta^{-1}(s_i, \alpha_i)$$

(2.11)

Media geométrica ponderada: Tal y como ocurre en la media aritmética ponderada, los diferentes valores de x_i tienen diferente importancia, esto es debido a que cada valor de x_i se le asocia un peso w_i, el cual indica la relevancia de un valor sobre otro.

$$\overline{x}(x_1, ..., x_n) = \left(\prod_{i=1}^{n} x_i^{\alpha_i}\right)^{\frac{1}{\sum_i \alpha_i}}$$

Donde las α_i son los «pesos».

(2.12)

Sea $x = \{(s_1,\alpha_1),..., (s_n,\alpha_n)\}$ un conjunto de 2-Tuplas con valores lingüísticos, n la dimensión y w_i vector de pesos, un operador de media ponderada es definido como:

$$TWGA((s_1, \alpha_1), \ldots, (s_n, \alpha_n)) = \Delta(\prod_{i=1}^{n} \beta_i^{w_i}), \quad \beta_i = \Delta^{-1}(s_i, \alpha_i)$$

(2.13)

Media armónica: es el recíproco de la media aritmética, y se utiliza para calcular el promedio de los valores que varían en el tiempo.

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

(2.14)

Wei define varias extensiones de este operador para hacer frente a los valores lingüísticos 2-Tupla [50].

Sea $x = \{(s_1,\alpha_1),..., (s_m,\alpha_m)\}$ un conjunto de 2-Tuplas con valores lingüísticos, un operador de media armónica es definido como:

$$THA((s_1,\alpha_1),\ldots,(s_n,\alpha_n)) = \Delta\left(n/\sum_{j=1}^{n}\frac{1}{\beta_i}\right),\ \beta_i = \Delta^{-1}(s_i,\alpha_i)$$

(2.15)

Media armónica ponderada: Este operador también admite tener más importancia de unos valores de x_i que otros mediante la asociación de pesos w_i.

$$H = \frac{n}{\sum_{i=1}^{n}\frac{w_i}{x_i}}$$

(2.16)

Sea $x = \{(s_1,\alpha_1),..., (s_n,\alpha_n)\}$ un conjunto de 2-Tuplas con valores lingüísticos, n la dimensión y w_i vector de pesos, un operador de media ponderada es definido como:

$$TWHA((s_1,\alpha_1),\ldots,(s_n,\alpha_n)) = \Delta\left(1/\sum_{j=1}^{n}\frac{w_i}{\beta_i}\right),\ \beta_i = \Delta^{-1}(s_i,\alpha_i)$$

(2.17)

Capítulo 3: FLINTSTONES

3.1. Sistema de soporte a la decisión

En el anterior capítulo se da una visión general de lo que es la toma de decisión. Este apartado se centra en qué consisten los sistemas de soporte a la decisión, ya que Flintstones es el sistema de soporte a la decisión que se va a utilizar en este proyecto.

Los sistemas de soporte a la decisión [86, 87] son herramientas de ayuda para el proceso de toma de decisión, este apoyo se presta en todo el proceso desde que reunimos la información hasta que se llega a tomar la decisión final; siempre tenemos que tener en cuenta que este tipo de herramientas sirven de apoyo, puesto que la decisión final de tomar una decisión u otra la llevará a cabo una persona en última instancia.

Los SSD son un conjunto de herramientas y programas que en un ambiente de incertidumbre nos posibilita obtener la información necesaria para el proceso de toma de decisión. Ayudan a la toma de decisiones gracias a que combinan datos, modelos analíticos sofisticados y software de fácil manejo en un solo sistema cuyo potencial es alto y que puede dar soporte tanto a la toma de decisiones semiestructuradas como no estructuradas.

La principal finalidad de este tipo de sistema de soporte a la decisión es prestar apoyo a la toma de decisiones mediante la generación y evaluación de diferentes alternativas o escenarios de decisión, todo esto utilizando modelos y herramientas computacionales.

Gracias a herramientas software de este tipo las personas encargadas de tomar decisiones lo pueden hacer de una forma mucho más acertada, ya que poseen información mucho más amplia y precisa, así como diferentes escenarios.

3.2. Eclipse RCP

A continuación se explica que es Eclipse RCP, ya que es la tecnológica en la que se ha desarrollado Flintstones y que hace que su funcionalidad sea fácilmente ampliable, como se explicará más adelante.

El mínimo conjunto de plug-ins necesarios para construir una aplicación rica de escritorio se conoce en su conjunto como Eclipse RCP (Rich Client Platform). Por lo tanto una aplicación Ecplise RCP es una aplicación de escritorio desarrollada bajo tecnologías Eclipse [93].

Una aplicación Eclipse consta de varios componentes Eclipse. Un componente software en Eclipse se denomina plug-in. La plataforma Eclipse permite al desarrollador ampliar las aplicaciones Eclipse fácilmente mediante IDEs de Eclipse con funcionalidades adicionales a través de plug-ins.

Las aplicaciones Eclipse utilizan una especificación llamada OSGI. Un componente software OSGI se llama paquete. Un paquete OSGI también siempre será un plug-in Eclipse. Ambos términos se pueden utilizar indistintamente [92].

OSGI es una especificación que describe un enfoque modular para desarrollar aplicaciones Java basadas en componentes. El modelo de programación de OSGI permite

definir componentes software dinámico, ejemplo OSGI services. Equinox es una implementación de la especificación OSGI y es usada por la plataforma Eclipse como entorno de ejecución. Este entorno de ejecución proporciona las APIs necesarias y un espacio de trabajo para ejecutar una aplicación modular de Eclipse.

Creando un nuevo plug-in podríamos añadir nueva funcionalidad a nuestra aplicación por ejemplo dotándolo de una barra de herramientas para deshacer los últimos cambios realizados en la aplicación, estos plug-in no solo tienen porque añadir nuevas funcionalidades sino que también pueden extender funcionalidades que ya teníamos, como por ejemplo partiendo de un plug-in que realiza gráficos de barras podemos crear otro plug-in para hacer que tengamos más tipos de gráficos para representar nuestros datos y así conseguiríamos extender la funcionalidad de este plug-in.

La plataforma Eclipse forma parte de la IDE de más éxito de Java. Por lo tanto es muy estable y se utiliza ampliamente. Utiliza una interfaz de usuario nativa que es rápida y fiable. Tiene un enfoque modular fuerte basado en el sistema de módulos estándar de Java (OSGI) que permite a los desarrolladores diseñar sistemas basados en componentes.

La plataforma Eclipse también posee una gran comunidad de personas que proporcionan apoyo, documentación y ampliaciones de la infraestructura Eclipse. El aprovechamiento de este ecosistema permite encontrar recursos e información. De ahí su éxito y su popularidad.

La plataforma Eclipse ha sido diseñada para servir como una plataforma de herramientas abierta, lo que significa que podría ser usada para crear cualquier aplicación de cliente.

Las aplicaciones Eclipse RCP se benefician de la interfaz de usuario existente y el marco interno, y pueden reutilizar los plug-ins y características existentes en el entorno Eclipse.

3.3. Flintstones

Hay una diversidad de modelos lingüísticos que podemos aplicar a la toma de decisiones, aunque no todos cumplen los requisitos de la computación con palabras, el modelo lingüístico 2-Tuplas sin embargo si cumple con ese requisito.

La falta de herramientas de software para la toma de decisiones lingüística dentro del paradigma de computación con palabras ha provocado el desarrollo de un software reutilizable y fácilmente ampliable, gracias a su tipo de programación y estructura. A este software se le nombra como Flintstones acrónimo de Fuzzy LINguistic deciSion TOols eNhancEment Suite, que significa en español "Herramienta para la toma de decisiones lingüística difusa".

Flintstones es una herramienta software que implementa el modelo lingüístico 2-Tuplas para resolver problemas de toma de decisión con un contexto de incertidumbre y sus extensiones para hacer frente a marcos complejos como los marcos lingüísticos multigranulares, marcos heterogéneos y marcos lingüísticos no balanceados.

3.3.1. Arquitectura y tecnología

El software Flintstones no sólo tiene por objeto facilitar el proceso de toma de decisiones lingüísticas utilizando el modelo lingüístico 2-tupla y sus extensiones, sino también a facilitar la reutilización y la ampliación de sus componentes actuales, sin tener que volver hacer una nueva versión del programa. Además se puede añadir más funcionalidades o extender las que ya había mediante extensiones sin que estás afecten al núcleo del programa.

Por estos motivos se decidió utilizar para el desarrollo de la herramienta Flintstones la tecnología de Eclipse RCP.

Flintstones incluye más de 100 componentes, que se pueden agrupar en nueve tipos de componentes básicos:

- Núcleo (Core): Proporciona estructuras y procedimientos con el objetivo de apoyar la resolución de los problemas de decisión lingüísticas. Estos algunos de estos componentes son los expertos, los criterios y alternativas, soporte para deshacer y rehacer acciones, y funcionalidades para el almacenamiento y exportación de datos.
- Interfaz gráfica de usuario (GUI): Hace que el usuario pueda interactuar con el paquete de software de herramientas.
- Fases de resolución (Resolution Phases): Fases del proceso de toma de decisión.
- Los esquemas de resolución (Resolution Schemes): conjunto ordenado de las fases de resolución que llevan a cabo el proceso de toma de decisión.
- Dominios (Domains): El conjunto de dominios en los que los expertos pueden proporcionar sus evaluaciones.
- Las valoraciones (Valuations): Soporte para tipos específicos de evaluaciones lingüística, numérica, o intervalos valorados.
- Fases Método (Method Phases): fases específicas de los métodos de toma de decisión del modelo 2-tupla.
- Métodos (Method): Este módulo se desarrolla el modelo computacional lingüístico de 2-tupla y sus extensiones para resolver los problemas de toma de decisiones con los contextos lingüísticos y complejos.
- Los operadores (Operators): los operadores de agregación que se pueden utilizar para agrupar la información involucrada en los problemas de decisión.

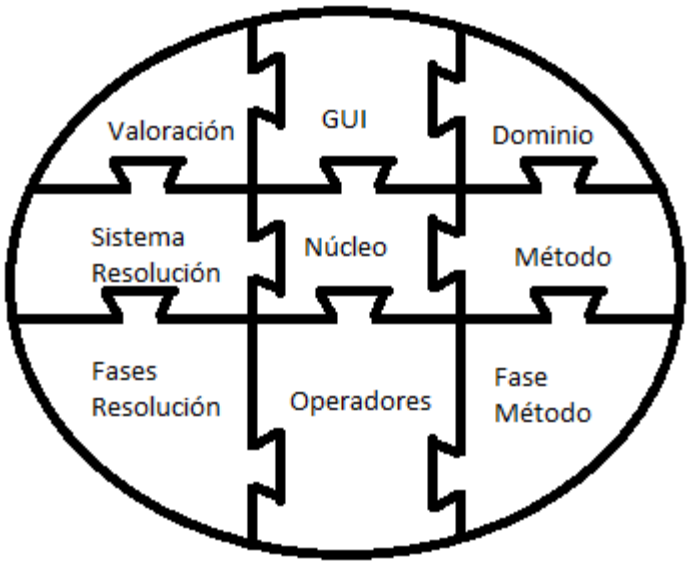

Figura 3.1 Tipos de componentes en Flintstones

La arquitectura basada en componentes ofrece varias ventajas sobre la base de su arquitectura y tecnologías:

- Flintstones ha sido desarrollado con Eclipse RCP basado en Java. Por lo tanto, es una suite de herramientas multiplataforma que puede utilizarse en cualquier máquina con la máquina virtual de Java (JVM), independientemente del sistema operativo en el que esté funcionando la máquina.
- La suite de software está dividida en componentes independientes, por lo tanto, es posible actualizar cualquier componente o añadirlo sin tener que hacer cambios en los componentes que no estén involucrados.
- La estructura de Flintstones está preparada para incluir nuevas funcionalidades o elementos tales como nuevos operadores de agregación, nuevos procesos de resolución, o nueva modelización de preferencias, esto se hace a través de los puntos de extensión, que definen interfaces para otros plug-ins.

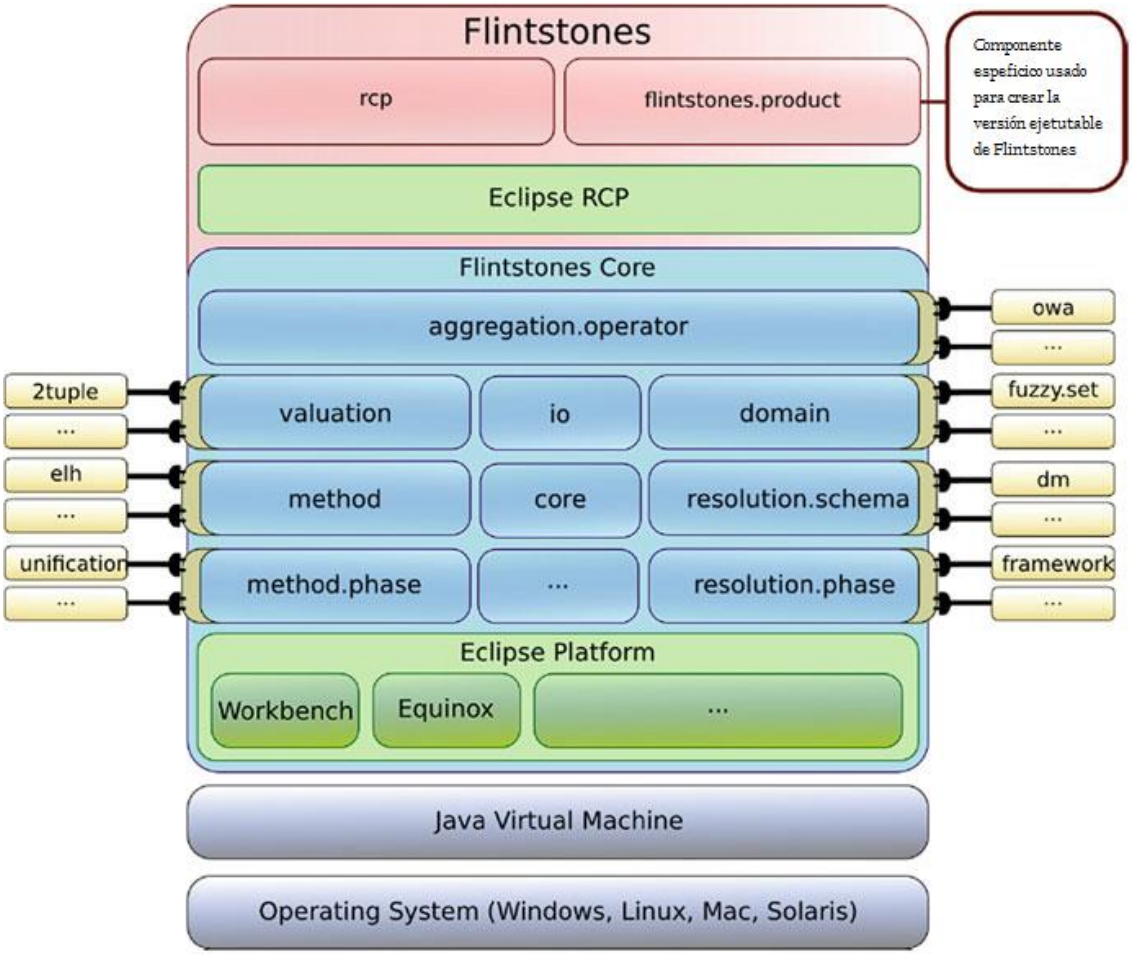

Figura 3.2 Arquitectura en Flintstones

En este contexto, una vez seleccionado el método de resolución y los operadores de agregación que deseamos para la resolución del problema. Flintstones nos permite resolver y analizar los resultados obtenidos.

En Flintstones hay diferentes plug-ins que componen la aplicación. En esta versión la parte gráfica está totalmente separada de la parte del modelo. Todos aquellos plug-ins que tienen en su nombre la terminación ".ui" son plug-ins de interfaz. Es decir que estos plug-ins se encargarán de la visualización de diferentes elementos de la aplicación, por ejemplo los dominios, expertos, alternativas, criterios, tablas, vistas, iconos etc.

Algunos de estos plug-ins son:

1. De.kupzog.ktable.KTable: plug-in que contienen los JARs de la librería externa KTable. Se usa para representar algunas tablas dentro de Flintstones.
2. org.jfree.chart.JFreeChart: plug-in que contiene los JARs de la librería externa JFreechart. Se usa para representar los gráficos dentro de Flintstones.
3. sinbad2.aggregationoperator: plug-in que define el concepto de operador de agregación.
4. sinbad2.aggregationoperator.max: plug-in que define el operador máximo.

5. sinbad2.aggregationoperator.maxMin: plug-in que define el operador máximo-mínimo.
6. sinbad2.aggregationoperator.median: plug-in que define el operador de mediana.
7. sinbad2.aggregationoperator.min: plug-in que define el operador mínimo.
8. sinbad2.aggregationoperator.owa: plug-in que define el operador OWA.
9. sinbad2.aggregationoperator.arithmeticMean: plug-in que define el operador de media aritmética.
10. sinbad2.aggregationoperator.geometricMean: plug-in que define el operador de media geométrica.
11. sinbad2.aggregationoperator.harmonicMean: plug-in que define el operador de media armónica.
12. sinbad2.aggregationoperator.weightedmean: plug-in que define el operador de media aritmética ponderada.
13. sinbad2.aggregationoperator.weightedmeanmodified: plug-in que define el operador de media aritmética ponderada modificada.
14. sinbad2.aggregationoperator.weightedgeometric: plug-in que define el operador de media geométrica ponderada.
15. sinbad2.aggregationoperator.weightedharmonic: plug-in que define el operador de media armónica ponderada.
16. sinbad2.core: plug-in que se encarga de controlar acciones del núcleo de Flintstones. Por ejemplo el rehacer/deshacer, gestión de ficheros tipo Flintstones etc.
17. sinbad2.core.ui: plug-in que se encarga de controlar acciones de interfaz del núcleo de Flintstones. Por ejemplo cambio de barra de título etc.
18. sinbad2.domain: plug-in que define el concepto de dominio.
19. sinbad2.domain.ui: plug-in que define la parte gráfica de un dominio.
20. sinbad2.domain.linguistic.fuzzyset: plug-in que define un dominio lingüístico.
21. sinbad2.domain. linguistic.fuzzyset.ui: plug-in que define la parte gráfica de un dominio lingüístico.
22. sinbad2.domain.linguistic.unbalanced: plug-in que define un dominio no balanceado.
23. sinbad2.domain. linguistic. unbalanced.ui: plug-in que define la parte gráfica de un dominio no balanceado.
24. sinbad2.domain.numeric.integer: plug-in que define un dominio numérico entero.
25. sinbad2.domain. numeric.integer.ui: plug-in que define la parte gráfica de un dominio numérico entero.
26. sinbad2.domain.numeric.real: plug-in que define un dominio numérico real.
27. sinbad2.domain. numeric.real.ui: plug-in que define la parte gráfica de un dominio numérico real.
28. sinbad2.domain.valuations: plug-in que se encarga de manejar la relación entre dominios y valoraciones.
29. sinbad2.element: plug-in que define el concepto de elemento dentro del problema. Expertos, alternativas, criterios.
30. sinbad2.element.ui: plug-in que define la parte gráfica de un elemento del problema.

31. sinbad2.method.linguistic: plug-in que define el concepto de modelo lingüístico.
32. sinbad2.method.x: plug-in que definen distintos modelos lingüísticos con los que resolver un problema de toma de decisión.
33. sinbad2.phasemethod: plug-in que define el concepto de fase de un modelo lingüístico.
34. sinbad2.phasemethod.ui: plug-in que define la parte gráfica de una fase de método.
35. sinbad2.phasemethod.x: plug-in que definen las diferentes fases que componen un modelo lingüístico concreto.
36. sinbad2.rcp: plug-in principal de la aplicación RCP donde se encuentra el .product y desde donde se ejecuta la aplicación y se incluyen los plug-ins necesarios para ejecutarla.
37. sinbad2.resolutionphase: plug-in que define el concepto de fase de un esquema de resolución.
38. sinbad2.resolutionphase.ui: plug-in que define la parte gráfica de una fase de un esquema de resolución.
39. sinbad2.resolutionphase.x: plug-in que definen las diferentes fases del esquema de resolución que se emplea en Flintstones. Las fases son: Framework (donde se define el problema), FrameworkStructuring (donde se asignan los dominios), Gathering (donde se recogen las opiniones de los expertos), Rating (donde se resuelve el problema en función del modelo lingüístico elegido) y SensitivityAnalysis (donde se hace un análisis más profundo de la solución del problema).
40. sinbad2.resolutionscheme: plug-in que define el concepto de esquema de resolución.
41. sinbad2.resolutionscheme.ui: plug-in que define la parte gráfica de un esquema de resolución.
42. sinbad2.resolutionscheme.dm: plug-in que define un esquema de resolución para un problema de toma de decisión.
43. sinbad2.resolutionscheme.dm.ui: plug-in que define la parte gráfica del esquema de resolución para problemas de toma de decisión.
44. sinbad2.target.rcp: plug-in en el que se encapsulan los diferentes plug-ins que va a emplear Eclipse para ejecutar la aplicación (no se ejecutan los plug-ins por defecto).
45. sinbad2.valuation: plug-in que define el concepto de valoración/evaluación de un experto.
46. sinbad2.valuation.ui: plug-in que define la parte gráfica de una valoración/evaluación.
47. sinbad2.valuation.x: plug-in que definirán las diferentes valoraciones soportadas por Flintstones. Pueden ser: valoraciones hesitant, enteras, enteras intervalares, reales, reales intervalares, lingüísticas, unificadas y 2-Tuple.
48. sinbad2.valuationSet: plug-in que se encarga de gestionar el conjunto de valoraciones.
49. sinbad2.valuationSet.ui: plug-in que se encarga de la parte gráfica del conjunto de valoraciones.

3.3.2. Estructura de la interfaz

La organización de la estructura en Flintstones al igual que ocurre en la arquitectura se divide en fases como el proceso de toma de decisión

Al igual que ocurre con las fases de resolución de los problemas de toma de decisión, Flintstones se compone de cinco bloques funcionales independientes que son los siguientes:

- Marco de evaluación (Framework).
- Estructura del marco de evaluación (Framework Structuring).
- Recogida de información (Gathering).
- Valoración de alternativas (Rating).
- Análisis (Sensitivity Analysis).

3.3.2.1. Marco de evaluación

En este bloque se establece la estructura del problema donde se incluirá todos los elementos que tienen relevancia para el problema como son:

- Expertos: La aplicación permite definir uno o múltiples expertos, así como la agrupación por subgrupos.
- Criterios: La aplicación permite definir los criterios involucrados en la definición del problema, así como, árboles de criterios de hasta tres niveles. Criterios de coste y de beneficio.
- Alternativas: La aplicación permite introducir las diferentes alternativas propuestas por cada uno de los expertos.
- Dominios: La aplicación permite la definición de los dominios de expresión para valorar las distintas alternativas.

3.3.2.2. Estructura del marco de evaluación

En este bloque es donde se asignan los dominios de expresión que se utilizan para valorar las diferentes alternativas. Los dominios permitidos son: numérico, intervalar, lingüístico, hesistant y no balanceados.

Asignación de dominios de expresión permite o tiene un área para la asignación de los distintos dominios de expresión a cada experto, criterio y alternativa.

3.3.2.3. Recogida de información

En este bloque básicamente se recoge la información de las diferentes valoraciones, estas valoraciones son recogidas a través de la aplicación que permite un filtrado por criterios, expertos o alternativas para hacer así más fácil la recolección de datos o información.

Una vez recogida toda la información ya podemos pasar al siguiente bloque para así poder valorar todas las valoraciones recogidas en este bloque.

3.3.2.4. Valoración de alternativas

Para la valoración de alternativas se selecciona el método para la resolución del problema, aquí es donde aparecerá todos los métodos disponibles como por ejemplo el modelo 2-tuplas entre otros.

Una vez seleccionado el método, la aplicación le guiara mediante pestañas para la elección del operador de agregación deseado, y así mostrar los resultados parciales y finales.

Hay que tener en cuenta que dependiendo del método seleccionado tendremos unas pestañas u otras.

Por ejemplo en el caso que la elección del método de resolución haya sido el modelo 2-Tuplas, los operadores de agregación que podremos escoger son:

- Media aritmética
- Media armónica
- Media geométrica
- Media aritmética ponderada
- Media armónica ponderada
- Media geométrica ponderada

Tenemos que tener en cuenta que en caso de elegir cualquier operador de agregación ponderado, deberemos indicar el peso que se le asigna a cada experto o criterio, para ello la aplicación nos guiara mediante una ventana para que a cada alternativa se le pueda asignar su correspondiente peso o importancia.

3.3.2.5. Análisis sensitivo

Por último tenemos el bloque análisis, aquí es donde también influye el método de resolución escogido, puesto que algunos métodos ofrecen mayor detalle a la hora de exponer los resultados obtenidos, algunos métodos disponen de gráficos y tablas para facilitar así el análisis de los resultados.

Aquí también es donde tenemos que tener en cuenta la consistencia de nuestra solución, la cual está dentro de un rating de soluciones, y para comprobar la consistencia de la solución obtenida debemos fijarnos en el rating que esta solución tiene respecto de la siguiente, cuanto mayor sea la diferencia entre la primera solución y las demás mayor será el grado de consistencia de la solución, en cambio sí con un cambio leve la solución primera cambia entonces la solución no tiene la suficiente consistencia debido a que algún paso no haya sido correcto o bien el operador usado no sea la suficientemente bueno.

3.4. Puntos de extensión en Flintstones

Los puntos de extensión definen interfaces para otros plug-ins para contribuir a su funcionalidad, utilizando los puntos de extensión podemos extender o modificar la funcionalidad de Flintstones. Un punto de extensión define un contrato al que deben ajustarse las extensiones. Los componentes a los que se les desea ampliar o personalizar su funcionalidad deben ajustarse a este contrato.

En Flintstones estos puntos de extensión hacen posible que la extensión de nueva funcionalidad se realice de forma simple y sin tener que modificar otros componentes ya existentes, aunque a veces puede ocurrir que al introducir alguna nueva funcionalidad, ésta entre en conflicto con algún componente, y en este caso sí que sea necesario modificar el antiguo componente o eliminarlo.

Por otro lado necesitaremos estudiar bien la nueva funcionalidad que queremos introducir en Flintstones, porque como ya hemos visto debemos pensar no solo en la actual funcionalidad sino en las futuras funcionalidades; para así no tener código redundante.

Precisamente cuando estamos haciendo uso de código redundante, es cuando nos debemos plantear la idea de usar un nuevo punto de extensión para así reducir la redundancia, y que nuestro programa sea más fácil de manejar y administrar por parte de los desarrolladores.

Otra ventaja de utilizar los puntos de extensión en Flintstones es la detección de errores, debido a que la funcionalidad de Flintstones está compuesta de distintos componentes, cuando alguno de estos falle, solo debemos revisarlo siendo los demás componentes transparentes.

Flintstones hace un uso intensivo de estos conceptos, siendo los siguientes siete componentes los más importantes o fundamentales para resolución de un problema de toma de decisión:

- Fases de resolución
- Los esquemas de resolución
- Dominios
- Las valoraciones
- Las fases del método
- Métodos
- Operador de agregación

Cada una de estas extensiones se define en componentes separados. Cada componente que define los puntos de extensión gestiona el ciclo de vida de todas las extensiones que implementan este punto de extensión. La aplicación interna de los puntos de extensión de estos componentes es similar al esquema que se muestra en la Figura 3.3, pero adaptado a las necesidades específicas de la funcionalidad extendida.

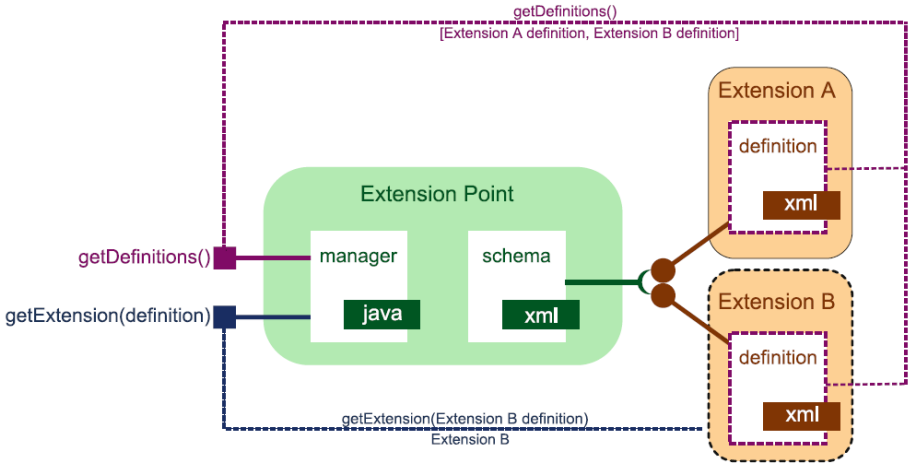

Figura 3.3 Esquema de un punto de extensión

Los puntos de extensión que podemos encontrarnos en Fintstones son:

- Flintstones.aggregationoperator.exsd: Punto de extensión que define un operador de agregación. Un operador de agregación puede ser ponderado o no (unweighted o weighted) y se definen los tipos que soporta (numérico, lingüístico etc.).
- Flintstones.domain.exsd: Punto de extensión que define un dominio. Un dominio puede ser de dos tipos, lingüístico o numérico.
- Flintstones.domain.ui.exsd: Punto de extensión que define la parte gráfica de un dominio. Estará formado por un chart (representación gráfica del dominio) y un icono que lo represente.
- Flintstones.domain.ui.modify.dialog: Punto de extensión que define una ventana (dialog) para la modificación de un dominio.
- Flintstones.domain.ui.new.dialog: Punto de extensión que define una ventana para la creación de un dominio.
- Flintstones.method.exsd: Punto de extensión que define un modelo lingüístico. Un modelo lingüístico está formado por diferentes fases de modelo y además se le indica que tipo de agregación soporta.
- Flintstones.method.ui.exsd: Punto de extensión que define la parte gráfica de un modelo lingüístico. La parte gráfica de un modelo está formado por la parte gráfica de diferentes fases de modelo.
- Flintstones.phasemethod.exsd: Punto de extensión que define una fase de modelo lingüístico.
- Flintstones.phasemethod.ui.exsd: Punto de extensión que define la parte gráfica de una fase de modelo. Estará formado por un conjunto de vistas (steps) que definen los diferentes pasos que se llevan a cabo en una fase de modelo.
- Flintstones.resolutionphase.exsd: Punto de extensión que define una fase de esquema de resolución
- Flintstones.resolutionphase.ui.exsd: Punto de extensión que define la parte visual de una fase de esquema de resolución. Está formado por una "perspectiva" que será su visualización.

- Flintstones.resolutionscheme.exsd: Punto de extensión que define un esquema de resolución. Un esquema de resolución está formado por diferentes fases de esquema de resolución.
- Flintstones.resolutionscheme.ui.exsd: Punto de extensión que define la parte gráfica de un esquema de resolución. La parte gráfica de un esquema de resolución está formado por diferentes visualizaciones de fases de esquema de resolución.
- Flintstones.valuation.exsd: Punto de extensión que define una valoración. Una valoración puede tener diferentes tipos.
- Flintstones.valuation.ui.exsd: Punto de extensión que define la parte gráfica de una valoración. Una valoración tiene asignada una ventana para representar la valoración en un dominio, una ventana para modificar una valoración y presentarla en un dominio y un panel donde se representa dicha valoración.

3.5. Ventajas de utilizar Eclipse RCP

En este trabajo el proceso de ingeniería de software para la integración de los distintos operadores se hace de forma avanzada, ya que al utilizar Eclipse RCP para el desarrollo del mismo, no es necesario saber la estructura de Flintstones para la integración de nuevas funcionalidades, ya que se usan los puntos de extensión para dotar a Flintstones de nuevos operadores de agregación, solo se necesita conocer los componentes que intervienen para agregar los nuevos operadores.

Al utilizar una programación que se basa en componentes para agregar un nuevo operador solo hay que seguir el esquema que se indica en la Figura 3.4.

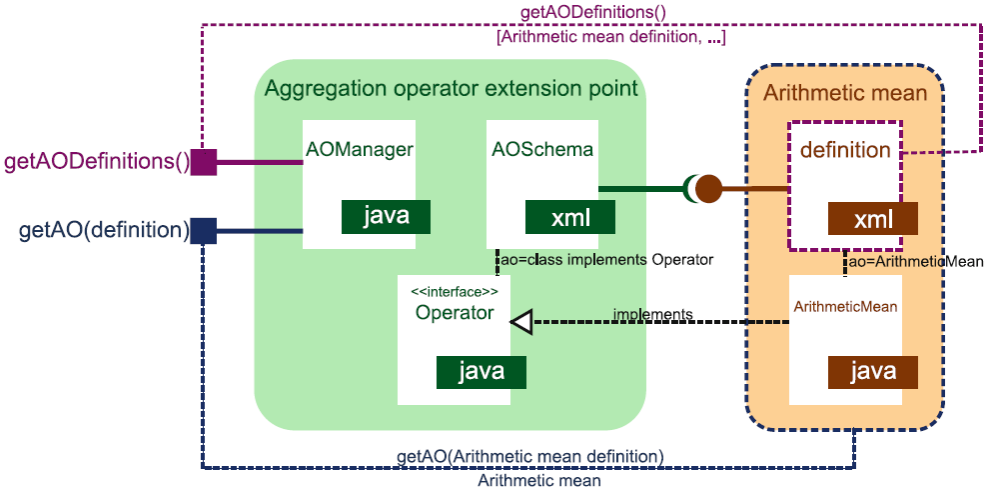

Figura 3.4 Punto de extensión de un operador de agregación

Capítulo 4: Proceso de ingeniería del software

4.1. Introducción

La Ingeniería del Software [88] es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar de forma económica software que sea fiable y que trabaje eficientemente en máquinas reales.

Para la realización del proyecto, se realiza el proceso de ingeniería del software. El proceso de ingeniería del software se divide en las fases como se muestra a continuación:

- **Especificación de requerimientos**: Se establece el propósito del proyecto, así como también las propiedades que tendrá.
- **Análisis del sistema**: Este proceso de análisis a través de una serie de requerimientos funcionales y no funcionales establece la naturaleza de la aplicación.
- **Diseño del sistema**: Se establecen los objetivos del proyecto y como llevarlos a cabo para conseguir alcanzarlos.
- **Implementación**: Convierte del modelo lógico del sistema a código fuente.
- **Pruebas**: Esta fase se comprueba que los anteriores pasos son correctos, y para ello se realizan pruebas para verificar que el proyecto ha conseguido el objetivo.

4.2. Especificación de requerimientos

En el primer capítulo ya se ha definido el propósito de este proyecto, por lo tanto lo que debemos tener en cuenta ahora son los requerimientos que necesitará para que la aplicación llegue a conseguir con dicho propósito.

4.2.1. Requerimientos funcionales

En un sistema software los requerimientos funcionales son aquellos que tiene el sistema en su funcionamiento habitual, es decir, las funciones básicas que todo sistema tiene.

Aunque este proyecto se centra en la creación de nuevos operadores de agregación mediante plug-ins, a continuación veremos los requisitos funcionales de todo el sistema para así entender mejor Flintstones.

Por lo tanto, nuestro sistema tendrá los siguientes requerimientos funcionales:

- Creación de un nuevo problema: Aquí es donde se establece el nuevo problema y contiene los siguientes elementos básicos:
    - Creación de un nuevo experto (RF01): Se crea un nuevo experto para el problema.
    - Creación de una nueva alternativa (RF02): Se crea una nueva alternativa para el problema.

  - Creación de un nuevo criterio (RF03): Se crea un nuevo criterio para el problema.
  - Creación de un nuevo dominio de expresión (RF04): Se crea un nuevo dominio de expresión que podrá ser asociado a cualquier elemento del problema, de esta forma las valoraciones de ese elemento deberán ser dadas en dicho dominio.
  - Asignación de dominio de expresión (RF05): Se asocia un dominio de expresión a cualquiera de los elementos que ya han sido creados en el sistema.

- Valoración de las distintas alternativas (RF06): Se realiza una valoración de una alternativa asignada a un experto y sobre un criterio dentro del dominio de expresión que ese elemento tenga asignado.

- Selección del método de resolución (RF07): Se selecciona un método de resolución con el que se resolverá el problema.

- Selección de los distintos operadores de agregación: Existen los dos tipos de operadores que son:

  - Selección de operadores sin pesos (RF08): Se selecciona un operador de agregación para obtener los resultados.
  - Selección de operadores con pesos (RF09): Se selecciona un operador de agregación que requiere la asignación de pesos para obtener los resultados.

Los requerimientos funcionales principales son los dos últimos, ya que este proyecto su principal propósito es aumentar la funcionalidad de Flintstones mediante integración de operadores de agregación.

4.2.2. Requerimientos no funcionales

Los requerimientos no funcionales de un sistema software son aquellos que no están relacionados con la funcionalidad básica del sistema, sino que son factores externos al funcionamiento básico del sistema.

Puesto que el software que se cree se deberá integrar en una aplicación ya existente, esto da lugar a nuevos requerimientos no funcionales.

Por lo tanto los requerimientos no funcionales serían los siguientes:

- Tiempo de respuesta aceptable cuando el sistema es utilizado.
- La aplicación será multiplatafoma y no deberá perder la estructura visual ni su funcionalidad anterior.
- Requerimientos de la interfaz: tienen que ver con la usabilidad del sistema y se pueden resumir en:

  - Aprendizaje sencillo: El uso del sistema debe ser similar a otras herramientas del mismo tipo para que así el usuario pueda sacarle el máximo partido.

  - Flexibilidad: Se refiere al intercambio de información entre el usuario y el sistema, y este intercambio debe de poder realizarse de diferentes formas.
  - Robustez: Es la fiabilidad del sistema, o el grado en que el sistema es tolerante a fallos.

- Requerimientos relacionados con la integrar el software creado en la aplicación "Flinstones":

  - Agregar el nuevo software no deberá afectar a la estructura visual de la aplicación.
  - Los nuevos operadores de agregación deberán tener la misma estructura interna que los antiguos operadores de agregación y su utilización deberá ser igual o similar.

4.3. Análisis del sistema

En esta fase se analiza y crear un modelo, robusto y verificable. Para ello, deberemos definir los diferentes perfiles de usuario, los casos de uso según los requisitos vistos en el apartado 4.2.1.

4.3.1. Perfiles de usuario

El primer paso del análisis de un sistema software es definir los usuarios que intervendrán en él. De esta forma se podrá definir los requisitos de usabilidad que habrá que tener en cuenta en las diferentes etapas de desarrollo de software, en este paso solo se tendrá en cuenta los usuarios que interactúan con el sistema, ya que existiría un usuario administrador encargado del mantenimiento y la ampliación de la aplicación, este usuario en principio solo interactúa con la aplicación para hacer pruebas. Por lo tanto este administrador no se tendrá en cuenta.

En este sistema definiremos dos tipos de usuarios distintos, dependiendo de los conocimientos que tiene el usuario puede ser:

- **Experto**: Se trata de un usuario con muchos conocimientos sobre la toma de decisión, y además tiene experiencia con aplicaciones similares.
  - Debe tener un alto conocimiento sobre el problema de toma de decisiones, para proceder y obtener un resultado al problema propuesto.
  - Puede tener conocimientos informáticos para manejar correctamente la aplicación.
  - Este usuario también tendrá que tener una gran capacidad para decidir los métodos y operadores más apropiado dependiendo del problema propuesto.

- **Intermedio**: Se trata de un usuario con conocimientos sobre el problema de toma de decisión pero no alcanza a tener los conocimientos de un experto.
  - Debe conocer el proceso de toma de decisiones para poder seguir los pasos a resolver de un problema propuesto y conocer todos los elementos que intervienen en el mismo.

- Debe tener conocimientos informáticos para manejar correctamente la aplicación.

Si el usuario experto tiene conocimientos informáticos será el encargado de insertar las valoraciones sino deberá de relegar este papel al usuario intermedio que si tiene conocimientos informáticos.

El uso de un usuario intermedio y otro experto se hace debido a que aplicaciones de este tipo por lo general son aplicaciones no muy difíciles de manejar, en cambio sí son complicadas de entender debido a la dificultad de problemas con incertidumbre, ya que los conocimientos que se tienen que tener del problema son altos. Por lo tanto no tiene sentido tener usuarios noveles para este tipo de aplicaciones, puesto que no serían capaces de interpretar los resultados obtenidos. Además el usuario intermedio tiene conocimientos sobre el problema pero las valoraciones las realizaría un usuario experto y el usuario intermedio solo tendría que introducir las valoraciones hechas por los expertos.

4.3.2. Casos de uso

Los casos de uso definen un uso del sistema y cómo este interactúa con el usuario.

La realización de un caso de uso implica la identificación de un posible conjunto de clases, también implica un conocimiento de cómo podrían interactuar las clases entre sí para ofrecer la funcionalidad del caso de uso. El conjunto de clases es conocido como una colaboración.

Para obtener una visión básica de la colaboración:

- Hay que identificar los objetos implicados en la colaboración a partir del análisis del caso de uso asociado.
- Hay que identificar los vínculos entre los objetos de la colaboración.

Esta visión aún no indica cómo interactúan ni muestra cómo se relacionan con otras partes del modelo.

Los casos de uso explican un uso del sistema concreto y contiene los siguientes elementos:

- Nombre único del caso de uso.
- Actores participantes.
- Condiciones de entrada.
- Flujo de eventos (narrativa).
- Condiciones de salida.
- Requerimientos especiales.

El siguiente paso que debemos realizar es la identificación de los actores que serán los que intervengan en cada uno de los casos de uso.

Se le llama actor a toda entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. Los actores tienen un nombre y una

descripción, el nombre debe de ser único e inequívoco. En nuestro sistema tendremos únicamente el siguiente actor:

- Usuario: Engloba a todos los usuarios que interactúan con las diferentes funcionalidades que tiene el sistema. Aquí se engloban tanto los usuarios expertos como los intermedios.

Después de establecer los actores del sistema, el siguiente paso será definir los distintos casos de uso. Hay que tener en cuenta para la realización de los casos de uso los requerimientos funcionales.

El primer paso que deberemos hacer es el de establecer la funcionalidad básica del sistema y con ello podremos realizar los casos de uso. Para ello se emplea un diagrama frontera, es decir, un diagrama que describe completamente la funcionalidad del sistema como podemos observar en la Figura 4.1.

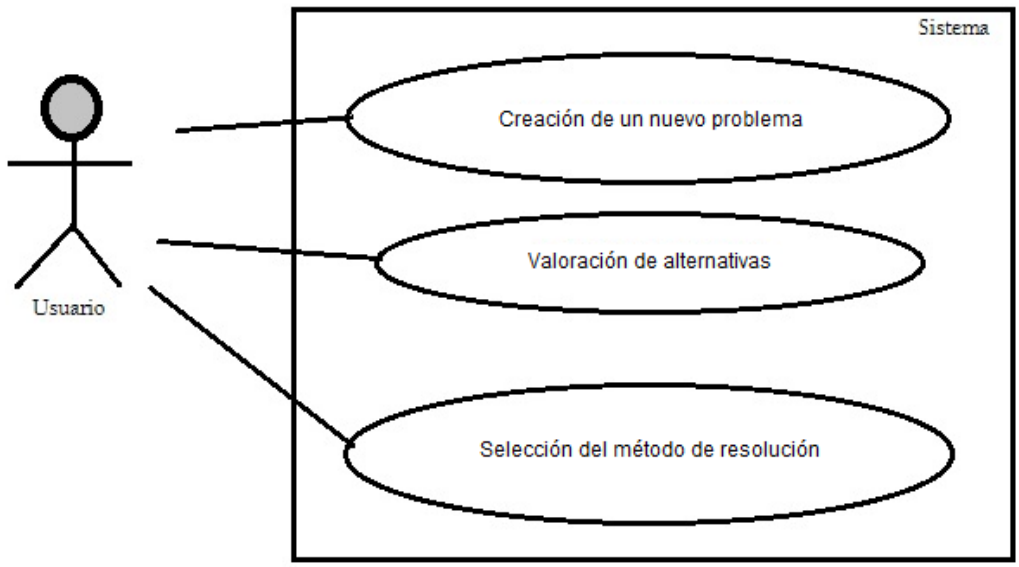

Figura 4.1 Diagrama frontera.

Ya que con un diagrama frontera lo que tenemos es una visión global del sistema y en ocasiones necesitamos mayor detalle que el expresado en un diagrama frontera.

En este caso, existen varios casos de uso que deben ser tratados con mayor detalle.

A continuación, en los siguientes apartados se verán con más detalle dichos casos de uso, que son:

1. Creación de un nuevo problema.

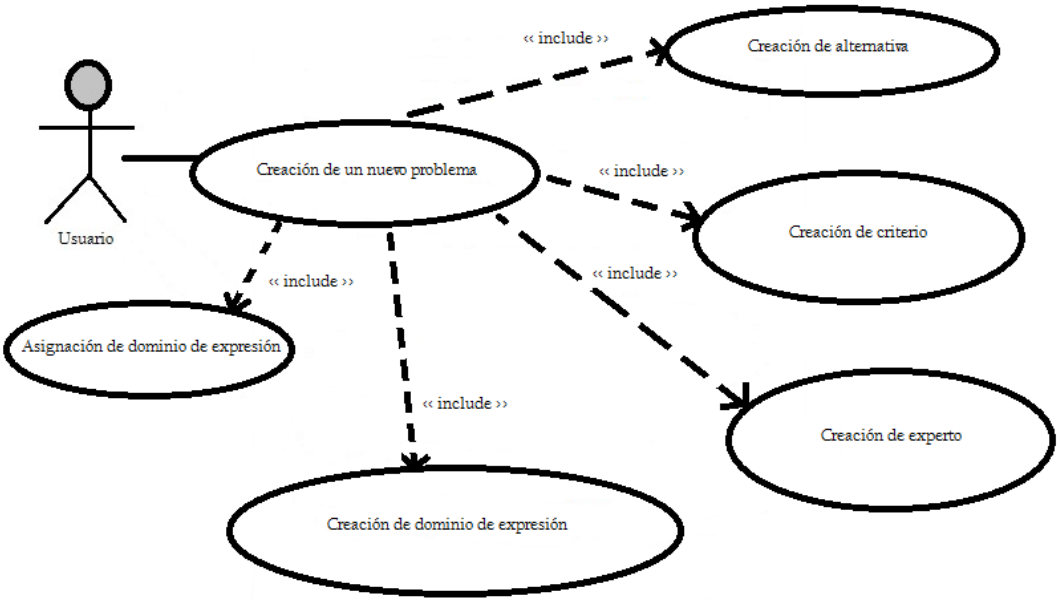

Figura 4.2 Diagrama de caso de uso "Creación de un nuevo problema".

2. Selección del método de resolución.

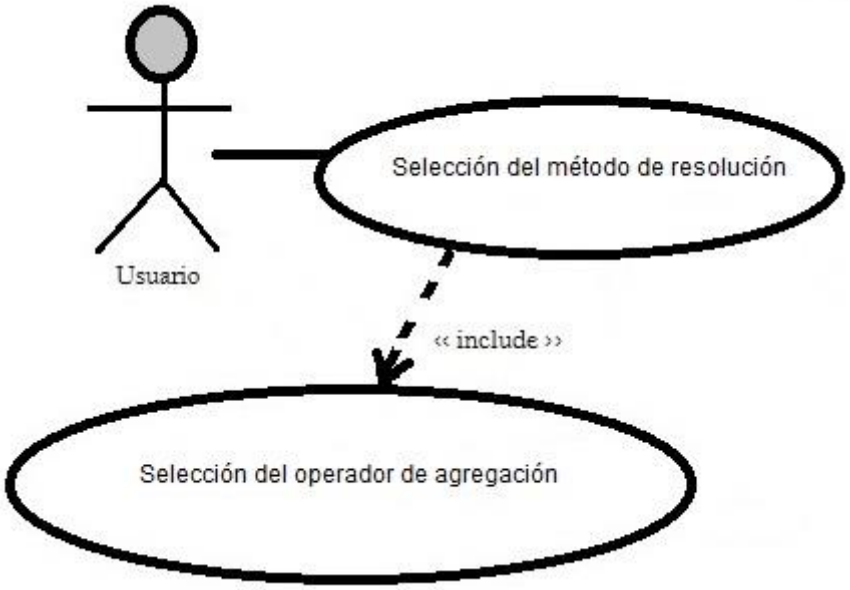

Figura 4.3 Diagrama de caso de uso "Selección del método de resolución".

3. Selección de operador de agregación.

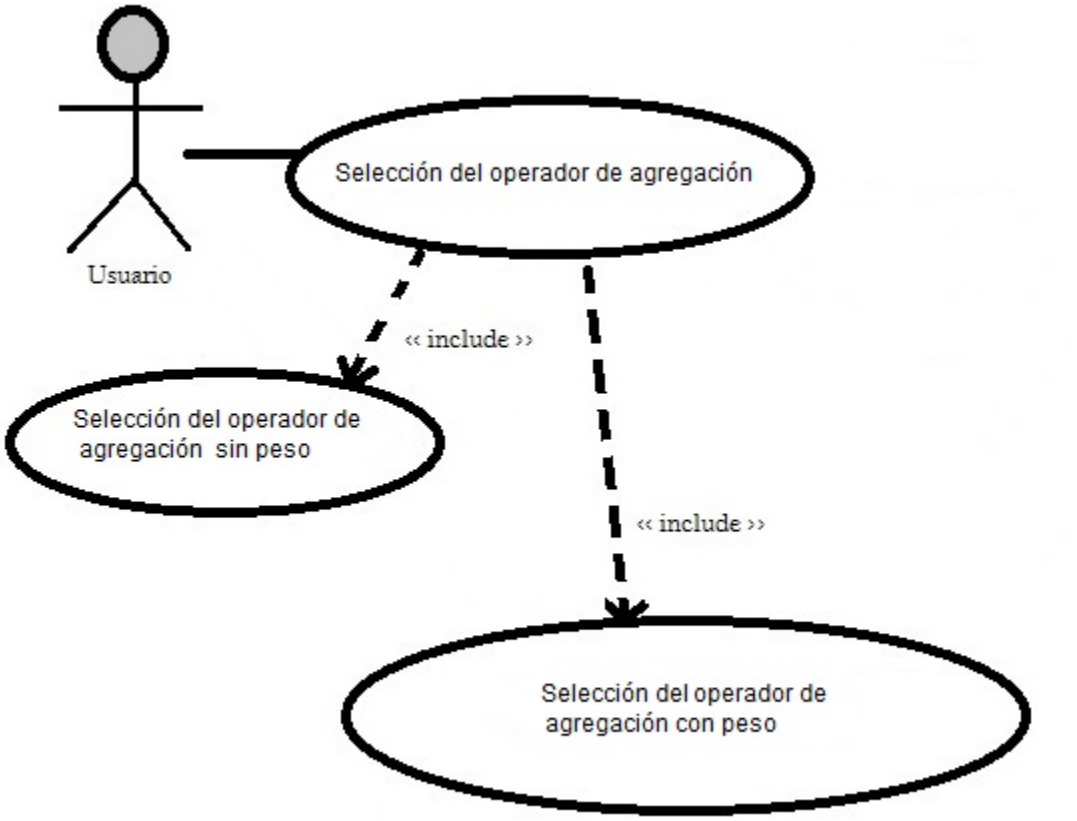

Figura 4.4 Diagrama de caso de uso "Selección del operador de agregación".

Como se puede ver en los diagramas anteriores podemos extraer muchos casos de uso diferentes. Pero no basta solo con mostrar los diagramas de casos de uso para entenderlos por lo tanto a continuación se describen detalladamente.

- **Caso de Uso 1: Creación de un nuevo problema**

Actores principales: Usuario
Condiciones de entrada: Ninguna.
Flujo de eventos:

1. El usuario solicita crear un nuevo problema.
2. Según el estado en el que se encuentre el sistema:

2.1 Si no hay datos de un problema anterior o el problema mostrado en ese momento no ha sido modificado, el sistema sólo vuelve a la página principal.

2.2 Si los datos que hay son de un problema anterior, y éstos han sido modificados:

2.2.1 El sistema muestra un mensaje para guardar o descartar los cambios.

2.2.2 El usuario solicita la opción que desee.

2.2.3 El sistema guarda los datos o los descarta y vuelve a la pantalla inicial de la aplicación.

Condiciones de salida: El sistema permite al usuario establecer el nuevo problema mediante la inserción de sus elementos.

Excepciones: Ninguna.

- **Caso de Uso 2: Creación de experto**

Actores principales: Usuario

Condiciones de entrada: El usuario actualmente está en la página inicial de la aplicación y desea comenzar con la inserción del experto o expertos del problema.

Flujo de eventos:

1. El usuario solicita la opción de crear un nuevo experto.
2. El sistema permite mediante una ventana la inserción de los datos del nuevo experto.
3. El usuario introduce los datos del experto y solicita la opción de añadirlo al problema.
4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el experto se ha añadido correctamente (E-1).

Condiciones de salida: El usuario ha introducido un nuevo experto en el sistema.

Excepciones:

E-1: Si al intentar introducir el nuevo experto, éste tiene el mismo nombre que otro experto creado previamente o bien éste experto ya se había creado anteriormente, el sistema muestra un error al usuario.

- **Caso de Uso 3: Creación de alternativa**

Actores principales: Usuario

Condiciones de entrada: El usuario actualmente está en la página inicial de la aplicación y desea hacer la inserción de las alternativas del problema.

Flujo de eventos:

1. El usuario solicita la opción de crear una nueva alternativa.
2. El sistema permite mediante una ventana la inserción de los datos de la nueva alternativa.
3. El usuario introduce los datos de la alternativa y solicita la opción de añadirlo al problema.
4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que la alternativa se ha añadido correctamente (E-1).

Condiciones de salida: El usuario ha introducido una nueva alternativa en el sistema.

Excepciones:

E-1: Si al intentar introducir la nueva alternativa, ésta tiene el mismo nombre que otra alternativa creada previamente o bien ésta alternativa ya se había creado, el sistema muestra un error al usuario.

- **Caso de Uso 4: Creación de criterio**

Actores principales: Usuario
Condiciones de entrada: El usuario actualmente está en la página inicial de la aplicación y desea hacer la inserción del criterio o criterios del problema.
Flujo de eventos:

1. El usuario solicita la opción de crear un nuevo criterio.
2. El sistema permite mediante una ventana la inserción de los datos del nuevo criterio.
3. El usuario introduce los datos del criterio y solicita la opción de añadirlo al problema.
4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el criterio se ha añadido correctamente (E-1).

Condiciones de salida: El usuario ha introducido un nuevo criterio en el sistema.
Excepciones:
E-1: Si al intentar introducir el nuevo criterio, éste tiene el mismo nombre que otro criterio creado previamente o bien éste criterio está vacío, el sistema muestra un error al usuario.

- **Caso de Uso 5: Creación de dominio de expresión**

Actores principales: Usuario
Condiciones de entrada: El usuario actualmente está en la página inicial de la aplicación y desea crear un nuevo dominio de expresión.
Flujo de eventos:

1. El usuario solicita la opción de crear un nuevo domino de expresión al problema.
2. El sistema permite mediante una ventana para insertar el nuevo dominio y se indicarán cuáles son los dominios que pueden ser seleccionados.
3. El usuario selecciona el dominio de expresión que desea entre las diferentes opciones existentes y selecciona la opción de introducir valores.
4. El sistema permite la inserción mediante una ventana de los valores del dominio de expresión elegido.
5. El usuario introduce los nuevos valores y selecciona la opción de añadir dicho dominio al sistema.
6. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el nuevo dominio se ha añadido correctamente (E-1).

Condiciones de salida: El usuario ha introducido un nuevo dominio de expresión en el problema.
Excepciones:
E-1: Si al intentar introducir el nuevo dominio de expresión, éste tiene el mismo nombre que otro dominio de expresión creado previamente o bien éste dominio de expresión está vacío, el sistema muestra un error al usuario.

- **Caso de Uso 6: Asignación de dominio de expresión**

Actores principales: Usuario
Condiciones de entrada: El usuario actualmente está en la página inicial de la aplicación y desea asignar un dominio de expresión.
Flujo de eventos:

1. El usuario solicita la opción de asignar un dominio de expresión a alguno de los elementos del sistema.
2. El sistema permite la selección de un dominio de expresión mediante una ventana, así como, el elemento al que se desea asignar.
3. El usuario selecciona el elemento o elementos a los que desea asignar el dominio que también tendrá que seleccionar y selecciona la opción añadir asignación.
4. El sistema realiza la asignación correspondiente a los datos introducidos por el usuario y muestra al usuario una confirmación de que se ha realizado correctamente.

Condiciones de salida: El sistema realiza la asignación indicada por el usuario y la tendrá en cuenta para pasos posteriores.
Excepciones: Ninguna.

- **Caso de Uso 7: Valoración de alternativas**

Actores principales: Usuario
Condiciones de entrada: Ya se tiene la estructura del problema (expertos, criterios, alternativas y dominios) y se desea valorar las alternativas.
Flujo de eventos:

- El usuario mediante una ventana realiza las valoraciones de las diferentes alternativas.
- El sistema muestra la información de las distintas alternativas, así como, las valoraciones de las mismas si es que están valoradas.
- El usuario selecciona la alternativa que desea valorar.
- El sistema permite la inserción mediante una ventana de la valoración de la alternativa seleccionada.
- El usuario realiza la inserción de la valoración de la alternativa seleccionada y confirma la acción.
- El sistema asigna la valoración insertada a la alternativa seleccionada y muestra una confirmación de que la acción se ha realizado correctamente.

Condiciones de salida: El usuario ha realizado la valoración de una de las alternativas del problema.
Excepciones: Ninguna.

- **Caso de Uso 8: Selección del método de resolución**

Actores principales: Usuario
Condiciones de entrada: Ya se han realizado todas las valoraciones correspondientes a los elementos introducidos en el problema.
Flujo de eventos:

- El usuario mediante una ventana seleccionar resolver el problema que está abierto en ese momento (E-1).
- El sistema permite al usuario mediante una ventana seleccionar el método de resolución que desee aplicar para resolver el problema.
- El usuario selecciona el método de resolución que desea aplicar para resolver el problema.
- El sistema muestra la información sobre el método de resolución seleccionado y habilita las nuevas ventanas adaptadas a cada uno de los métodos diferentes que se pueden seleccionar (E-2).

Condiciones de salida: El usuario ha realizado la selección del método de resolución con el que resolver el problema y el sistema ha iniciado el proceso de resolución del mismo aplicando dicho método de resolución.

Excepciones:

E-1: Si el usuario selecciona la opción de resolver el problema y no ha realizado un mínimo de valoraciones necesarias, el sistema informará que debe realizarlas y no habilitará la pantalla de selección del método.

E-2: Si las condiciones del método de resolución seleccionado no se adaptan con las características del problema, el sistema informará del error y no iniciará la resolución de dicho problema.

- **Caso de Uso 9: Selección del operador de agregación sin peso**

Actores principales: Usuario

Condiciones de entrada: El usuario desea elegir un operador de agregación después de haber escogido el método de resolución anteriormente.

Flujo de eventos:

- El usuario selecciona un operador de agregación para expertos que no requiere pesos para obtener la solución.
- El sistema valida los datos introducidos y realiza la agregación mostrando al usuario el resultado de la misma (E-1).

Condiciones de salida: El sistema realizar la agregación mediante el operador sin peso seleccionado por el usuario.

Excepciones:

E-1: Si el usuario selecciona un operador de agregación y las valoraciones no cumplen los requisitos de dicho operador, el sistema informa del error y no realiza la agregación.

- **Caso de Uso 10: Selección del operador de agregación con peso**

Actores principales: Usuario

Condiciones de entrada: El usuario desea elegir un operador de agregación con peso después de haber escogido el método de resolución anteriormente.

Flujo de eventos:

- El usuario selecciona un operador de agregación para expertos que requiere pesos para obtener la solución.
- El sistema solicita al usuario mediante una ventana los pesos necesarios para poder realizar la agregación.

- El usuario introduce los valores de los distintos pesos, y selecciona la opción para realizar la agregación.
- El sistema valida los datos introducidos y realiza la agregación mostrando al usuario el resultado de la misma (E-1).

Condiciones de salida: El sistema realizar la agregación mediante el operador con parámetros seleccionado por el usuario.

Excepciones:

E-1: Si el usuario selecciona un operador de agregación y las valoraciones no cumplen los requisitos de dicho operador o los pesos introducidos son erróneos, el sistema informa del error y no realiza la agregación.

4.3.3. Escenarios

Los escenarios describen posibles interacciones que los usuarios puedan hacer de forma ficticia con la aplicación. Estos escenarios deben de ser detallistas para que así los encargados de la implementación tengan más facilidades, ya que es mejor tratar con una situación concreta que con un concepto abstracto [88].

También es una forma de ver como se comportarán los usuarios con el sistema y ver cuáles son las opciones que se pueden mejorar.

Los escenarios están compuestos de una serie de elementos básicos que son:

- Un nombre único e inequívoco.
- Una descripción.
- Los actores participantes.
- El flujo de eventos que define a la acción.

Se pueden realizar escenarios de todas las actividades que se pueden realizar en la aplicación, pero no tiene mucho sentido que el número de escenarios sea elevado, ya que centrándonos en los principales será más que suficiente, para ello se definirán cuatro escenarios que son los siguientes:

- **Escenario 1: Creación de un nuevo problema.**

  Nombre: creación de un nuevo problema

  Descripción: Jesús Pedrero ya ha usado anteriormente sistemas de soporte a la decisión, ahora desea crear un nuevo problema.

  Actores principales: Jesús Pedrero

  Flujo de eventos:

  1. El usuario selecciona opción de nuevo problema.
  2. El sistema muestra la página inicial.
  3. El usuario selecciona la opción añadir criterios.
  4. El sistema muestra el formulario con los datos del criterio que se desea añadir.
  5. El usuario introduce los datos del criterio y pulsa aceptar.
  6. El sistema actualiza la información del problema con los mismos datos.

7. El usuario repite los pasos para la creación del resto de criterios, alternativas y expertos, así como, para los dominios de expresión
8. El sistema actualiza la información del problema.
9. El usuario asigna a todos el mismo dominio de expresión, en este caso numérico, a todos los elementos del problema.
10. El sistema muestra la información del problema actualizada al usuario.

- **Escenario 2: Valoración de alternativas**.

  Nombre: Valoración de alternativas
  Descripción: Jesús Pedrero ya ha usado anteriormente sistemas de soporte a la decisión desea realizar las valoraciones de acuerdo a los elementos del problema.
  Actores principales: Jesús Pedrero
  Flujo de eventos:

  1. El usuario selecciona opción de valorar alternativas.
  2. El sistema muestra la información relativa a las diferentes alternativas del problema.
  3. El usuario selecciona las alternativas que desea valorar.
  4. El sistema muestra el formulario para que el usuario introduzca la valoración deseada.
  5. El usuario introduce la valoración de la alternativa deseada y pulsa aceptar.
  6. El sistema actualiza la información de las alternativas con los nuevos datos introducidos.
  7. El usuario repite los pasos 3-6 para todas las alternativas que desea valorar.
  8. El sistema actualiza la información de las distintas alternativas con las valoraciones realizadas.

- **Escenario 3: Selección del operador de agregación sin peso.**

  Nombre: selección del operador de agregación sin peso
  Descripción: Jesús Pedrero ya ha usado anteriormente sistemas de soporte a la decisión y desea seleccionar un operador de agregación. Previamente se han introducido los criterios, alternativas, expertos, dominios y valoraciones del problema.
  Actores principales: Jesús Pedrero
  Flujo de eventos:

  1. El usuario selecciona un modelo de resolución.
  2. El sistema muestra una lista desplegable con los operadores de agregación disponibles para ese modelo de resolución.
  4. El usuario selecciona un operador de agregación para agregar las valoraciones.
  5. El sistema agrega las valoraciones con el operador seleccionado y muestra un ranking de las alternativas.
  6. El sistema muestra gráficamente el ranking de las alternativas.

- **Escenario 4: Selección del operador de agregación con peso.**

  Nombre: selección del operador de agregación con peso
  Descripción: Jesús Pedrero ya ha usado anteriormente sistemas de soporte a la decisión y desea seleccionar un operador de agregación. Previamente se han introducido los criterios, alternativas, expertos, dominios y valoraciones del problema.
  Actores principales: Jesús Pedrero
  Flujo de eventos:

  1. El usuario selecciona un modelo de resolución.
  2. El sistema muestra una lista desplegable con los operadores de agregación disponibles para ese modelo de resolución.
  4. El usuario selecciona un operador de agregación con peso para las valoraciones.
  5. El sistema muestra una ventana para introducir el peso en función de los criterios o expertos.
  6. El sistema agrega las valoraciones con el operador seleccionado y muestra un ranking de las alternativas.
  7. El sistema muestra un gráfico con ranking de las alternativa.

  Este proyecto se centrará en los dos últimos escenarios.

4.3.4. Modelo del dominio

El modelo de dominio muestra a los diseñadores y desarrolladores de software un conjunto de clases conceptuales significativas en un dominio concreto de problema [89].

Este diagrama se usa conocer cuáles son los objetos dentro de la aplicación y como se interrelacionan entre ellos.

Para crear el modelo del dominio, se identifican las clases que intervienen en el dominio del problema y a continuación se observa cómo se interrelacionan las diferentes clases y se crea un diagrama en UML, además se añaden los atributos a la clases [88, 89].

Un modelo de dominio suelen mostrar los siguientes elementos:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las distintas clases conceptuales.
- Atributos de las clases conceptuales.

Con el modelo de dominio podemos tener de forma clara y simple la información de todo nuestro problema, de forma resumida, y esto supone una ventaja puesto que se puede observar claramente cómo se relacionan las clases entre si y ver todos los elementos que componen el problema.

A continuación el Figura 4.5 podemos observar el modelo de dominio de la aplicación en un diagrama UML:

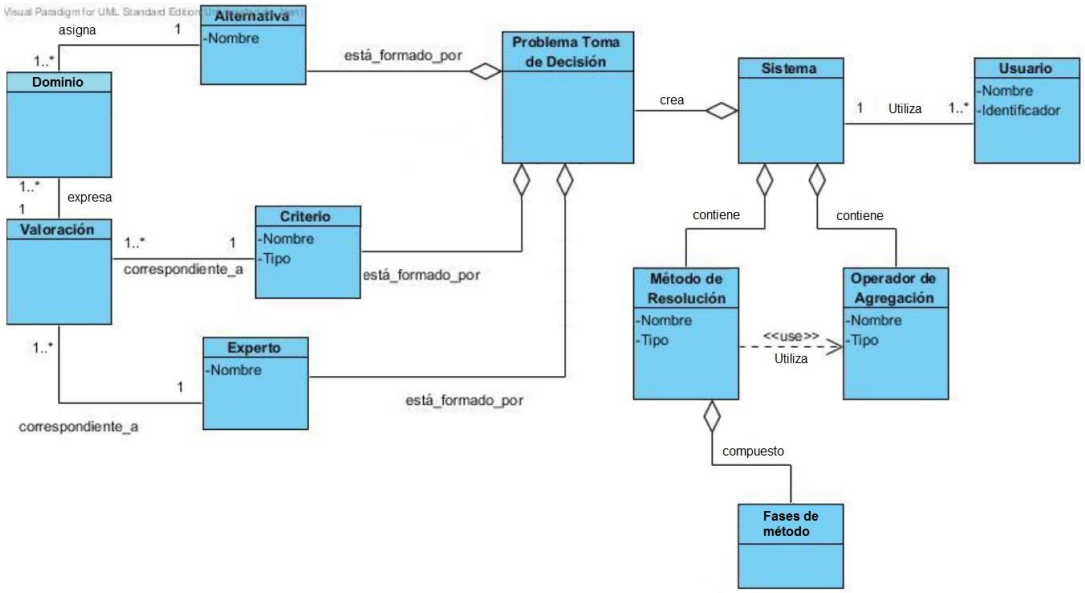

Figura 4.5 Diagrama UML del Modelo de dominio.

A continuación, se describen con mayor detalle el diagrama que se observa en la Figura 4.5:

- Usuario: Clase que representa a los usuarios que interactúan con el sistema.
- Sistema: Esta clase representa a la toda la aplicación y es donde se crean los diferentes elementos que contiene un problema, desde su planteamiento hasta su resolución.
- Método de resolución: En esta clase contiene todos los tipos diferentes de modelos de resolución, dependiendo del modelo de resolución escogido tendremos acceso a unos operadores de agregación o a otros.
- Operador de agregación: En esta clase podemos encontrar distintos tipos de operadores de agregación disponibles dependiendo del modelo de resolución escogido previamente.
- Problema de toma de decisión: En esta clase representa un problema de toma de decisión y se componen de los siguientes elementos: expertos, criterios, alternativas y dominio.
- Criterio: Clase que representa un criterio definido dentro de un problema de toma de decisiones.
- Alternativa: Clase que representa una alternativa definida dentro de un problema de toma de decisiones.
- Experto: Clase que representa un experto definido dentro de un problema de toma de decisiones.
- Valoración: Clase que representa un criterio concreto definido dentro de un problema de toma de decisiones.
- Dominio: Clase que representa un dominio de expresión concreto definido para valorar las alternativas.

4.4. Diseño del sistema

Todo el proceso de ingeniería del software es imprescindible en un proyecto para que así tenga calidad, consistencia y robustez.

Todas las partes que componen el proceso de ingeniería del software son importantes y no se puede decir que una es más importante que otra, pero la parte de diseño es una de las partes más delicadas y complejas de realizar.

Es parte es delicada debido a que si no se hace de forma correcta la implementación puede ser incorrecta y provocar funcionamientos incorrectos o incompletos, tampoco es fácil en ocasiones pasar del diseño a la implementación convirtiéndose este proceso en un proceso complejo y tedioso.

Sin embargo, el sistema no se suele diseñar de una vez, sino que hay que realizar varias pasadas hasta llegar a una versión final [88].

4.4.1. Diagrama de clases de diseño

En un diagrama de clases de diseño se describe la estructura global de un sistema donde se muestran cómo se interrelacionan sus clases, con sus atributos y operaciones [88].

El diseño se realiza en base a la estructura que tendrá el sistema y de todos los componentes que componen dicho sistema [88].

Los componentes principales de un diagrama de clases de diseño son los siguientes:

- Clases: Es el principal componente del que se componen los diagramas de este tipo. Además están compuestos por una serie de atributos y de operaciones que se pueden realizar con ellos.
- Relaciones: Son la interrelaciones con las que se conectan las clases. Existen 4 tipos de relaciones:
    - Generalización: Es una relación de especialización.
    - Asociación: Es una relación estructural. Pueden existir dos tipos: agregación y composición.
    - Realización: Es una relación contractual, en la que una clase requiere una serie de condiciones para que exista la relación.
    - Dependencia: Representa una relación de uso.

Flinstones se estructura de forma que módulos se separan en diferentes componentes, de esta manera existe una separación entre el sistema de soporte a la decisión, el conjunto de elementos de representación de la interfaz, los métodos de resolución incluidos y los operadores de agregación implementados para resolver los problemas de toma de decisiones.

Los principales módulos básicos que forman parte de la aplicación son los siguientes:

- Bibliotecas o librerías: Estas librerías ofrecen las estructuras y procedimientos con el objetivo de apoyar y facilitar la resolución de problemas de decisión. Estas librerías incluyen elementos tales como: los expertos, los criterios, las alternativas y los dominios de expresión.
- Interfaz gráfica de usuario: La interfaz de usuario (GUI) permite a los usuarios interactuar con el SSD.
- Método de resolución: Estos métodos desarrollan las diferentes metodologías basadas en distintos tipos de datos, con el fin de resolver los problemas de toma de decisiones.
- Operadores de agregación: Estos operadores implementan el conjunto de operadores de agregación que se han utilizado para agregar la distinta información involucrada en el problema de decisión.

A continuación se muestra en la Figura 4.6 un diagrama de clases de la creación de un operador de agregación, como ya se ha comentado anteriormente no necesitamos conocer todos los componentes del sistema sino únicamente los componentes que están involucrados con la creación de un operador de agregación.

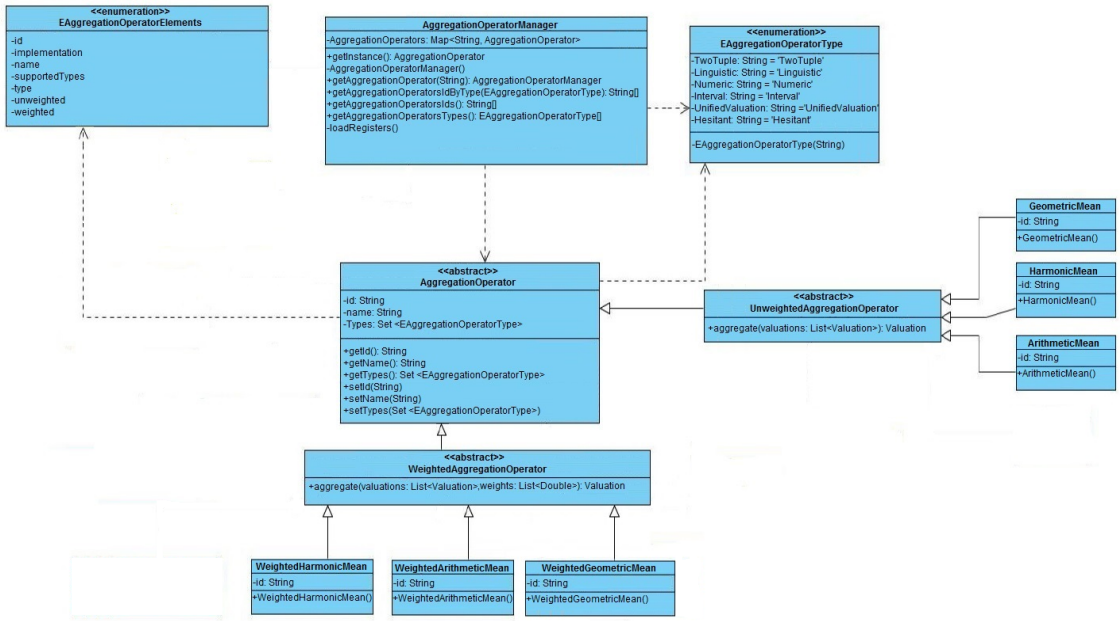

Figura 4.6 Diagrama de clases de la creación de un operador de agregación.

Las ventajas que tiene la separación de los principales componentes del sistema software según la funcionalidad son:

- Permite la agregación tanto de métodos de resolución como de atributos de forma simple y sin tener la necesidad de realizar ninguna modificación en la interfaz del sistema.
- Permite ampliar la funcionalidad del sistema, únicamente añadiendo nuevos plug-ins sin necesidad de tener que cambiar la estructura interna del sistema.
- Se pueden añadir operadores de agregación y modelos de resolución de forma sencilla y en poco tiempo mediante plug-in.

4.4.2. Diagrama de secuencia

El diagrama de secuencia de UML muestra la forma en que los objetos se comunican entre sí al transcurrir el tiempo. [88].

Si se dispone de la descripción de cada caso de uso, entonces usamos esas descripciones para hacer un guía de los objetos que debemos seguir para realizar una secuencia de pasos.

- Los diagramas de secuencia muestran:
  - Los objetos participando en la interacción.
  - La intercambiados de mensajes.

1. Un diagrama de secuencia contiene:
   - Objetos con su línea de vida
   - Mensajes intercambiados entre objetos de una secuencia ordenada.
   - Línea de vida activa

Tipos de mensajes:

- Los mensajes *sincrónicos* son los mensajes que más se utilizan. Su uso implica que el emisor del mensaje espera que la activación del método asociado al destinatario finalice para poder continuar su actividad.
- Los mensajes de *retorno* se encargan de devolver el control al objeto que originó el mensaje que inició la activación.
- Los mensajes *asincrónicos* el emisor no espera el término de la activación invocada por el destinatario.

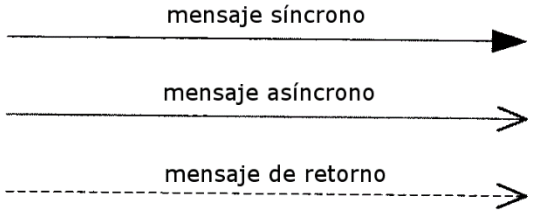

Figura 4.7 Tipos de mensajes en un diagrama de secuencia.

Para este proyecto se ha realizado un total de cinco diagramas de secuencia que corresponden a los casos de uso más importante, que se definieron en la especificación de requerimientos.

- Diagrama de secuencia 1: "Creación de un nuevo problema"

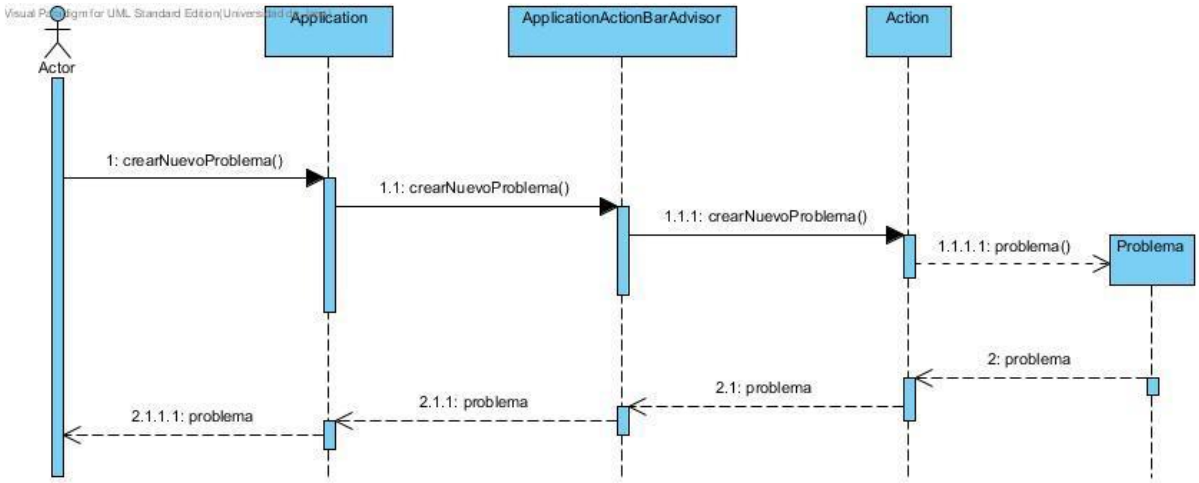

Figura 4.8 Diagrama de secuencia 1 "Creación de un problema".

- Diagrama de secuencia 2: "Creación de criterio"

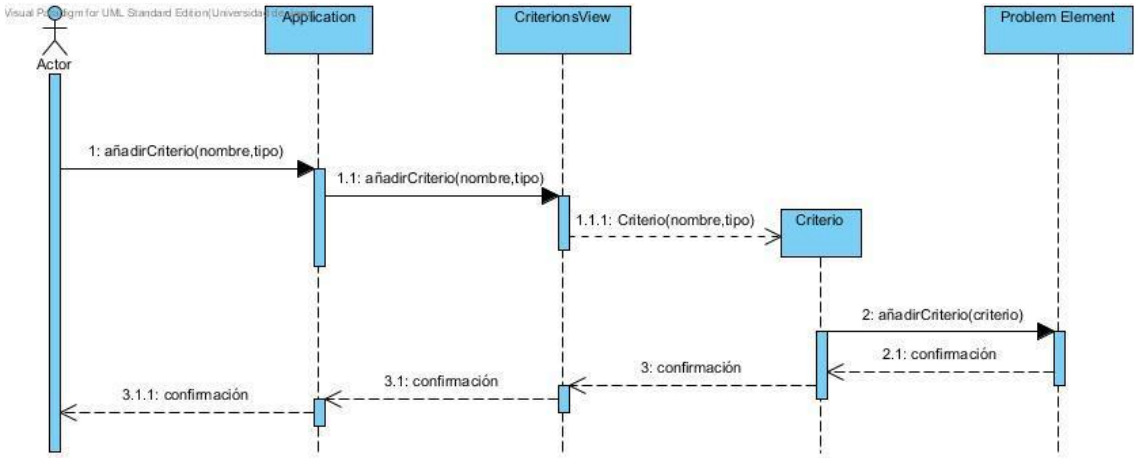

Figura 4.9 Diagrama de secuencia 2 "Creación de criterio".

- Diagrama de secuencia 3: "Valoración de alternativas"

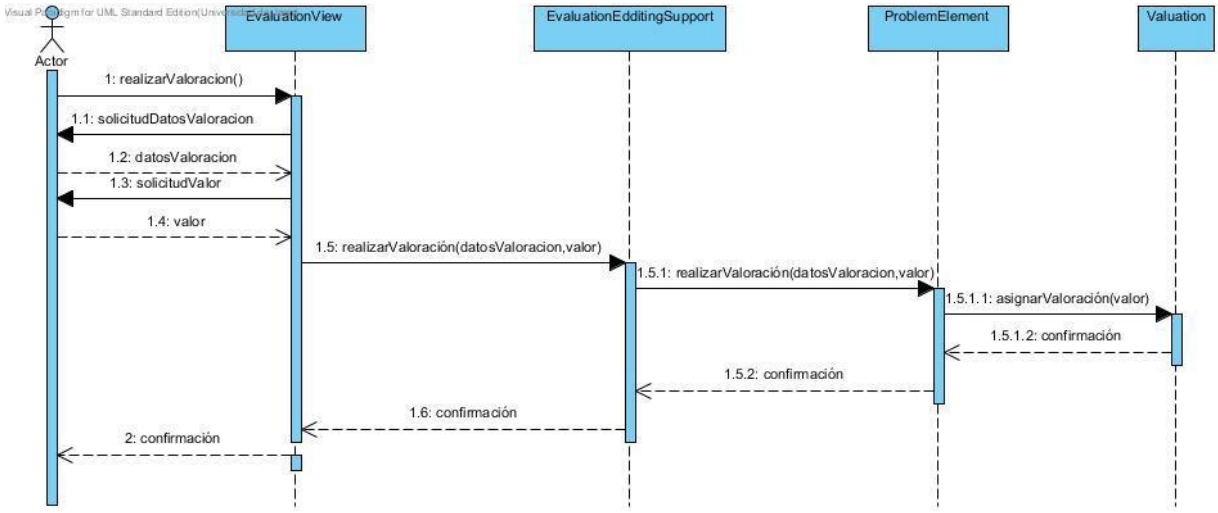

Figura 4.10 Diagrama de secuencia 3 "Valoración de alternativas".

- Diagrama de secuencia 4: "Selección del operador de agregación sin peso"

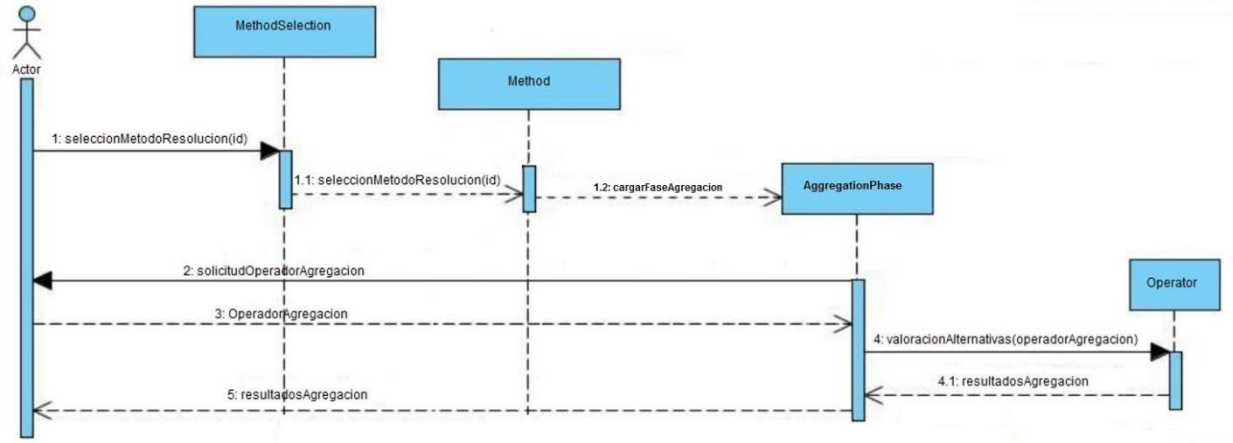

Figura 4.11 Diagrama de secuencia 4 "Selección del operador de agregación sin peso".

- **Diagrama de secuencia 5: "Selección del operador de agregación con peso"**

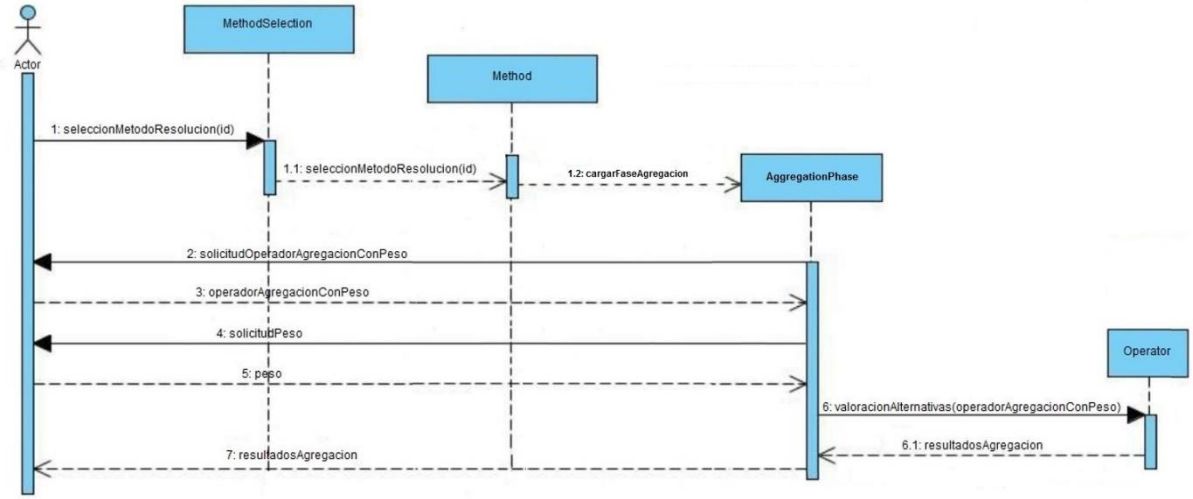

Figura 4.12 Diagrama de secuencia 5 "Selección del operador de agregación con peso".

4.5. Diseño de interfaz

Los elementos de diseño de la interfaz del software permiten que la información fluya hacia dentro y fuera del sistema, y definen cómo están comunicados los componentes que forman parte de la arquitectura [95].

Existen tres elementos importantes del diseño de la interfaz:

- La interfaz de usuario.
- Las interfaces externas.
- Las Interfaces internas.

Nos centraremos en la interfaz de usuario, la cual define el aspecto final que tendrá la aplicación, teniendo en cuenta factores estéticos, ergonómicos y técnicos.

4.5.1. Estilo

En esta parte del diseño se define unas guías de estilo que tendrá nuestro proyecto, como puede ser el tipo, tamaño, estilo y alineación de letra que tendrán las pantallas que forman nuestro proyecto.

Ya que este proyecto tiene como propósito ampliar la funcionalidad de un software ya existente el estilo seguirá siendo el mismo que ya hay en Flintstones.

4.5.2. Metáforas

En esta parte se indican el significado de los distintos iconos que forman parte de Flintstones y a continuación se detallarán los más relevantes para este proyecto.

Figura 4.13 Metáfora "Expertos".

En la Figura 4.13 podemos observar el cuadro de dialogo donde se encuentran los expertos, donde el icono  indica el conjunto de expertos y el icono  indica a cada experto.

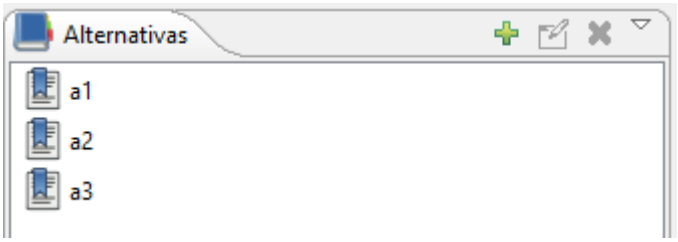

Figura 4.14 Metáfora "Alternativas".

En la Figura 4.14 podemos observar el cuadro de dialogo donde se encuentran las alternativas, donde el icono  indica el conjunto de alternativas y el icono  indica a cada alternativa.

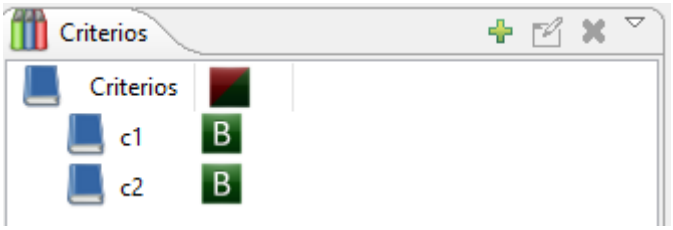

Figura 4.15 Metáfora "Criterios".

En la Figura 4.15 podemos observar el cuadro de dialogo donde se encuentran los criterios, donde el icono  indica el conjunto de criterios y el icono  indica a cada criterio.

Los criterios pueden ser de beneficio como se muestra en la Figura 4.16 o bien pueden ser de coste como se muestra en la Figura 4.17.

Figura 4.16 Metáfora "Criterio de beneficio".

Figura 4.17 Metáfora "Criterio de coste".

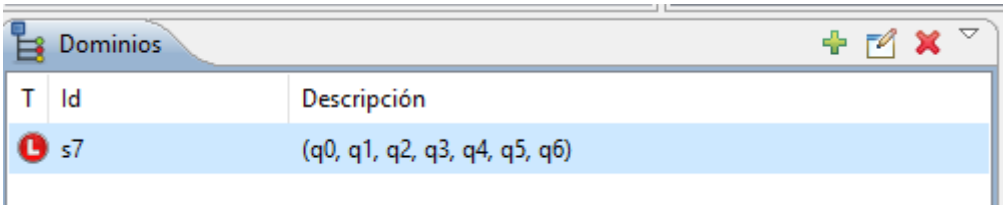

Figura 4.18 Metáfora "Dominio de expresión".

En la Figura 4.18 podemos observar el cuadro de dialogo donde se encuentran los dominios expresión, donde el icono  indica los dominios de expresión.

Tanto en los expertos, alternativas, criterios y dominios de expresión tenemos la opción de añadir más, modificar los que ya hay o bien eliminarlos, y los iconos asociados a estas opciones son los que se observan en la Figura 4.19.

Figura 4.19 Metáfora "Acciones asociadas a los diferentes elementos de un problema de toma de decisión".

Por último indicar que en la siguiente Figura nos encontramos con los iconos que representan a las diferentes fases que se componen un problema de toma de decisión, que son:

1. Marco de evaluación (Framework).
2. Estructura del marco de evaluación (Framework Structuring).
3. Recogida de información (Gathering).
4. Valoración de alternativas (Rating).
5. Análisis (Sensitivity Analysis).

Figura 4.20 Metáfora "Fases de un proceso de toma de decisión en Flintstones".

4.5.3. Prototipo de Pantalla de la fase de agregación

Ya que el propósito de este proyecto como ya se ha comentado anteriormente es la de ampliar un software ya existente en esta parte del diseño nos centraremos en el diseño de las pantallas relacionadas con los operadores de agregación.

La pantalla que está relacionada con los operadores de agregación es como el prototipo que se muestra en la Figura 4.21, y está pantalla aparece justo después de elegir el modelo de resolución.

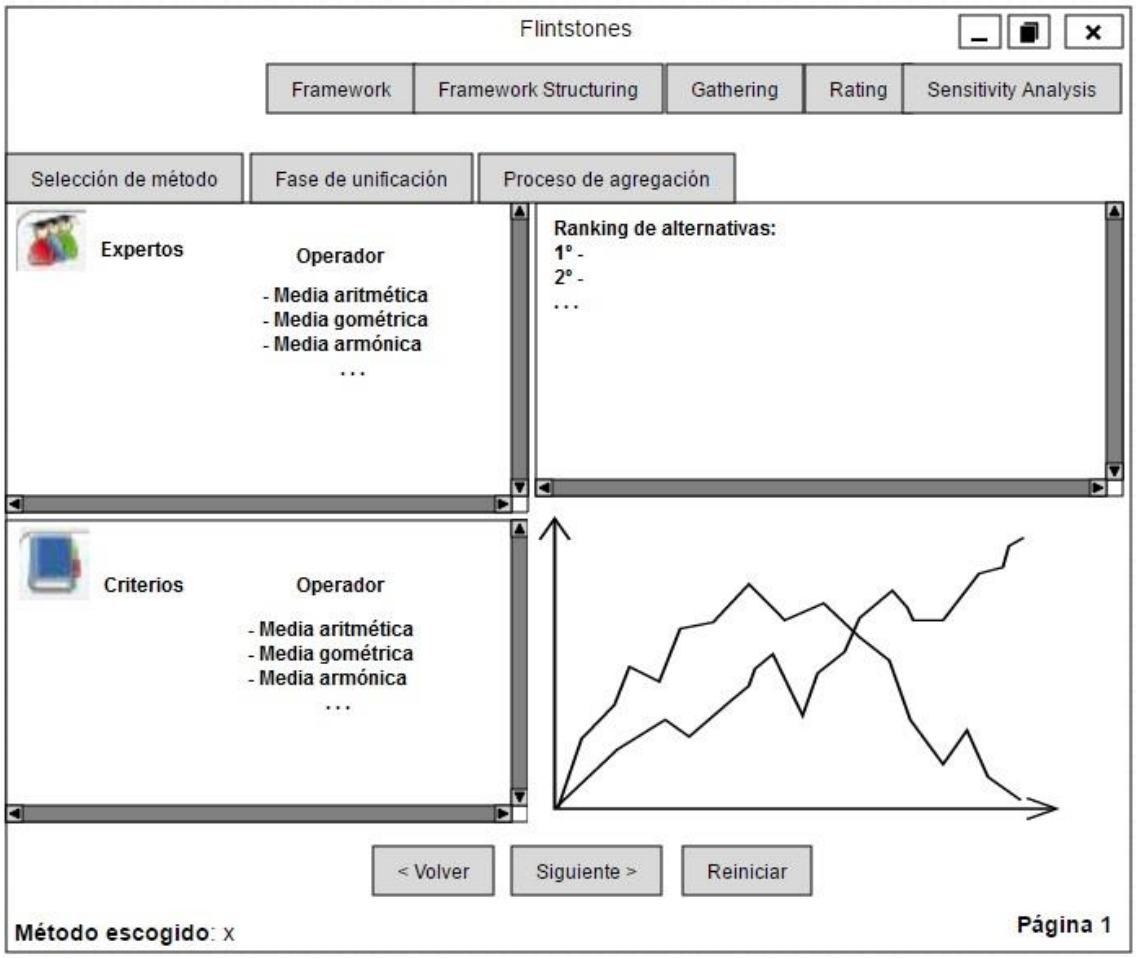

Figura 4.21 Selección de los operadores de agregación.

Una vez escogido el operador de agregación deseado ya hemos terminado la parte de agregación, tal y como se muestra en la Figura 4.22.

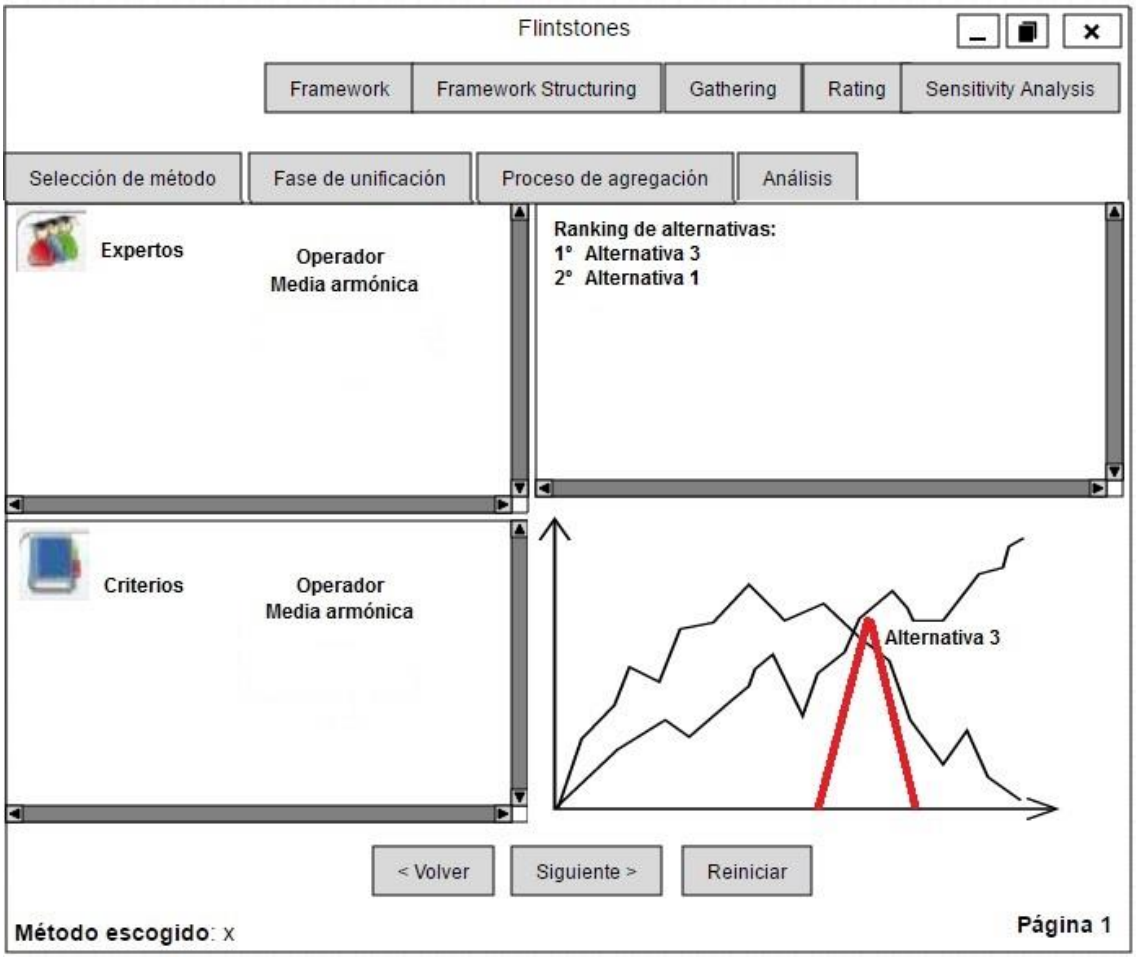

Figura 4.22 Fase de agregación.

A continuación se explica con mayor detalle cada parte de la pantalla anterior.

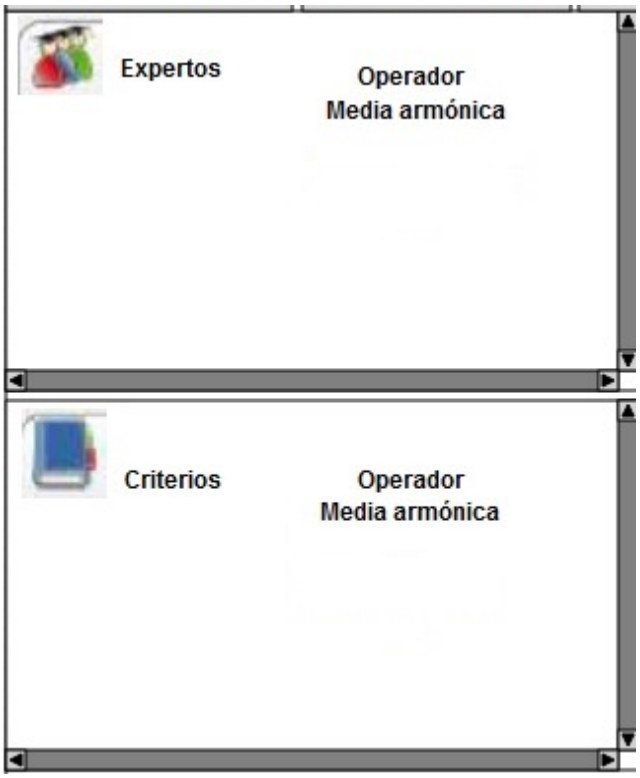

Figura 4.23 Fase de agregación parte 1.

Como podemos observar en la Figura 4.23 se ha escogido de ejemplo el operador de agregación de media armónica.

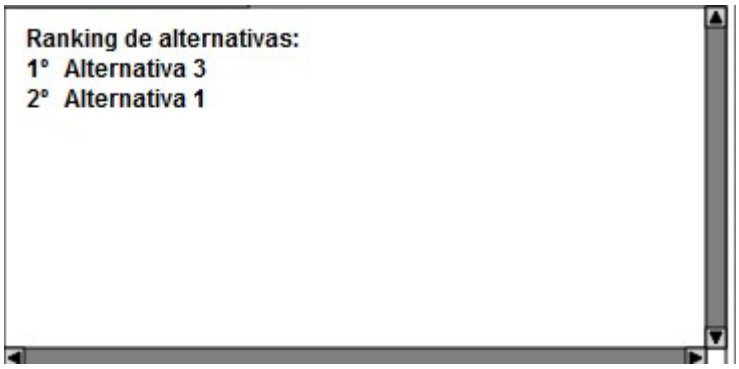

Figura 4.24 Fase de agregación parte 2.

En la Figura 4.24 observamos el ranking de alternativas con el operador de agregación escogido.

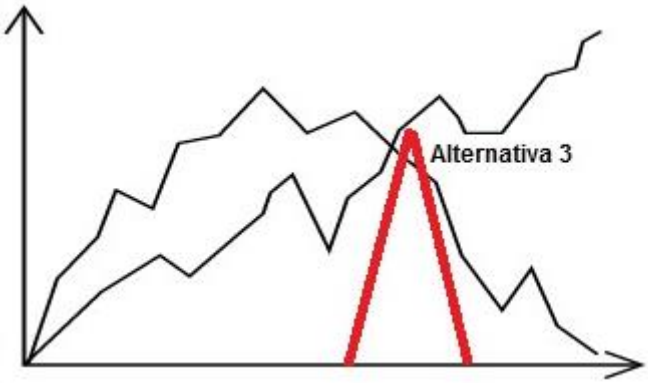

Figura 4.25 Fase de agregación parte 3.

En la Figura 4.25 podemos observar gráficamente el ranking de alternativas anteriores.

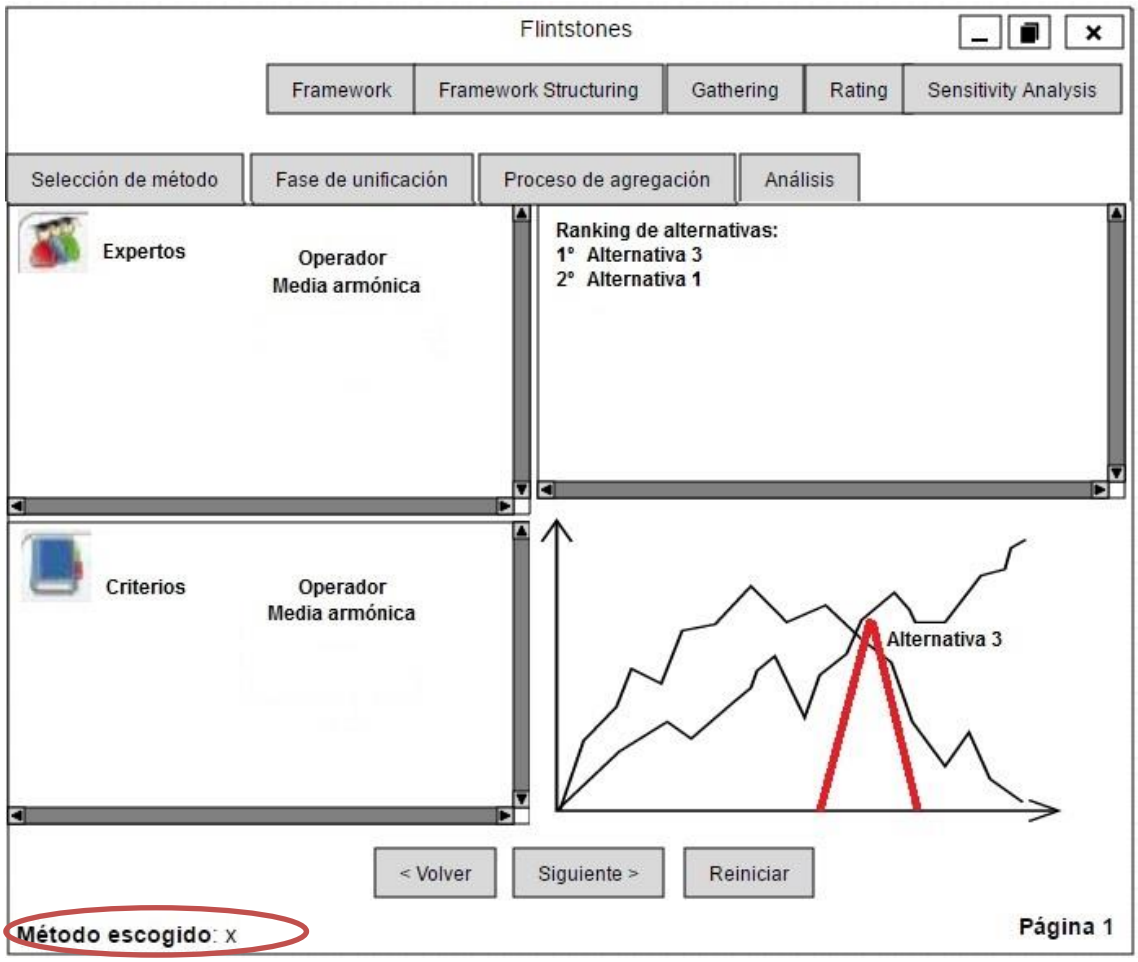

Figura 4.26 Fase de agregación parte 4.

En la Figura 4.26 observamos un círculo rojo en el podemos ver que el modelo de resolución escogido es el modelo 2-Tuplas, y dependiendo del modelo de resolución escogido los operadores de agregación pueden variar.

En caso de elegir un operador de agregación con peso, el sistema nos mostrará una ventana como se muestra en la Figura 4.27 para introducir los pesos, debemos de tener en cuenta que la suma debe estar en el intervalo [0, 1].

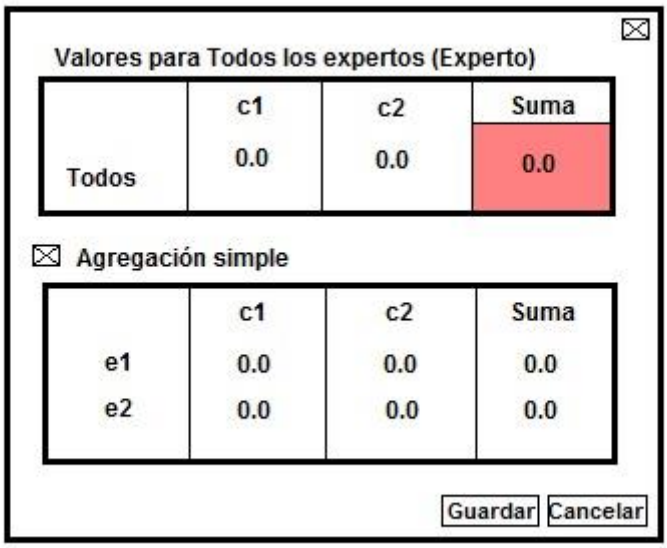

Figura 4.27 Ventana para introducir pesos.

4.6. Implementación

La implementación tiene como objetivo convertir el diseño en código. Este código utiliza un lenguaje de programación para ello se escoge un lenguaje y se codifica. Además deberemos de también escoger un entorno para desarrollar nuestro lenguaje de programación. Este paso ya depende mucho si se empieza desde cero o sí bien ya existe una base y hay que partir de ella.

Por lo tanto el siguiente paso será escoger tanto el lenguaje de programación como el entorno donde desarrollaremos este lenguaje.

4.6.1. Lenguaje de programación

En este caso no se comienza de cero sino que tenemos una aplicación y queremos ampliar su funcionamiento, y ésta aplicación llamada Flintstones ha sido desarrollada en Java, así que Java será nuestro lenguaje de programación

Una de las principales características de Java es que se trata de un lenguaje compilado e interpretado. En Java todo programa debe de compilarse y dando lugar o generando un código de bytecodes, el cual será interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma [90].

Java es un lenguaje orientado a objetos de propósito general. Aunque Java comenzará a ser conocido como un lenguaje de programación de applets que se ejecutan

en el entorno de un navegador web, se puede utilizar para construir cualquier tipo de proyecto [90].

Su sintaxis es muy parecida a la de C y C++ pero hasta ahí llega el parecido. Java no es una evolución ni de C++ ni un C++ mejorado.

La seguridad en Java es alta debido a que en su diseño se tuvo mucho en cuenta este punto. Existiendo varios niveles de seguridad en Java, desde el ámbito del programador, hasta el ámbito de la ejecución en la máquina virtual.

4.6.2. Herramienta de desarrollo

La herramienta de desarrollo para este proyecto es Eclipse RCP (plataforma de cliente enriquecido), ya que es la herramienta con la que se ha desarrollado Flintstones, de ahí que se utilice la misma herramienta para ampliar la funcionalidad de Flintstones.

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma [91].

Esta herramienta se basa en un conjunto de plug-ins dinámicos incorporados a lo largo del desarrollo del proyecto. La principal utilidad que aporta RCP es que permite desarrollar aplicaciones complejas en un tiempo reducido, además facilita el mantenimiento y es una herramienta fácil de extender. Por otra parte, los componentes que forman parte del RCP son de calidad y son de código abierto.

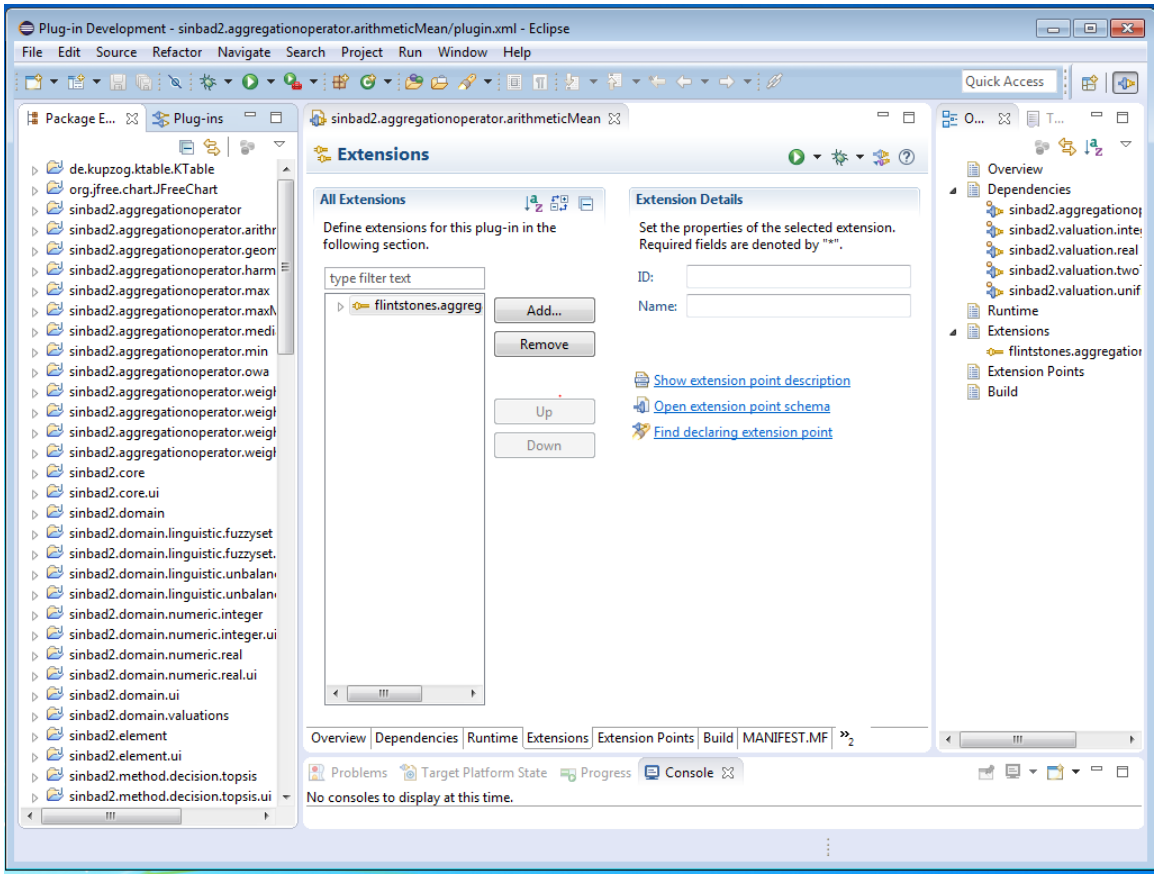

Figura 4.28 Entorno de trabajo de Eclipse RCP.

4.6.3. Implementación software

La implementación software se divide en dos partes, por una parte tenemos los operadores básicos y por otra parte tenemos los operadores de agregación.

En el anexo 1 de este trabajo se muestran los pasos necesarios para la creación de un operador de agregación.

A continuación se mostrará la estructura de los operadores básicos y de los operadores de agregación, así como también se mostrará la implementación de ambos tipos de operadores.

Los operadores básicos tienen en común con los operadores de agregación la estructura, por lo tanto mostrando la estructura de uno de ellos es suficiente para entender al otro.

4.6.3.1. Operadores básicos

La estructura de los operadores es como se muestra en la Figura 4.29, en esta sección solo nos centraremos en la implementación, por lo tanto solo mencionaremos las dependencias para ver cómo se tratan las dependencias deberemos ir al anexo 1 de este trabajo.

Para ver la estructura de los operadores, se ha escogido al operador Max como ejemplo, y a continuación se mostrará su estructura paso a paso.

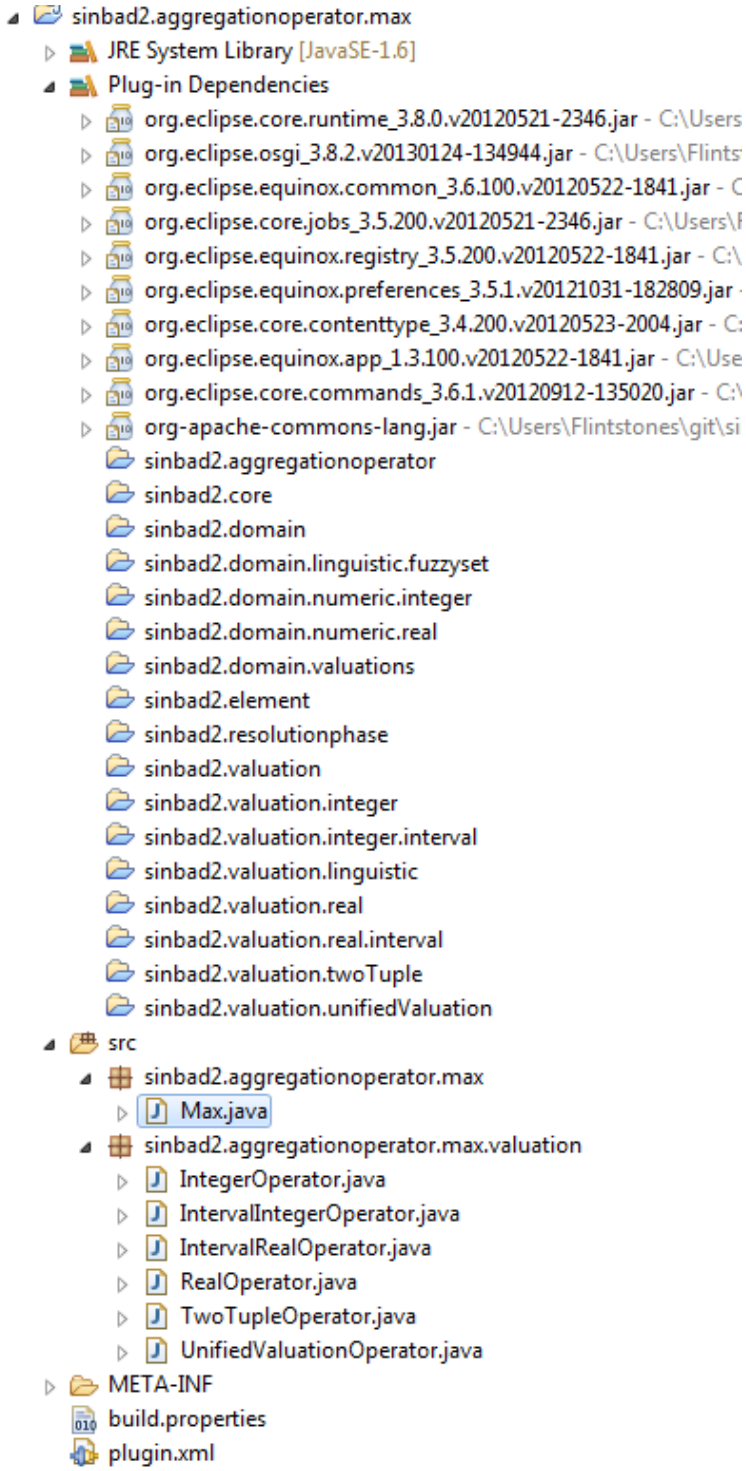

Figura 4.29 Estructura de un operador en Eclipse RCP

Todos los operadores que se crean, contiene dentro de la carpeta "src" dos package, en el primero nos encontramos la implementación como podemos observar en la Figura 4.30, donde se encuentra las validaciones de las valoraciones soportadas.

```java
package sinbad2.aggregationoperator.max;

import java.util.LinkedList;

public class Max extends UnweightedAggregationOperator {

    public static final String ID = "flintstones.aggregationoperator.max";

    @Override
    public Valuation aggregate(List<Valuation> valuations) {
        Validator.notNull(valuations);

        List<Valuation> auxValuations = new LinkedList<Valuation>();
        for (Valuation valuation : valuations) {
            if (valuation != null) {
                auxValuations.add(valuation);
            }
        }

        if (auxValuations.size() != valuations.size()) {
            valuations = auxValuations;
        }

        if(valuations.size() > 0) {
            for(Valuation valuation: valuations) {
                if(valuation instanceof IntegerValuation) {
                    return IntegerOperator.aggregate(valuations);
                } else if(valuation instanceof RealValuation) {
                    return RealOperator.aggregate(valuations);
                } else if(valuation instanceof IntegerIntervalValuation){
                    return IntervalIntegerOperator.aggregate(valuations);
                } else if(valuation instanceof RealIntervalValuation) {
                    return IntervalRealOperator.aggregate(valuations);
                } else if(valuation instanceof TwoTuple){
                    return TwoTupleOperator.aggregate(valuations);
                }  else if(valuation instanceof UnifiedValuation) {
                    return UnifiedValuationOperator.aggregate(valuations);
                } else {
                    throw new IllegalArgumentException("Not supported type");
                }
            }
        }

        return null;

    }

}
```

Figura 4.30 Función de agregación del operador Max

En el segundo package al que llamamos valuation, es donde encontramos la validación de los distintos dominios de expresión que admite ese operador, teniendo un archivo .java por cada dominio de expresión que admita este operador, a continuación en la Figura 4.31 se define la implementación del operador para el dominio de expresión 2-Tuplas.

```
package sinbad2.aggregationoperator.max.valuation;

import java.util.Collections;

public class TwoTupleOperator {

    private TwoTupleOperator() {}

    public static Valuation aggregate(List<Valuation> valuations) {
        Valuation result = null;
        FuzzySet domain = null;
        List<Valuation> values = new LinkedList<Valuation>();

        for(Valuation valuation: valuations) {
            Validator.notIllegalElementType(valuation, new String[] {TwoTuple.class.toString()});

            if(domain == null) {
                domain = (FuzzySet) valuation.getDomain();
            } else if(!domain.equals(valuation.getDomain())) {
                throw new IllegalArgumentException("Invalid domain");
            }

            values.add(valuation);
        }

        if(domain != null) {
            Collections.sort(values);
            result = values.get(values.size() - 1);
            result = (Valuation) result.clone();
        }

        return result;
    }

}
```

Figura 4.31 Validación del dominio de expresión 2-Tuplas

Ya hemos visto la implantación del operador Max, todos los demás operadores básicos se estructuran de la misma forma cambiando únicamente la validación de los distintos dominios de expresión soportado por el operador.

4.6.3.2. Operadores de agregación

A continuación se explicará la implementación de los operadores de agregación, ya que la estructura se ha visto anteriormente.

Lo único que hay que tener en cuenta con estos operadores es si son ponderados o no, ya que su implementación cambia un poco, como observaremos a continuación.

Comenzaremos por la media aritmética que será el operador base, ya que los demás no suelen cambiar mucho respecto a este.

Media aritmética

```
package sinbad2.aggregationoperator.arithmetic.valuation;

import java.util.List;

public class TwoTupleOperator {

    private TwoTupleOperator() {}

    public static Valuation aggregate(List<Valuation> valuations) {
        TwoTuple result = null;
        double beta = 0;
        FuzzySet domain = null;
        int size = valuations.size();

        for(Valuation valuation : valuations) {
            Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

            if(domain == null) {
                domain = (FuzzySet) valuation.getDomain();
            } else if(!domain.equals(valuation.getDomain())) {
                throw new IllegalArgumentException("Invalid domain");
            }

            beta += (((TwoTuple) valuation).calculateInverseDelta());
        }

        beta /= size;

        if (domain != null) {
            result = (TwoTuple) valuations.get(0).clone();
            result.calculateDelta(beta);
        }

        return result;
    }
}
```

Figura 4.32 Fórmula del operador de agregación media aritmética para el modelo 2-Tuplas.

Media armónica

```
  public static Valuation aggregate(List<Valuation> valuations) {
      TwoTuple result = null;
      double beta = 0, aux =0;
      FuzzySet domain = null;

      for(Valuation valuation : valuations) {
          Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

          if(domain == null) {
              domain = (FuzzySet) valuation.getDomain();
          } else if(!domain.equals(valuation.getDomain())) {
              throw new IllegalArgumentException("Invalid domain");
          }

          beta += 1d / ((TwoTuple) valuation).calculateInverseDelta();
      }

      aux = valuations.size() / beta;
      beta = aux;

      if (domain != null) {
          result = (TwoTuple) valuations.get(0).clone();
          result.calculateDelta(beta);
      }

      return result;
  }
```

Figura 4.33 Fórmula del operador de agregación media armónica para el modelo 2-Tuplas.

Media geométrica

```
public static Valuation aggregate(List<Valuation> valuations) {
    TwoTuple result = null;
    double beta = 1, aux = 0;
    FuzzySet domain = null;
    int size = valuations.size();

    for(Valuation valuation : valuations) {
        Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

        if(domain == null) {
            domain = (FuzzySet) valuation.getDomain();
        } else if(!domain.equals(valuation.getDomain())) {
            throw new IllegalArgumentException("Invalid domain");
        }

        beta *= (((TwoTuple) valuation).calculateInverseDelta());
    }

    aux = Math.pow(beta, 1d / size);
    beta = aux;

    if (domain != null) {
        result = (TwoTuple) valuations.get(0).clone();
        result.calculateDelta(beta);
    }

    return result;
}
```

Figura 4.34 Fórmula del operador de agregación media geométrica para el modelo 2-Tuplas.

Media aritmética ponderada

Los operadores con peso son implementados de forma algo distinta a los operadores sin peso, en la fase de selección dentro de la fase de agregación de operadores tenemos un implementación estándar para los operadores con peso como podemos observar en la Figura 4.35 y Figura 4.36.

```java
package sinbad2.aggregationoperator.weightedmean;

import java.util.LinkedList;

public class Weightedmean extends WeightedAggregationOperator {

    @Override
    public Valuation aggregate(List<Valuation> valuations, List<Double> weights) {
        Validator.notNull(valuations);
        Validator.notNull(weights);

        int valuationsSize = valuations.size();
        int weightsSize = weights.size();

        if(valuationsSize > 0) {
            if(valuationsSize == weightsSize) {
                double sum = 0;
                for(Double weight : weights) {
                    if(weight == null) {
                        throw new IllegalArgumentException("Null weight");
                    }
                    sum += weight;
                }

                if(sum != 1) {
                    List<Double> newWeights = new LinkedList<Double>();
                    for(Double w : weights) {
                        newWeights.add(w);
                    }
                    weights.clear();

                    for(Double w : newWeights) {
                        weights.add(w / sum);
                    }
                }

                double remainder = 0;
                int nullValuations = 0;
                for(int i = 0; i < valuationsSize; i++) {
                    if(valuations.get(i) == null) {
                        remainder += weights.get(i);
                        nullValuations++;
                    }
                }
```

Figura 4.35 Implementación de un operador con peso. Parte 1

```
                if(nullValuations > 0) {
                    List<Valuation> auxValuations = new LinkedList<Valuation>();
                    List<Double> auxWeights = new LinkedList<Double>();

                    remainder /= (double) (valuationsSize - nullValuations);
                    if(remainder < 1) {
                        remainder = 1;
                    }
                    for(int i = 0; i < valuationsSize; i++) {
                        if(valuations.get(i) != null) {
                            auxWeights.add(weights.get(i) * remainder);
                            auxValuations.add(valuations.get(i));
                        }
                    }

                    valuations = auxValuations;
                    weights = auxWeights;
                }

                for(Valuation valuation : valuations) {
                    if(valuation instanceof IntegerValuation) {
                        return IntegerOperator.aggregate(valuations, weights);
                    } else if(valuation instanceof RealValuation) {
                        return RealOperator.aggregate(valuations, weights);
                    } else if(valuation instanceof TwoTuple) {
                        return TwoTupleOperator.aggregate(valuations, weights);
                    } else if(valuation instanceof UnifiedValuation) {
                        return UnifiedValuationOperator.aggregate(valuations, weights);
                    } else {
                        throw new IllegalArgumentException("Not supported type.");
                    }
                }
            } else {
                throw new IllegalArgumentException("Illegal number of weights.");
            }
        }
        return null;
    }

}
```

Figura 4.36 Implementación de un operador con peso. Parte 2

```
package sinbad2.aggregationoperator.weightedmean.valuation;

import java.util.LinkedList;

public class TwoTupleOperator {

    private TwoTupleOperator() {}

    public static Valuation aggregate(List<Valuation> valuations, List<Double> weights) {
        TwoTuple result = null;
        double beta = 0;
        List<Double> measures = new LinkedList<Double>();
        FuzzySet domain = null;

        for(Valuation valuation : valuations) {
            Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

            if (domain == null) {
                domain = (FuzzySet) valuation.getDomain();
            } else if (!domain.equals(valuation.getDomain())) {
                throw new IllegalArgumentException("Invalid domain");
            }
            measures.add(((TwoTuple) valuation).calculateInverseDelta());
        }

        if (domain != null) {
            int size = measures.size();
            for (int i = 0; i < size; i++) {
                beta += weights.get(i) * measures.get(i);
            }

            result = (TwoTuple) valuations.get(0).clone();
            result.calculateDelta(beta);
        }
        return result;
    }

}
```

Figura 4.37 Fórmula del operador de agregación media aritmética ponderada para el modelo 2-Tuplas.

Media armónica ponderada

```
    public static Valuation aggregate(List<Valuation> valuations, List<Double> weights) {
        TwoTuple result = null;
        double beta = 0;
        List<Double> measures = new LinkedList<Double>();
        FuzzySet domain = null;

        for(Valuation valuation : valuations) {
            Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

            if (domain == null) {
                domain = (FuzzySet) valuation.getDomain();
            } else if (!domain.equals(valuation.getDomain())) {
                throw new IllegalArgumentException("Invalid domain");
            }
            measures.add(((TwoTuple) valuation).calculateInverseDelta());
        }

        if (domain != null) {
            int size = measures.size();
            for (int i = 0; i < size; i++) {
                beta += weights.get(i) / measures.get(i);
            }
            beta = 1d / beta;
            result = (TwoTuple) valuations.get(0).clone();
            result.calculateDelta(beta);
        }
        return result;
    }
```

Figura 4.38 Fórmula del operador de agregación media armónica ponderada para el modelo 2-Tuplas.

Media geométrica ponderada

```
public static Valuation aggregate(List<Valuation> valuations, List<Double> weights) {
    TwoTuple result = null;
    double beta = 1;
    List<Double> measures = new LinkedList<Double>();
    FuzzySet domain = null;

    for(Valuation valuation : valuations) {
        Validator.notIllegalElementType(valuation, new String[] { TwoTuple.class.toString() });

        if (domain == null) {
            domain = (FuzzySet) valuation.getDomain();
        } else if (!domain.equals(valuation.getDomain())) {
            throw new IllegalArgumentException("Invalid domain");
        }
        measures.add(((TwoTuple) valuation).calculateInverseDelta());
    }

    if (domain != null) {
        int size = measures.size();
        for (int i = 0; i < size; i++) {
            beta *= Math.pow(weights.get(i), measures.get(i));
        }

        result = (TwoTuple) valuations.get(0).clone();
        result.calculateDelta(beta);
    }
    return result;
}
```

Figura 4.39 Fórmula del operador de agregación media geométrica ponderada para el modelo 2-Tuplas.

4.7. Pruebas y verificación del software

Por último para acabar con el proceso de ingeniería del software se deben verificar que los pasos anteriores se han seguido correctamente. Para ello se realizan varias pruebas que verifican que los requerimientos funcionales planteados anteriormente son correctos y no contienen errores.

4.7.1. Casos de test

Para realizar las pruebas se diseñan una serie de test que prueban el correcto funcionamiento del software.

- **Test 1: Creación de experto**

Requisito: RF01
Acción:
1. El usuario solicita la opción de crear un nuevo experto.

Checkpoint 1:
2. El sistema permite mediante una ventana la inserción de los datos del nuevo experto.

Acción:
3. El usuario introduce los datos del experto y solicita la opción de añadirlo al problema.

Checkpoint 2:

4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el experto se ha añadido correctamente.

- **Test 2: Creación de alternativa**

Requisito: RF02
Acción:

1. El usuario solicita la opción de crear una nueva alternativa.

Checkpoint 3:

2. El sistema permite mediante una ventana la inserción de los datos de la nueva alternativa.

Acción:

3. El usuario introduce los datos de la alternativa y solicita la opción de añadirlo al problema.

Checkpoint 4:

4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que la alternativa se ha añadido correctamente.

- **Test 3: Creación de criterio**

Requisito: RF03
Acción:

1. El usuario solicita la opción de crear un nuevo criterio.

Checkpoint 5:

2. El sistema permite mediante una ventana la inserción de los datos del nuevo criterio.

Acción:

3. El usuario introduce los datos del criterio y solicita la opción de añadirlo al problema.

Checkpoint 6:

4. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el criterio se ha añadido correctamente.

- **Test 4: Creación de dominio de expresión**

Requisito: RF04
Acción:

1. El usuario solicita la opción de crear un nuevo domino de expresión al problema.

Checkpoint 7:

2. El sistema permite mediante una ventana para insertar el nuevo dominio y se indicarán cuáles son los dominios que pueden ser seleccionados.

Acción:

3. El usuario selecciona el dominio de expresión que desea entre las diferentes opciones existentes y selecciona la opción de introducir valores.

Checkpoint 8:

4. El sistema permite la inserción mediante una ventana de los valores del dominio de expresión elegido.

Acción:

5. El usuario introduce los nuevos valores y selecciona la opción de añadir dicho dominio al sistema.

Checkpoint 9:

6. El sistema almacena los nuevos datos introduciéndolos en el problema y muestra al usuario una confirmación de que el nuevo dominio se ha añadido correctamente.

- **Test 5: Asignación de dominio de expresión**

Requisito: RF05

Acción:

1. El usuario solicita la opción de asignar un dominio de expresión a alguno de los elementos del sistema.

Checkpoint 10:

2. El sistema permite la selección de un dominio de expresión mediante una ventana, así como, el elemento al que se desea asignar.

Acción:

3. El usuario selecciona el elemento o elementos a los que desea asignar el dominio que también tendrá que seleccionar y selecciona la opción añadir asignación.

Checkpoint 11:

4. El sistema realiza la asignación correspondiente a los datos introducidos por el usuario y muestra al usuario una confirmación de que se ha realizado correctamente.

- **Test 6: Valoración de alternativas**

Requisito: RF06

Acción:

1. El usuario mediante una ventana realiza las valoraciones de las diferentes alternativas.

Checkpoint 12:

2. El sistema muestra la información de las distintas alternativas, así como, las valoraciones de las mismas si es que están valoradas.

Acción:

3. El usuario selecciona la alternativa que desea valorar.

Checkpoint 13:

4. El sistema permite la inserción mediante una ventana de la valoración de la alternativa seleccionada.

Acción:

5. El usuario realiza la inserción de la valoración de la alternativa seleccionada y confirma la acción.

Checkpoint 14:

6. El sistema asigna la valoración insertada a la alternativa seleccionada y muestra una confirmación de que la acción se ha realizado correctamente.

- **Test 7: Selección del método de resolución**

Requisito: RF07

Acción:

1. El usuario mediante una ventana seleccionar resolver el problema que está abierto en ese momento.

Checkpoint 15:

2. El sistema permite al usuario mediante una ventana seleccionar el método de resolución que desee aplicar para resolver el problema.

Acción

3. El usuario selecciona el método de resolución que desea aplicar para resolver el problema.

Checkpoint 16:

4. El sistema muestra la información sobre el método de resolución seleccionado y habilita las nuevas ventanas adaptadas a cada uno de los métodos diferentes que se pueden seleccionar.

- **Test 8: Selección del operador de agregación sin peso**

Requisito: RF08

Acción:

1. El usuario selecciona un operador de agregación para expertos que no requiere pesos para obtener la solución.

Checkpoint 17:

2. El sistema valida los datos introducidos y realiza la agregación mostrando al usuario el resultado de la misma (E-1).

- **Test 9: Selección del operador de agregación con peso**

Requisito: RF09

Acción:

1. El usuario selecciona un operador de agregación para expertos que requiere pesos para obtener la solución.

Checkpoint 18:

2. El sistema solicita al usuario mediante una ventana los pesos necesarios para poder realizar la agregación.

Acción:

3. El usuario introduce los valores de los distintos pesos, y selecciona la opción para realizar la agregación.

Checkpoint 19:

4. El sistema valida los datos introducidos y realiza la agregación mostrando al usuario el resultado de la misma.

Test	Checkpoint	Resultado
Test 1	Checkpoint 1	OK
	Checkpoint 2	OK
Test 2	Checkpoint 3	OK
	Checkpoint 4	OK
Test 3	Checkpoint 5	OK
	Checkpoint 6	OK
Test 4	Checkpoint 7	OK
	Checkpoint 8	OK
	Checkpoint 9	OK
Test 5	Checkpoint 10	OK
	Checkpoint 11	OK
Test 6	Checkpoint 12	OK
	Checkpoint 13	OK
	Checkpoint 14	OK
Test 7	Checkpoint 15	OK
	Checkpoint 16	OK
Test 8	Checkpoint 17	OK
Test 9	Checkpoint 18	OK
	Checkpoint 19	OK

Tabla 4.1 Resultado de los casos de test.

Capítulo 5: Conclusiones

Los seres humanos en nuestro día a día tenemos que tomar decisiones de diversa índole, en ocasiones estás decisiones tiene gran importancia por lo cual no es aconsejable tomar una decisión a la ligera, además hay muchos casos donde la información que tenemos de nuestro problema es incierta o insuficiente y necesitamos conocimiento elevado sobre el tema del problema para poder tomar una decisión óptima. Para ello existen herramientas que nos ayudan en la toma de decisiones como son los sistemas de soporte a la decisión. En este punto es donde entra Flintstones que es un sistema de soporte a la decisión.

Este proyecto surgió para extender la funcionalidad de Flintstones y dotarlo con un mayor repertorio de operadores de agregación para así hacer que la herramienta Flintstones tenga una funcionalidad más completa.

La principal dificultad de este proyecto ha sido precisamente entender tanto la teoría de toma de decisiones, la cual es compleja y densa, así como la tecnología de RCP, puesto que la he tenido que aprender.

Antes de empezar a realizar este proyecto no tenía mucha idea sobre el proceso de toma de decisiones y no era consciente de la dificultad que tiene sobre todo cuando afrontamos problemas con ambiente de incertidumbre.

Por otro lado tenemos la herramienta Flintstones, dicha herramienta he tenido que aprender a usarla, así como también he tenido que conocer su estructura interna.

Ahora también veo la utilidad de usar sistemas de soporte a la decisión como Flintstones, ya que nos ayuda a tomar la decisión de problemas con ambiente de incertidumbre y siempre teniendo en cuenta que para usar herramientas de este tipo se tiene que tener un conocimiento elevado sobre el área donde nuestro problema se encuentre, porque en última instancia el que toma la decisión es una persona o un grupo de personas en consenso.

En definitiva en este proyecto ha sido más complicada la parte de aprendizaje teórico que la de implementación de los operadores, aunque que haya sido una tecnología completamente nueva para mí.

Por último, concluir que desde mi punto de vista se ha conseguido los propósitos para los que se estableció este trabajo fin de grado y se han cumplido con todos los pasos necesarios para la realización de un proyecto de esta magnitud.

Capítulo 6: Bibliografía

1. http://sinbad2.ujaen.es/?q=es/linea/1448-toma-de-decisiones

2. D. Bouyssou, T. Marchant, M. Pirlot, P. Perny, and A. Tsoukiàs. Evaluation and Decision Models: A critical perspective.
Kluwer Academic Publishers, 2000.

3. Francisco Javier Pulgar Rubio. Implementación del método Topsis y nuevos operadores de agregación para la herramienta "Flintstones". 2013.

4. Luis Martínez, Rosa M. Rodríguez, Francisco Herrera. The 2-tuple Linguistic Model Computing with Words in Decision Making.

5. C. Zopounidis and M.Doumpos. Intelligent Decision Aiding Systems Based On Multiple Criteria For Financial Engineering.
Kluwer Academic Publishers, 2000.

6. R.R. Yager. A new methodology for ordinal multiobjective decision based on fuzzy sets.
Decision Science, 12:589–600, 1981.

7. R.R. Yager. Non-numeric multi-criteria multi-person decision making.
Group Decision and Negotiation, 2:81–93, 1993.

8. P.P. Bonissone. A fuzzy sets based linguistic approach: Theory and applications, pages 329–339.
Approximate Reasoning in Decision Analysis, North-Holland, 1982.

9. L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning.
Information Sciences, Part I, II, III, 8, 8, 9: 199–249,301–357, 43–80, 1975.

10. G.J. Klir and B. Yuan. Fuzzy sets and fuzzy logic: Theory and Applications. Prentice-Hall
PTR, 1995.

11. L.A. Zadeh. Fuzzy sets.
Information and Control, 8:338–353, 1965.

12. L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning. Part I.
Information Sciences, 8:199–249, 1975.

13. L. Martínez. Sensory evaluation based on linguistic decision analysis.
International Journal of Approximate Reasoning, 44(2):148–164, 2007.

14. R. R. Yager. An approach to ordinal decision making.
International Journal of Approximate Reasoning, 12(3):237–261, 1995.

15. R.M. Rodríguez, L. Martínez, and F. Herrera. Hesitant fuzzy linguistic term sets for decisionmaking.
IEEE Transactions on Fuzzy Systems, 20(1):109–119, 2012.

16. F. Herrera, E. Herrera-Viedma, and L. Martínez. A fuzzy linguistic methodology to deal with unbalanced linguistic term sets.
IEEE Transactions on Fuzzy Systems, 16(2):354–370, 2008.

17. V. Torra. Negation function based semantics for ordered linguistic labels.
International Journal of Intelligent Systems, 11:975–988, 1996.

18. M. Delgado, F. Herrera, E. Herrera-Viedma, and L. Martínez. Combining numerical and linguistic information in group decision making.
Information Sciences, 107(1–4):177–194, 1998.

19. P.P. Bonissone and K.S. Decker. Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-Off Precision and Complexity.
In L.H. Kanal and J.F. Lemmer, Editors., Uncertainty in Artificial Intelligence. North-Holland, 1986.

20. M. Delgado, M.A. Vila, and W. Voxman. On a canonical representation of fuzzy numbers.
Fuzzy Sets and Systems, 94:125–135, 98.

21. D. Dubois and H. Prade. The three semantics of fuzzy sets.
Fuzzy Sets and Systems, 90:141–150, 1997.

22. R. Bellman, L. Kalaba, and L.A. Zadeh. Abstraction and pattern classification.
Journal of Mathematical Analysis and Applications, 13(1):1–7, 1966.

23. L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility.
Fuzzy Sets and Systems, 1:3–28, 1978.

24. L.A. Zadeh. A theory of approximate reasoning, pages 149–194. In: Machine Intelligence.
Elsevier, 1979.

25. R.E. Bellman and L.A. Zadeh. Decision making in a fuzzy environment.
Management Science, 4(17), 1970.

26. M. Roubens. Fuzzy sets and decision analysis.
Fuzzy Sets and Systems, 90:199–206, 1997.

27. F. Herrera and E. Herrera-Viedma. Linguistic decision analysis: Steps for solving decision problems under linguistic information. Fuzzy
Sets and Systems, 115(1):67–82, 2000.

28. L.A. Zadeh. Fuzzy logic = computing with words.
IEEE Transactions on Fuzzy Systems, 4(2):103–111, 1996.

29. L.A. Zadeh. From computing with numbers to computing with words - from manipulation of measurements to manipulation of perceptions.
IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 46(1):105–119, 1999

30. J.M. Mendel, L.A. Zadeh, E. Trillas, R.R. Yager, J. Lawry, H. Hagras, and S. Guadarrama. What computing with words means tome: Discussion forum.
IEEE Computational Intelligence Magazine, 5(1):20–26, 2010.

31. L.A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes.
IEEE Transactions on Systems, Man and Cybernetics, SMC-3(1):28–44, 1973.

32. T. Evangelos. Multi-criteria decision making methods: a comparative study.
Kluwer Academic Publishers, Dordrecht, 2000.

33. M. Tong and P.P. Bonissone. A linguistic approach to decision making with fuzzy sets.
IEEE Transactions on Systems, Man and Cybernetics, 10:716–723, 1980.

34. K.S. Schmucker. Fuzzy Sets, Natural Language Computations, and Risk Analysis.
Computer
Science Press, Rockville, MD, 1984.

35. R.R. Yager. Computing with words and information/intelligent systems 2:applications, chapter Approximate reasoning as a basis for computing with words, pages 50–77.
Physica Verlag,1999.

36. R.R. Yager. On the retranslation process in Zadeh's paradigm of computing with words. Systems, Man, and Cybernetics, Part B: Cybernetics,
IEEE Transactions on, 34(2):1184–1195, 2004.

37. F. Herrera and L. Martínez. A 2-tuple fuzzy linguistic representation model for computing with words.
IEEE Transactions on Fuzzy Systems, 8(6):746–752, 2000.

38. J. Lawry. A framework for linguistic modelling.
Artificial Intelligence, 155(1–2):1–39, 2004.

39. Y. Tang and J. Zheng. Linguistic modelling based on semantic similarity relation among linguistic labels.
Fuzzy Sets and Systems, 157(12):1662–1673, 2006.

40. I.B. Türk¸sen. Meta-linguistic axioms as a foundation for computing with words.
Information Sciences, 177(2):332–359, 2007.

41. M. Ying. A formal model of computing with words.
IEEE Transactions on Fuzzy Systems, 10(5):640–652, 2002.

42. K. Anagnostopoulos, H. Doukas, and J. Psarras. A linguistic multicriteria analysis system combining fuzzy sets theory, ideal and anti-ideal points for location site selection.
Expert Systems with Applications, 35(4):2041–2048, 2008.

43. G. Fu. A fuzzy optimization method for multicriteria decision making: An application to reservoir flood control operation.
Expert Systems with Applications, 34(1):145–149, 2008.

44. M. Delgado, J.L. Verdegay, and M.A Vila. On aggregation operations of linguistic labels.
International Journal of Intelligent Systems, 8(3):351–370, 1993.

45. F. Herrera and L. Martínez. A model based on linguistic 2-tuples for dealing with multigranularity hierarchical linguistic contexts in multiexpert decision-making.
IEEE Transactions on Systems, Man and Cybernetics. Part B: Cybernetics, 31(2):227-234, 2001.

46. F. Herrera and L. Martínez. A fuzzy linguistic methodology to deal with unbalanced linguistic term sets.
IEEE Transactions on Fuzzy Systems, 16(2):354-370, 2008.

47. F. Herrera and L. Martínez. Managing non-homogeneous information in group decision making.
European Journal of Operational Research, 166(1):115-132, 2005.

48. Rosa M. Rodríguez. Un Nuevo Modelo para Procesos de Computación con Palabras en Toma de Decisión Lingüística.

49. R.R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making.
IEEE Trans. on Systems, Man, and Cybernetics, 18:183-190, 1988.

50. G.W. Wei. Some harmonic aggregation operators with 2-tuple linguistic assessment information and their application to multiple attribute group decision making.
International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 19(6):977–998, 2011.

51. R.R. Yager. The power average operator. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 31(6):724–731, 2001.

52. G. Choquet. Theory of capacities, volume 5, pages 131–295.
Annales de l'institut Fourier, 1953.

53. M. Grabisch. The application of fuzzy integrals in multicriteria decision making.
European Journal of Operational Research, 89(3):445–456, 1996.

54. Y.H. Lin, P.C. Lee, and H.I. Ting. Dynamic multi-attribute decision making model with grey number evaluations.
Expert Systems with Applications, 35(4):1638–1644, 2008.

55. X. Liu and W. Yang. A new multi-period linguistic aggregation operator and its application to financial product selection, 2013.

56. S. Aydin and C. Kahraman. A new fuzzy analytic hierarchy process and its application to vendor selection problem. Journal of Multiple-Valued Logic and Soft Computing, 20(3–4): 353–371, 2013.

57. M. Espinilla, R. de Andrés, F.J. Martínez, and L. Martínez. A 360-degree performance appraisal model dealing with heterogeneous information and dependent criteria.
Information Sciences, 222:459–471, 2013.

58. M. Espinilla, J. Liu, and L. Martínez. An extended hierarchical linguistic model for decisionmaking problems.
Computational Intelligence, 27(3):489–512, 2011.

59. M. Espinilla, I. Palomares, L. Martínez, and D. Ruan. A comparative study of heterogeneous decision analysis approaches applied to sustainable energy evaluation. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 20(supp01):159–174, 2012.

60. F.J. Estrella, M. Espinilla, F. Herrera, and L. Martínez. FLINTSTONES: a fuzzy linguistic decision tools enhancement suite based on the 2-tuple linguistic model and extensions.
Information Sciences, 280:152–170, 2014.

61. F.J. Estrella,M. Espinilla, and L. Martínez. Fuzzy linguistic olive oil sensory evaluation model based on unbalanced linguistic scales. Journal of Multiple-Valued Logic and Soft Computing, 22:501–520, 2014.

62. S. Gramajo and L. Martínez. A linguistic decision support model for QoS priorities in networking.
Knowledge-based Systems, 32(1):65–75, 2012.

63. F. Herrera, E. Herrera-Viedma, and L. Martínez. A fuzzy linguistic methodology to deal with unbalanced linguistic term sets.
IEEE Transactions on Fuzzy Systems, 16(2):354–370, 2008.

64. F. Herrera and L. Martínez. A model based on linguistic 2-tuples for dealing with multigranular hierarchical linguistic context in multi-expert decision making.
IEEE Transactions on Systems, Man, And Cybernetics - Part B: Cybernetics, 31(2):227–234, 2001.

65. E. Herrera-Viedma, F.J. Cabrerizo, J. Kacprzyk, and W. Pedrycz. A review of soft consensus models in a fuzzy environment.
Information Fusion, 17(0):4–13, 2014.

66. M. Kabak. A fuzzy DEMATEL-ANP based multi criteria decision making approach for personnel selection. Journal of Multiple-Valued Logic and Soft Computing, 20(5–6):571–593,
2013.

67. L. Martínez, M. Espinilla, J. Liu, L.G. Pérez, and P.J. Sánchez. An evaluation model with unbalanced linguistic information: Applied to olive oil sensory evaluation. Journal of Multiple-Valued Logic and Soft Computing, 15(2–3):229–251, 2009.

68. L. Martínez, M. Espinilla, and L.G. Pérez. A linguistic multigranular sensory evaluation model for olive oil. International Journal of Computational Intelligence Systems, 1(2):148–158, 2008.

69. I. Palomares, J. Liu, Y. Xu, and L. Martínez. Modelling experts' attitudes in group decision making.
Soft Computing, 16(10):1755–1766, 2012.

70. R.M. Rodríguez, M. Espinilla, P.J. Sánchez, and L. Martínez. Using linguistic incomplete preference relations to cold start recommendations.
Internet Research, 20(3):296–315, 2010.

71. P. J. Sánchez, L. Martínez, C. García-Martínez, F. Herrera, and E. Herrera-Viedma. A fuzzy model to evaluate the suitability of installing an enterprise resource planning system.
Information Sciences, 179(14):2333–2341, 2009.

72. G. Zhang, Y. Dong, and Y. Xu. Consistency and consensus measures for linguistic preference relations based on distribution assessments.
Information Fusion, 17(0):46–55, 2014.

73. Y. Zhang and Z.P. Fan. Uncertain linguistic multiple attribute group decision making approach and its application to software project selection.
Journal of Software, 6(4):662–669, 2011.

74. Francisco José Quesada Real. Sistema de soporte al consenso en problemas de toma de decisión en grupo basados en agentes inteligentes.

75. D. Ettema, T. Gärling, L. Eriksson, M. Friman, L.E. Olsson, and S. Fujii. Satisfaction with travel and subjective well-being: Development and test of a measurement tool. Transportation
Research Part F: Traffic Psychology and Behaviour, 14(3):167–175, 2011.

76. C. Fletcher. Performance appraisal and management: The developing.
Journal of Occupational and Organization Psychology, 74:473–487, 2001.

77. Y. Chen, X. Zeng, M. Happiette, P. Bruniaux, R. Ng, and W. Yu. Optimisation of garment design using fuzzy logic and sensory evaluation techniques.
Engineering applications of artificial intelligence, 22(2):272–282, 2009.

78. Z. Chen and D. Ben-Arieh. On the fusion of multi-granularity linguistic label sets in group decision making.
Computers and Industrial Engineering, 51(3):526–541, 2006.

79. L. Martínez, J. Liu, and J. B. Yang. A fuzzy model for design evaluation based on multiple criteria analysis in engineering systems.
International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 14(3):317–336, 2006.

80. Y. Chen, X. Zeng, M. Happiette, P. Bruniaux, R. Ng, and W. Yu. Optimisation of garment design using fuzzy logic and sensory evaluation techniques.
Engineering Applications of Artificial Intelligence, 22(2):272–282, 2009.

81. V.N. Huynh and Y. Nakamori. A satisfactory-oriented approach to multiexpert decisionmaking with linguistic assessments.
IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 35(2):184–196, 2005.

82. J. Wang, J.B. Yang, and P. Sen. Multi-person and multi-attribute design evaluations using evidential reasoning based on subjective safety and cost analyses.
Reliability Engineering and System Safety, 52(2):113–128, 1996.

83. J. Fodor and M. Roubens. Fuzzy preference modelling and multicriteria decision support.
Kluwer Academic Publishers, Dordrecht, 1994.

84. D. Dubois and H. Prade. Fuzzy Sets and Systems: Theory and Applications.
Kluwer Academic, New York, 1980.

85. G.J. Klir and B. Yuan. Fuzzy sets an fuzzy logic: Theory and Applications.
Prentice-Hall PTR, 1995.

86. After, S.L. Decision support systems: current practice and continuing challenges.
Reading, Mass., Addison-Wesley Pub, 1980.

87. Druzdzel, M. J. and R. R. Flynn. Decision Support Systems.
Encyclopedia of Library and Information Science. A. Kent, Marcel Dekker, Inc., 1999.

88. Pressman, Roger S., "Ingeniería del Software. Un enfoque práctico".
Quinta edición, México: Mc Graw Hill, 2003.

89. Larman, Craig, "UML y Patrones".
Segunda Edición, Prentice Hall, 2003.

90. Oscar Belmonte Fernández, introducción al lenguaje de programación Java. Una guía básica.
Última revisión 2.0. 06/06/2005.

91. http://www.eclipse.org/legal/epl-v10.html

92. http://www.vogella.com/tutorials/OSGI/article.html

93. http://www.vogella.com/tutorials/Eclipse3RCP/article.html

94. L. Alfonso Ureña López y J. Ignacio Gómez Espínola, Tema 5 modelado de la dinámica.
Fundamentos de ingeniería del software, 2011.

95. L. Alfonso Ureña López y J. Ignacio Gómez Espínola, Tema 9 fundamentos del diseño de software.
Fundamentos de ingeniería del software, 2011.

Anexo 1: Creación de un operador de agregación paso a paso

En este anexo se explicará paso a paso como hacer un operador de agregación mediante la creación de un punto de extensión.

Figura A1.1 Eclipse para RCP

Primero necesitamos tener una versión de Eclipse para RCP (Como podemos ver en la Figura A1.1), esta versión la podemos descargar de la página oficial de eclipse http://www.eclipse.org/downloads/eclipse-packages/. Una vez que tengamos Eclipse RCP y tengamos el proyecto de Flintstones abierto, para ello simplemente abriremos y exportaremos el proyecto de Flintstones a Eclipse RCP, podemos empezar a crear el nuevo plug-in, pulsando en la opción "Plug-in Project" como se observa en la Figura A1.2.

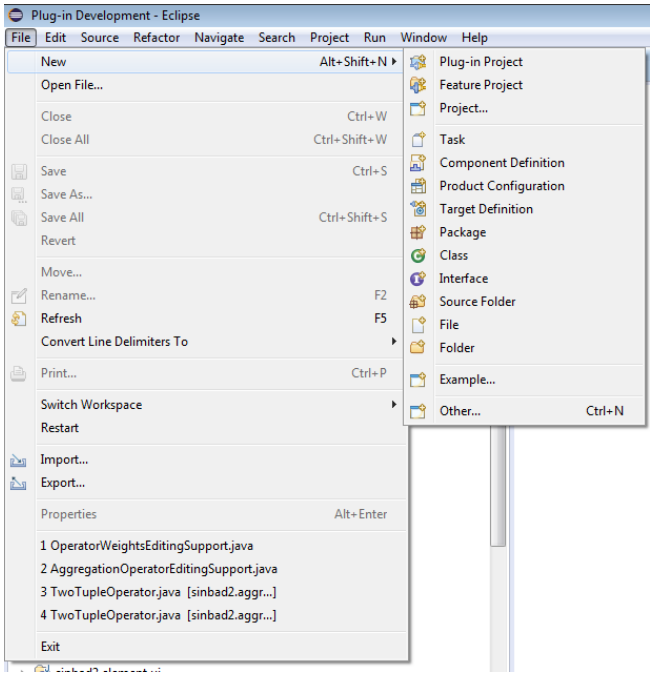

Figura A1.2 Crear plug-in

Así creamos el plug-in para crear el nuevo operador de agregación, a continuación nos aparecerá un asistente que nos guiara para la creación del plug-in como podemos observar en la Figura A1.3.

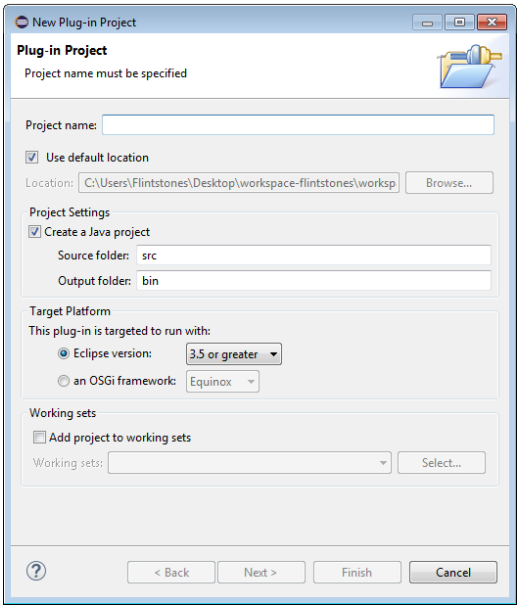

Figura A1.3 Asistente para crear plug-in, nombre del proyecto

Para seguir con la nomenclatura que ya tenemos en Flintstones, cuando creamos un nuevo operador de agregación lo de nominaremos sinbad2.aggregationoperator.x, donde x es el nombre del operador. En la casilla Project name indicaremos el nombre de nuestro operador.

En la siguiente pantalla del asistente deberemos de desmarcar todas las opciones que hay en options, ya que estás opciones no son necesarias, el apartado "options", lo podemos observar en la Figura A1.4, después pulsamos finalizar y ya tendremos creado el nuevo punto de extensión.

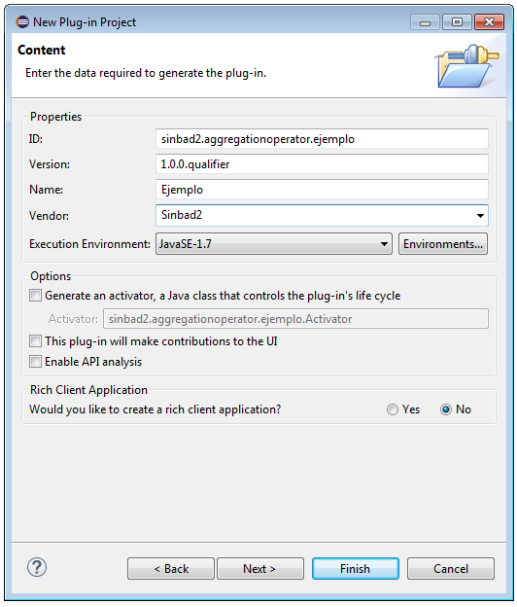

Figura A1.4 Asistente para crear plug-in, opciones

Ya tenemos creado el plug-in, ahora el siguiente paso es configurarlo. Para ello es necesario agregarle las dependencias, que no es otra cosa que indicarle a este nuevo componente cuales son los otros componentes que están relacionados con él.

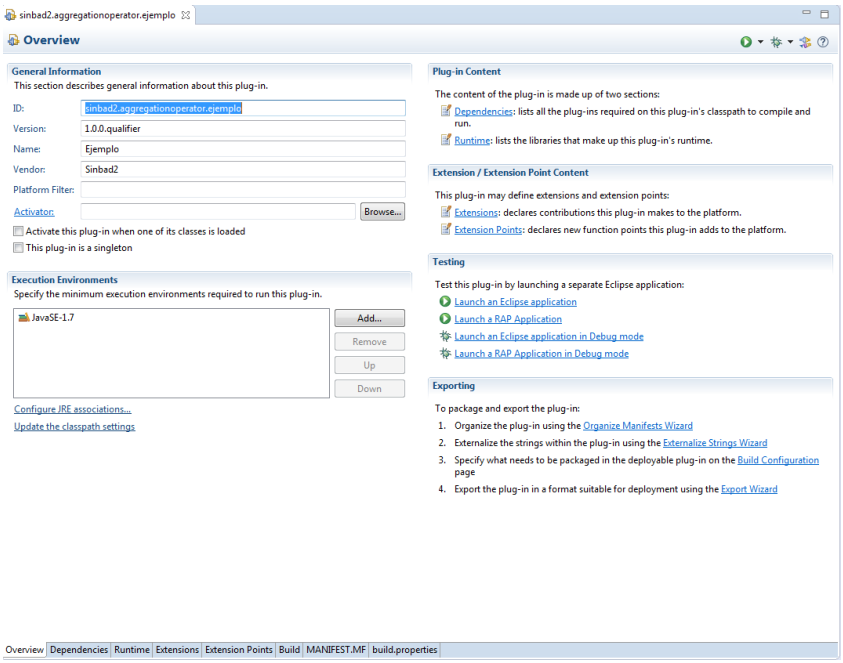

Figura A1.5 Pantalla de configuración del nuevo operador

Las dependencias que necesitaremos para este nuevo operador son las que observamos en la Figura A1.6. Se puede apreciar que depende del componente aggregationoperator, ya que el punto de extensión que define la estructura de un operador de agregación se encuentra en este plug-in y depende también del tipo de valoraciones que acepte el operador.

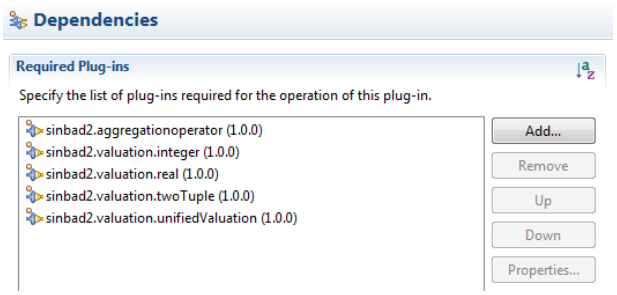

Figura A1.6 Dependencias

Tenemos que tener en cuenta que las dependencias serán diferentes en función del operador de agregación, siendo las de la Figura A1.6 dependencias de un operador de agregación sin peso, sin embargo cuando tenemos operadores de agregación con peso las dependencias son como observamos en la Figura A1.7.

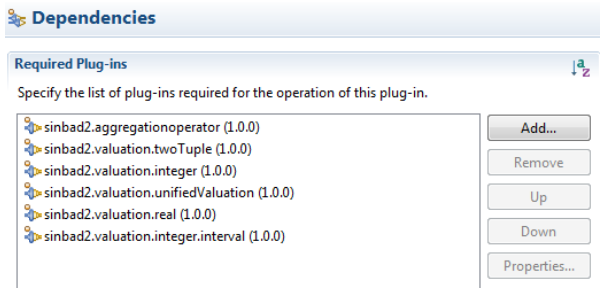

Figura A1.7 Dependencias para operadores de agregación con peso

Después de agregar las dependencias pasamos a la pestaña de extensions y agregamos la extensión flintstones.aggregationoperator como podemos observar en la Figura A1.8. En este punto estamos extendiendo el punto de extensión aggregationoperator. Este punto de extensión tiene una serie de elementos, que son la definición del operador de agregación, si el operador tiene peso o no y por último que tipo de datos soporta. En la Figura A1.8 se puede observar un círculo rojo que indica que en este ejemplo el operador elegido no tiene peso.

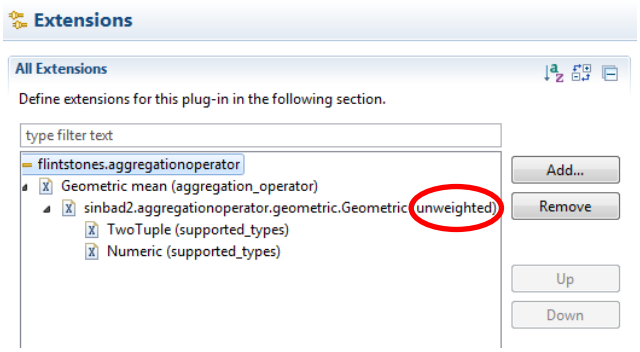

Figura A1.8 Extensions

Una vez agregada la extensión flintstones.aggregationoperator, creamos un aggregation_operator como observamos en la Figura A1.8, donde indicamos un nombre y un id. Esta id será la que nos sirva más adelante para identificar este operador.

Después de crear el aggregation_operator tendremos que indicar si el operador de agregación que estamos creando tiene peso o no, y también deberemos indicarle que tipo de datos soporta nuestro operador, en el ejemplo que observamos en la Figura A1.8, es un operador sin peso y los tipos soportados son numérico y valores 2-Tuplas.

Cuando indicamos si nuestro operador tiene o no peso, en la pantalla de la izquierda aparecerá un enlace con la palabra implementación como observamos en la Figura A1.9.

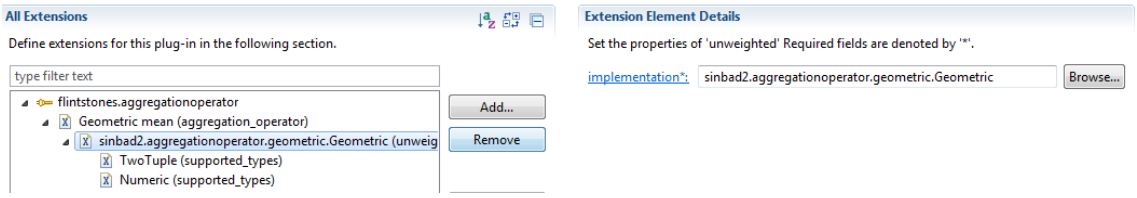

Figura A1.9 Implementación del operador

Las implementaciones que hay que realizar en esta fase ya se mostró en el capítulo 4.6.3.2 en la Figura 4.30 para operadores sin peso, y en las Figuras 4.35 y 4.36 para operadores con peso.

Para que las implementaciones anteriores funcionen debemos crear dentro de la carpeta "src" un package nuevo al que llamaremos sinbad2.aggregarionoperator.x.valuation, donde x es el nombre del operador que estamos implementando, quedando la estructura del nuevo operador como observamos en la Figura A1.10.

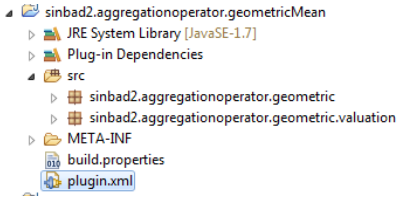

Figura A1.10 Estructura del nuevo operador de agregación

Dentro deberemos implementar la definición del operador en función del tipo de valoraciones que el operador acepte, tendremos tantos ficheros .java como tipos de valoraciones soporte el operador. En la Figura A1.11 podemos observar un ejemplo de un operador que soporta los tipos de valoraciones de entero, real, 2-Tuplas y unificado.

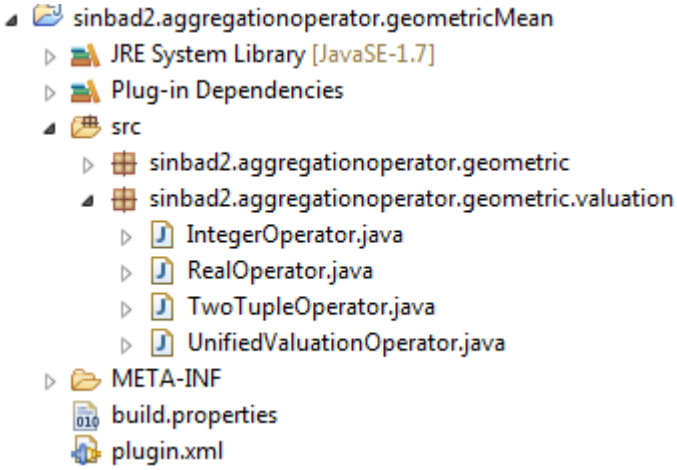

Figura A1.11 Estructura de sinbad2.aggregarionoperator.geometric.valuation

Ahora el siguiente paso es registrar el nuevo plug-in en Flintstones para ello deberemos irnos al componente de sinbad2.RCP y una vez dentro pulsar en flintstones.product como podemos observar en la Figura A1.12.

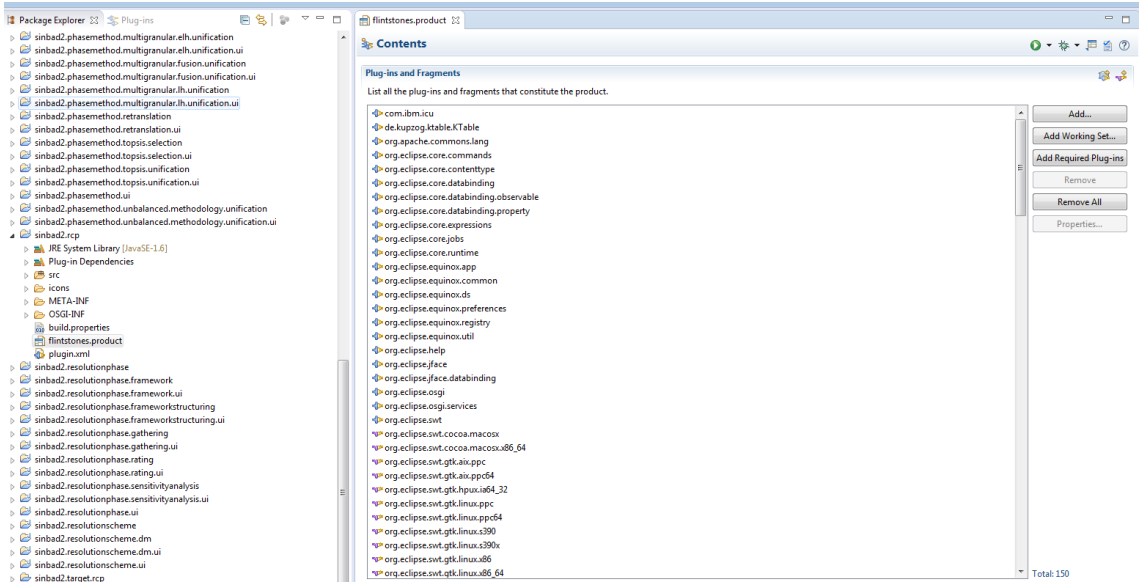

Figura A1.12 flintstones.product

Aquí solo deberemos añadir el nuevo operador que hemos creado para que así Flintstones lo tenga registrado, el último paso que tenemos que realizar es indicarle a la interfaz de Eclipse RCP el nuevo plug-in que ha sido creado y para ello deberemos de irnos a la pestañas run y pulsar run configurations y nos aparecerá una ventana como podemos ver en la Figura A1.13.

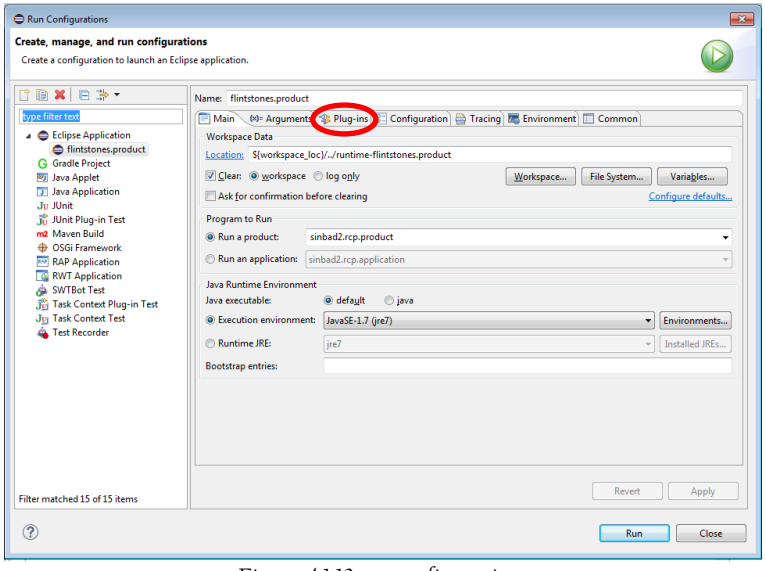

Figura A1.13 run configurations

Seleccionamos la pestaña Plug-ins y buscamos el nuevo plug-in, a continuación lo marcamos y aplicamos los cambios pulsando apply.

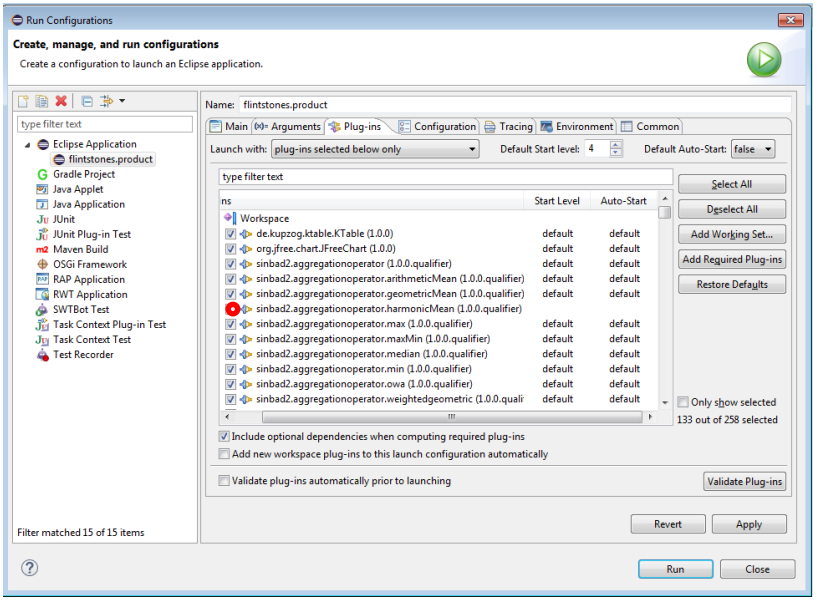

Figura A1.14 agregar el nuevo plug-in a Flintstones

En caso de que el operador que hemos creado sea con peso necesitaremos introducir los pesos cuando seleccionamos este operador, por lo tanto necesitamos introducir estos pesos de alguna manera. Para ello es necesario modificar los siguientes ficheros; AggregationOperatorEditingSupport.java y OperatorWeightsEditingSupport.java, que se encuentran dentro del componente sinbad2.phasemethod.aggregation.iu como observamos en la Figura A1.15. Modificando estos ficheros podemos hacer que al elegir este operador con peso aparezca una nueva pantalla donde introduciremos nuestros pesos.

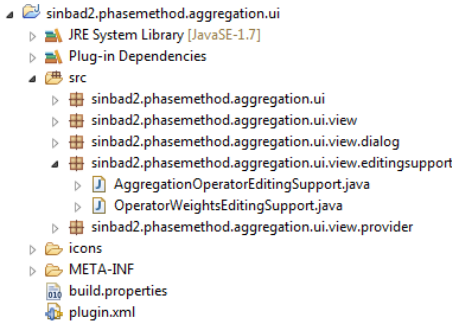

Figura A1.15 sinbad2.phasemethod.aggregation.iu

Dentro del fichero AggregationOperatorEditingSupport.java tendremos que modificar la función setOperator. A continuación en la Figura A1.16 tenemos un ejemplo del código que hay que incluir.

```
if (aggregationOperator.getName().equals("Weighted geometric")) { //$NON-NLS-1$

    ProblemElementsManager elementsManager = ProblemElementsManager.getInstance();
    ProblemElementsSet elementsSet = elementsManager.getActiveElementSet();

    ProblemElement nullElement = null;
    ProblemElement[] secondary;

    WeightsDialog dialog;
    if(elementType.equals("Expert")) {
        secondary = getLeafElements(nullElement, "criterion");
        dialog = new WeightsDialog(Display.getCurrent().getActiveShell(), elementsSet.getAllElementExpertChildren((Expert) element), secondary, null, QuantifiersDialog.SIMPLE, elementType, elementId);
    } else {
        secondary = getLeafElements(nullElement, "expert");
        dialog = new WeightsDialog(Display.getCurrent().getActiveShell(), elementsSet.getAllElementCriterionSubcriteria((Criterion) element), secondary, null, QuantifiersDialog.SIMPLE, elementType, elementId);
    }

    int exitValue = dialog.open();
    if(exitValue == WeightsDialog.SAVE) {
        _mapWeights = dialog.getWeights();
        _weights = null;
        operator = aggregationOperator;
    } else if(exitValue == QuantifiersDialog.CANCEL_ALL) {
        _mapWeights = null;
        _weights = null;
        _abort = true;
    }
}
```

Figura A1.16 Fragmento de código para modificar la función setOperator.

Dentro del fichero OperatorWeightsEditingSupport.java deberemos modificar la función setValue. A continuación en la Figura A1.17 tenemos un ejemplo del código que hay que incluir.

```
else if(operator.getName().equals("Weighted geometric")) { //$NON-NLS-1$

    ProblemElement nullElement = null;
    ProblemElement[] secondary;

    ProblemElementsManager elementsManager = ProblemElementsManager.getInstance();
    ProblemElementsSet elementsSet = elementsManager.getActiveElementSet();

    WeightsDialog dialog;
    if(elementType.equals("Expert")) {
        secondary = getLeafElements(nullElement, "criterion");
        dialog = new WeightsDialog(Display.getCurrent().getActiveShell(), elementsSet.getAllElementExpertChildren((Expert) problemElement), secondary, mapWeights, QuantifiersDialog.SIMPLE, elementType, elementId);
    } else {
        secondary = getLeafElements(nullElement, "expert");
        dialog = new WeightsDialog(Display.getCurrent().getActiveShell(), elementsSet.getAllElementCriterionSubcriteria((Criterion) problemElement), secondary, mapWeights, QuantifiersDialog.SIMPLE, elementType, elementId);
    }

    if(dialog.open() == QuantifiersDialog.SAVE) {
        mapWeights = dialog.getWeights();
        if (AggregationPhase.EXPERTS.equals(_type)) {
            _aggregationPhase.setExpertOperator(problemElement, operator, mapWeights);
        } else {
            _aggregationPhase.setCriterionOperator(problemElement, operator, mapWeights);
        }
    }
}
```

Figura A1.17 Fragmento de código para modificar la función setValue.

Anexo 2: Instalación de Flintstones y ejemplo de funcionamiento de un operador de agregación

Para usar Flintstones en nuestro equipo solo necesitaremos tener el ejecutable de la aplicación. Por lo tanto no necesitaremos instalar nada en nuestro equipo.

Junto con esta documentación se adjunta un fichero comprimido con el nombre "Flintstones", descomprimiremos este archivo. Cuando este descomprimido el fichero tendremos un carpeta llamada "Flintstones" y dentro de esta carpeta tendremos otro archivo comprimido con el nombre "eclipse", el cual descomprimiremos en esa misma carpeta, al descomprimir este último fichero tendremos una carpeta llamada "eclipse" y dentro encontraremos el ejecutable de Flintstones, cuyo nombre es "flintstones".

Además tenemos que tener en cuenta la versión de java, ya que Flintstones funcionará a partir de la versión 8 de java y la actualización 121 como observamos en la Figura A2.1.

Figura A2.1 Versión de Java.

Dentro de la carpeta de Flintstones tendremos tres carpetas que son:

- eclipse, donde encontramos el ejecutable de Flintstones.
- ejemplo, donde encontramos un ejemplo para probar los operadores de agregación, para así no tener que introducir los datos manualmente.
- repository, donde encontramos archivos java para el funcionamiento de Flintstones.

Así que para iniciar Flintstones bastará con hacer doble click sobre el ejecutable que se encuentra dentro de la carpeta "eclipse", y ya tendremos Flintstones funcionando.

Cuando iniciamos Flintstones la pantalla inicial es la que corresponde con el marco de evaluación, donde se establecen los diferentes elementos que componen un problema de toma de decisión como observamos en la Figura A2.2.

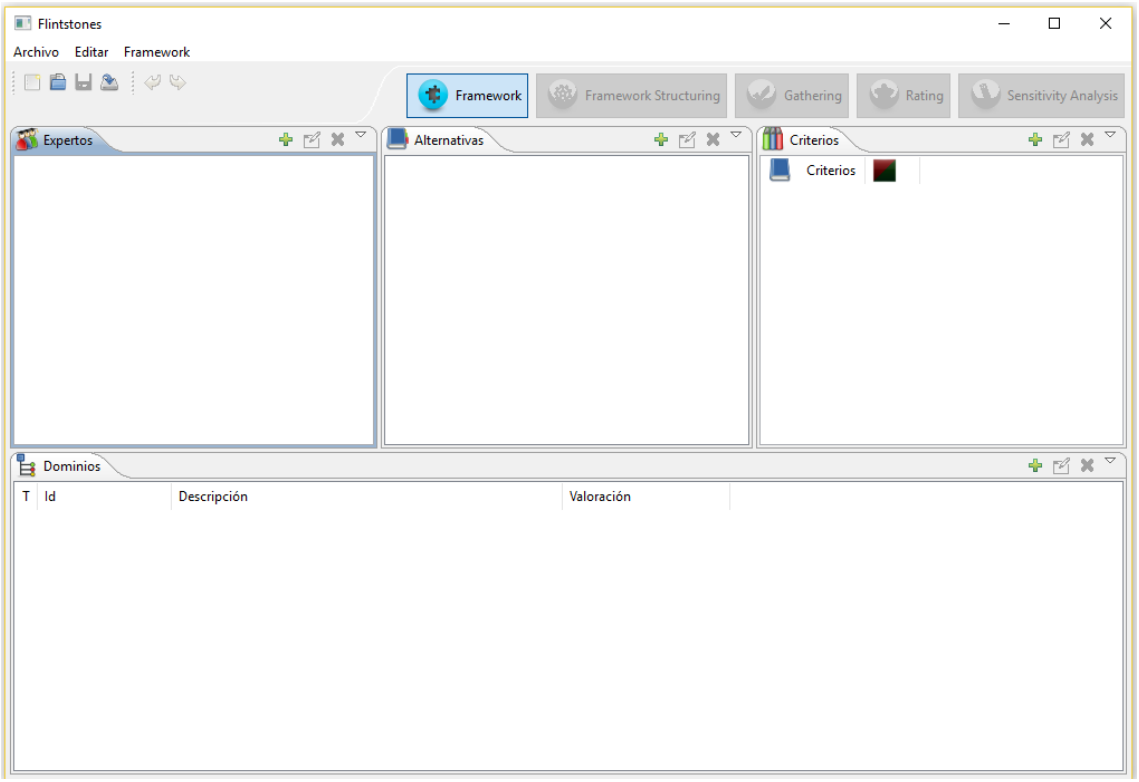

Figura A2.2 Pantalla inicial de Flintstones.

A continuación agregaremos los expertos, alternativas y criterios dentro del marco de evaluación, para ello pulsaremos sobre el icono  e introduciremos los datos.

También podemos optar por la opción de cargar un ejemplo, de esta forma podremos probar el funcionamiento de un operador de agregación más rápido que si se introducen los datos de forma manual.

El ejemplo que usaremos se encuentra dentro de la carpeta llamada "ejemplo".

Para cargar un problema simplemente deberemos pulsar en la opción Archivo y después Abrir como se muestra en la Figura A2.3.

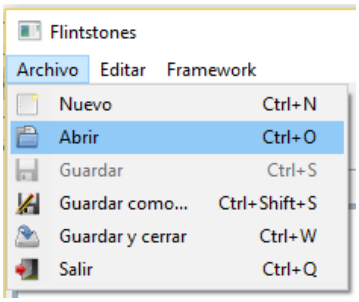

Figura A2.3 Abrir un ejemplo en Flintstones.

Después dentro de la carpeta "Flintstones" encontramos la carpeta "ejemplo" y en esta carpeta tendremos el ejemplo, como se muestra en la Figura A2.4.

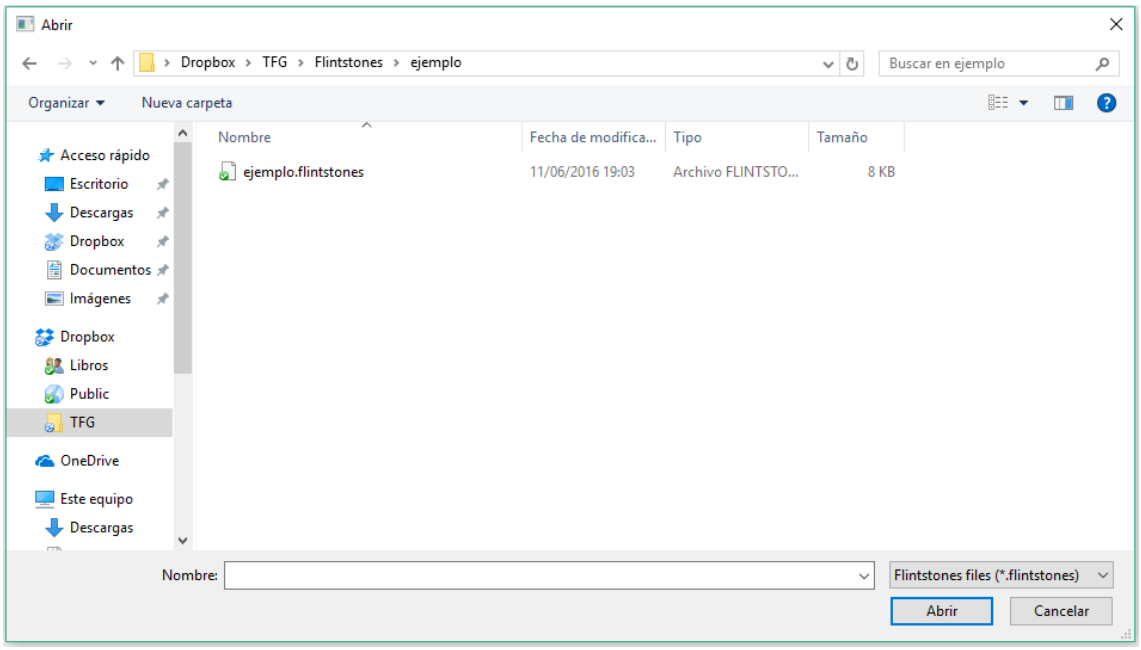

Figura A2.4 Cargar un ejemplo en Flintstones

Como podemos observar en la Figura A2.5 los datos del problema ya están cargados.

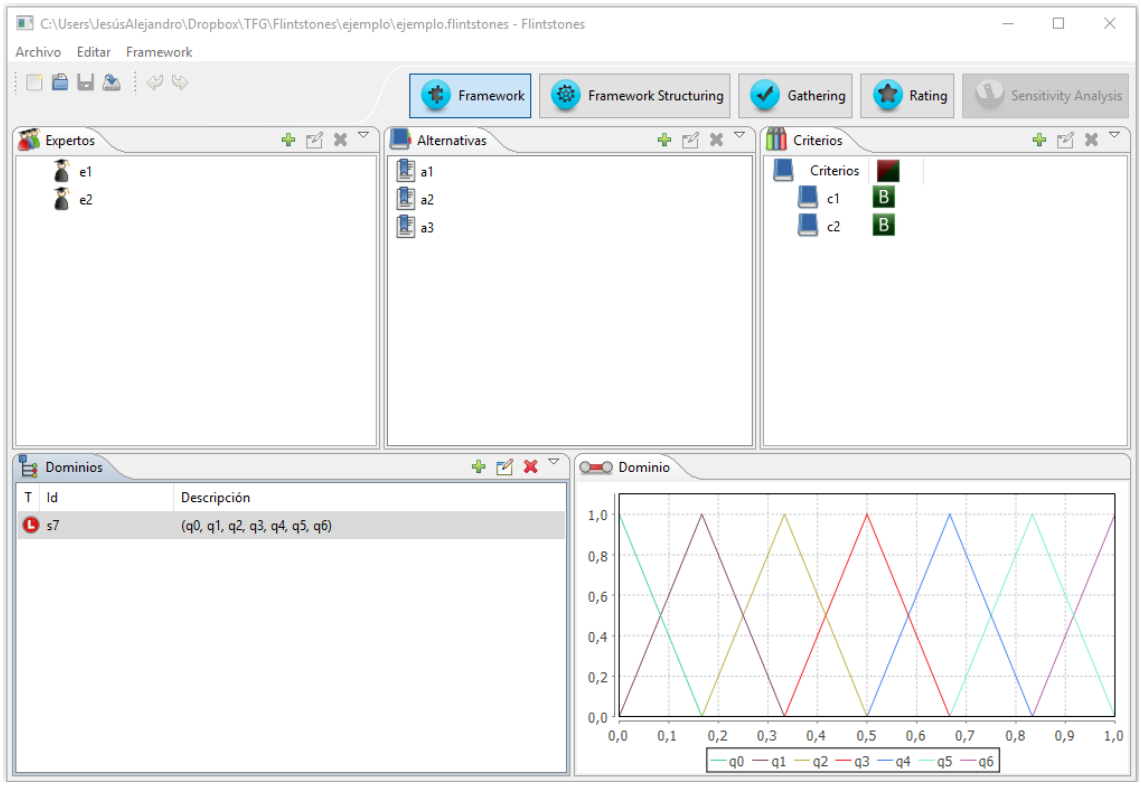

Figura A2.5 Marco de evaluación

El siguiente paso ya están definidos los dominios de expresión que utilizan en las alternativas como se muestra en la Figura A2.6 dentro de la estructura del marco de evaluación.

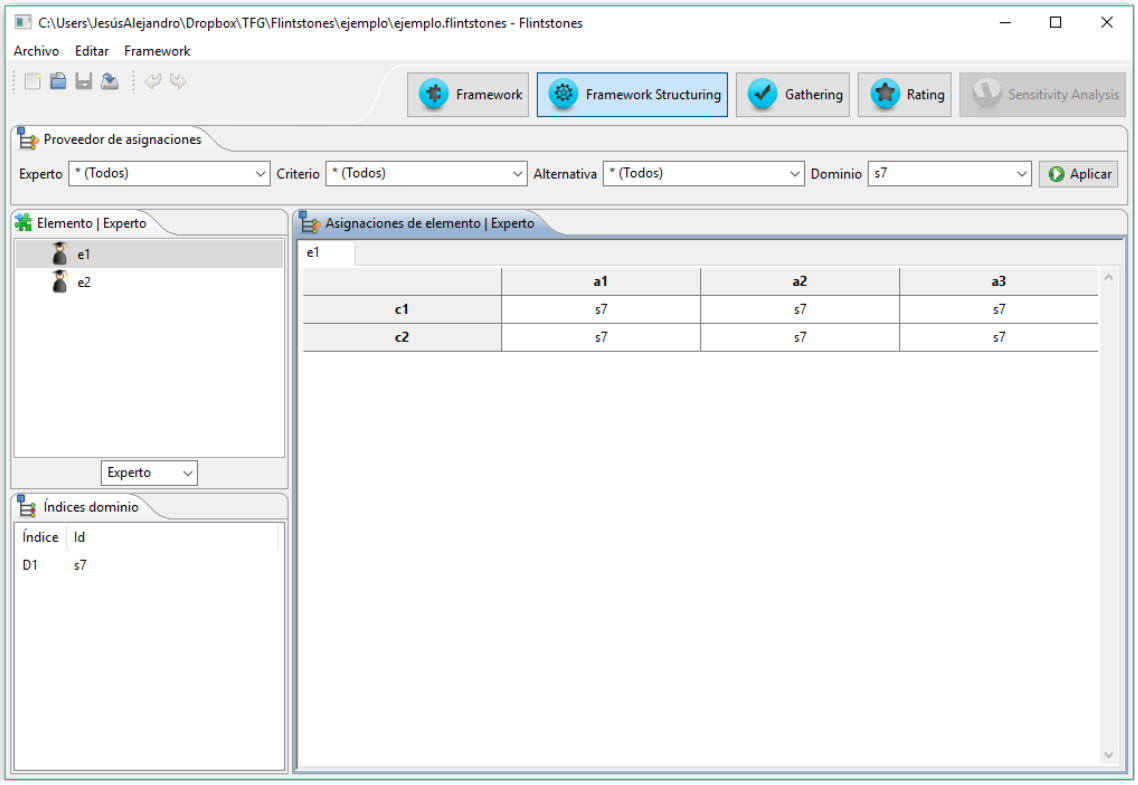

Figura A2.6 Estructura del marco de evaluación

Esta ventana permite o asignar de los diferentes dominios de expresión para cada experto, criterio y alternativa.

Después tenemos el bloque de recogida de información (Gathering) de las distintas valoraciones. A continuación tenemos las valoraciones para el experto1 como se muestra en la Figura A2.7 y para el experto2 en la Figura A2.8.

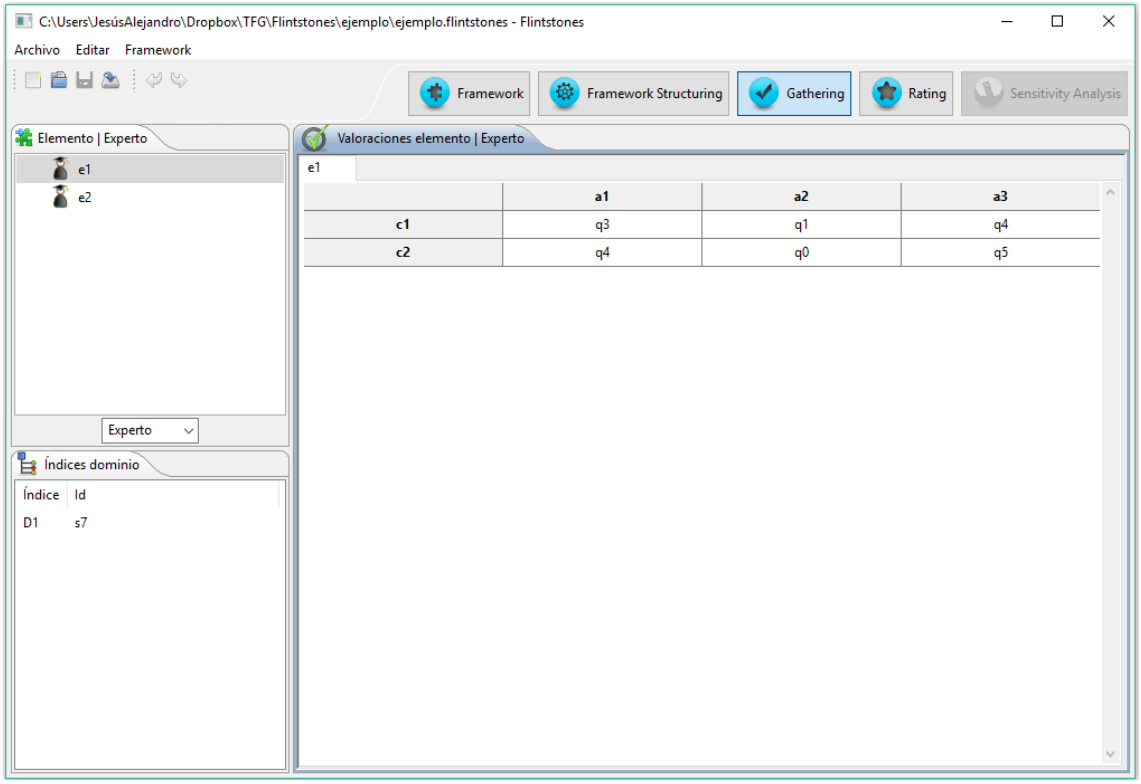

Figura A2.7 Recogida de información para el experto 1.

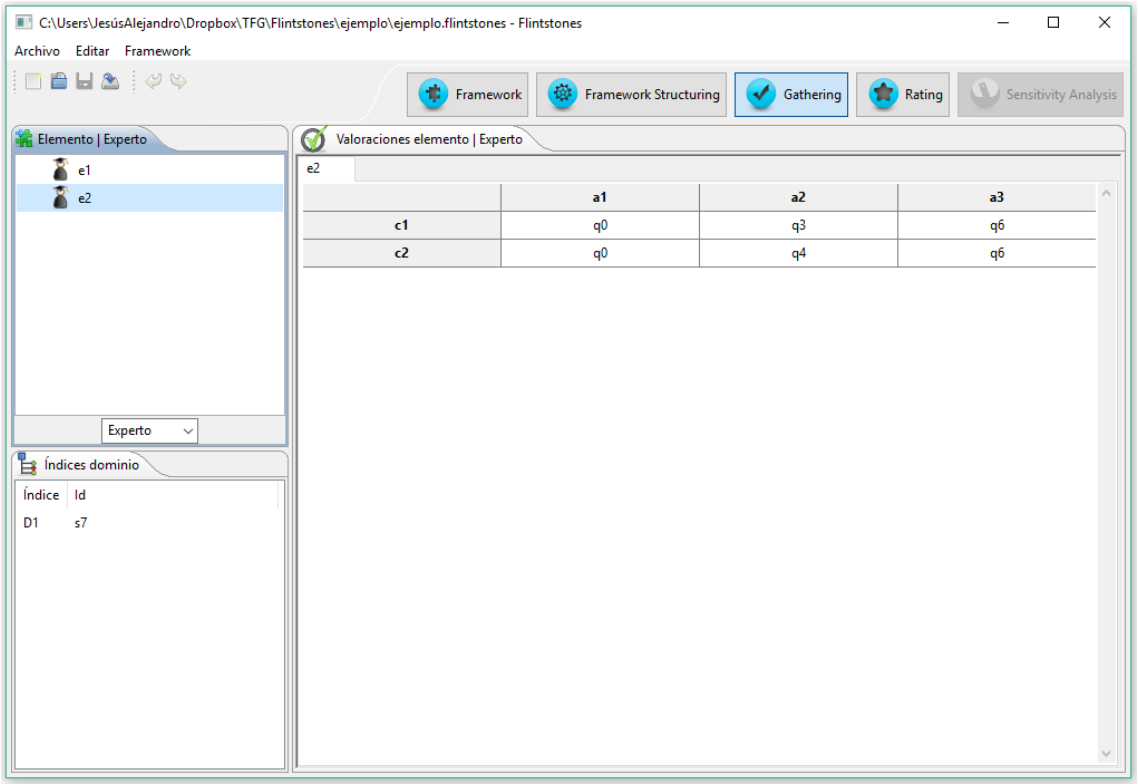

Figura A2.8 Recogida de información para el experto 2.

Para la valoración de las alternativas se escogerá un método de resolución de los disponibles como se muestra en la Figura A2.9.

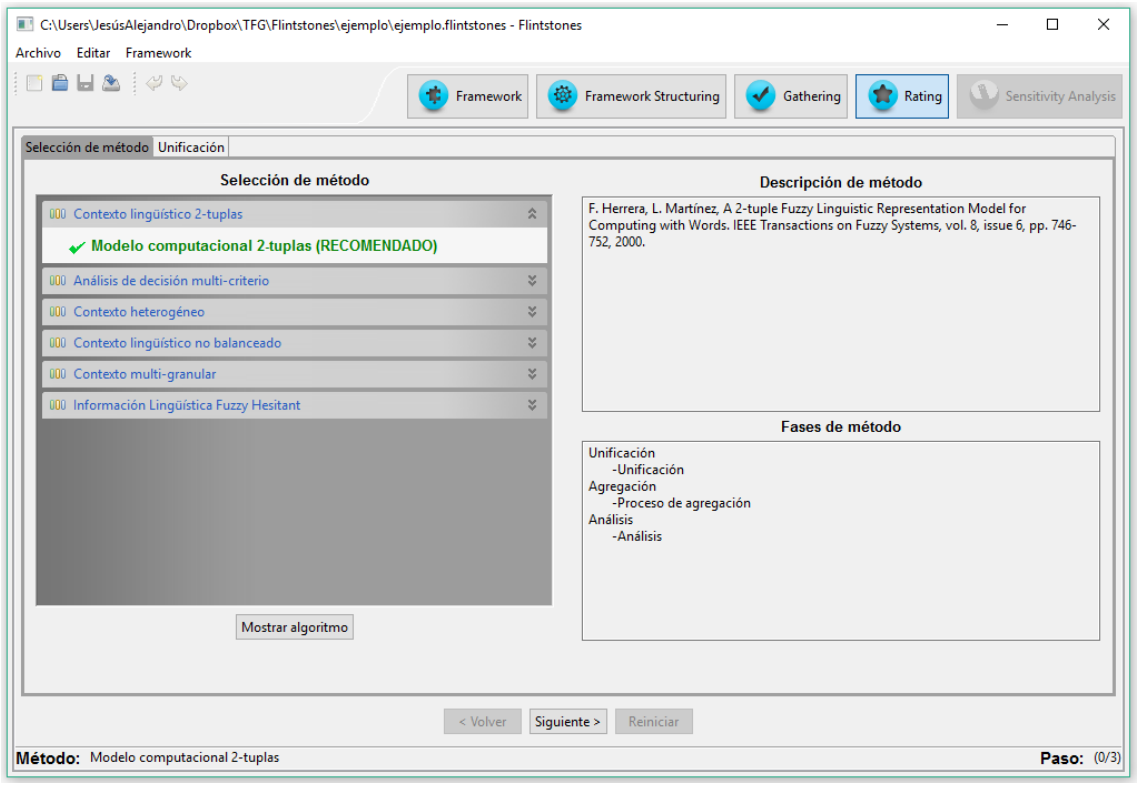

Figura A2.9 Valoración de alternativas.

Como podemos observar en la Figura A2.9 podemos utilizar el modelo 2-Tuplas para este ejemplo, ya que nos muestra la opción en verde como podemos ver en la Figura A2.10 indicándonos que es el modelo adecuado para nuestro ejemplo.

Figura A2.10 Modelo 2-Tuplas ok.

Ya hemos escogido el modelo 2-Tuplas para la resolución de nuestro problema de ejemplo, a continuación la aplicación nos guiará a través de una serie de pantallas como podemos ver en la siguientes figuras.

La primera pantalla que tenemos después de escoger el modelo computacional 2-Tuplas es la fase de unificación, donde las valoraciones se unifican en un solo dominio, como podemos observar en la Figura A2.11.

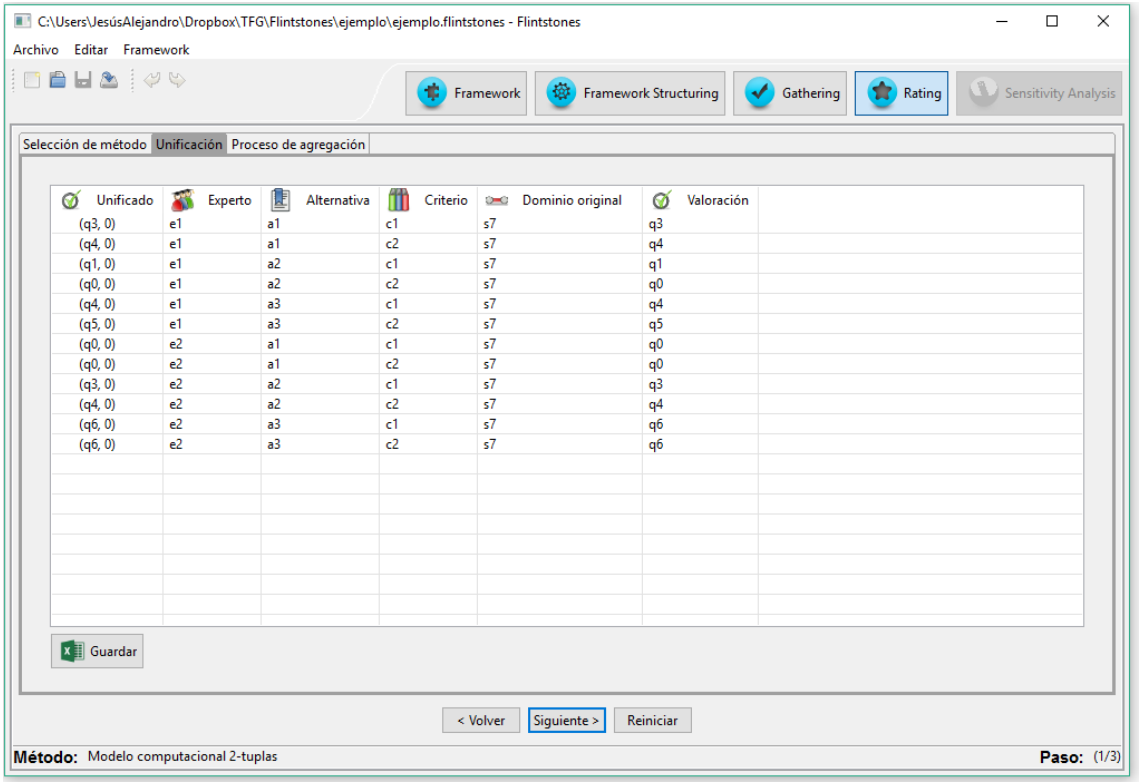

Figura A2.11 Unificación

En la siguiente pantalla corresponde con la fase de agregación, donde se elige el operador de agregación para los expertos y para los criterios, como podemos observar en la Figura A2.12.

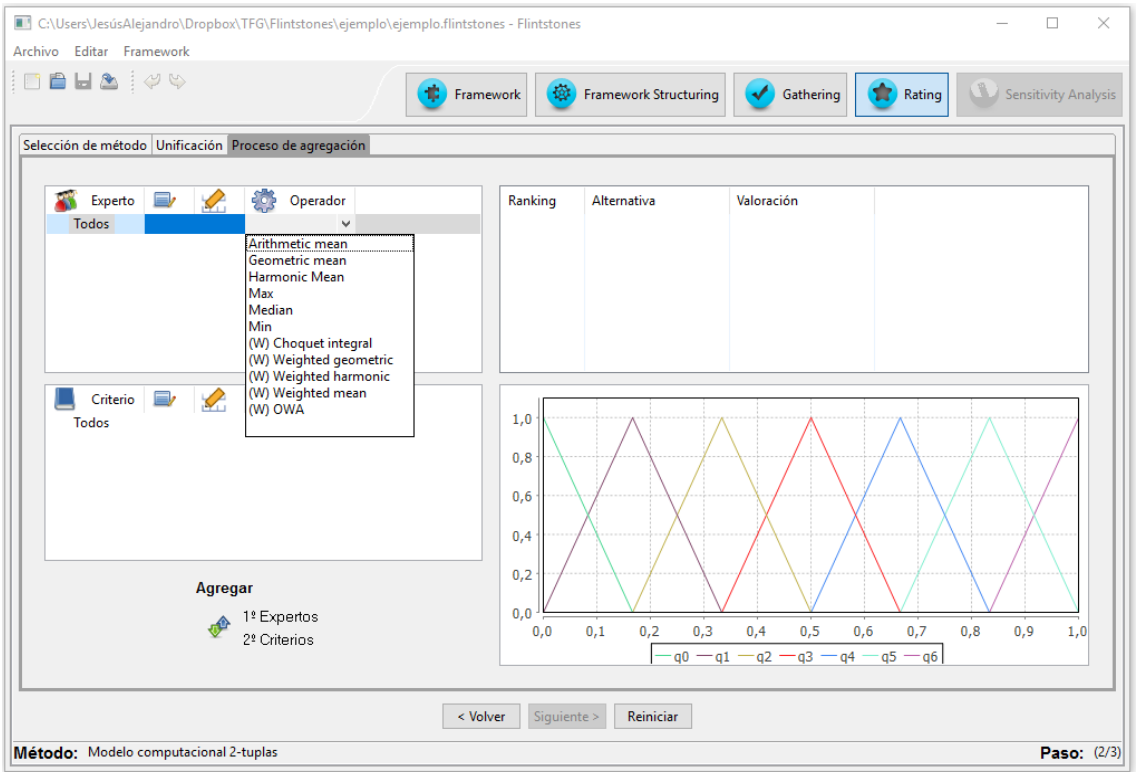

Figura A2.12 Fase de agregación.

Para este ejemplo el operador de agregación tanto para los expertos como para los criterios será la media geométrica, que es uno de los operadores creados en este proyecto como observamos en la Figura A2.13.

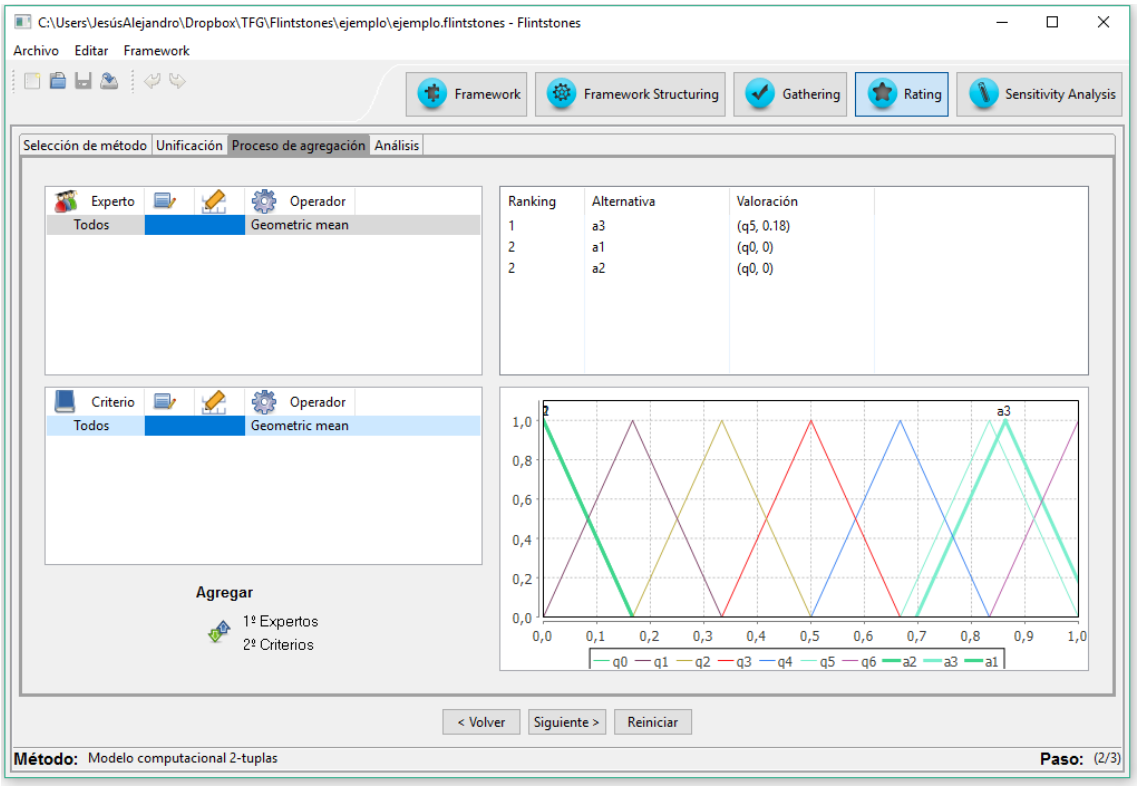

Figura A2.13 Modelo 2-Tuplas pantalla 2 elección del operador de agregación de media geométrica.

Después de escoger el operador de agregación tenemos la fase de análisis como podemos observar en la Figura A2.14.

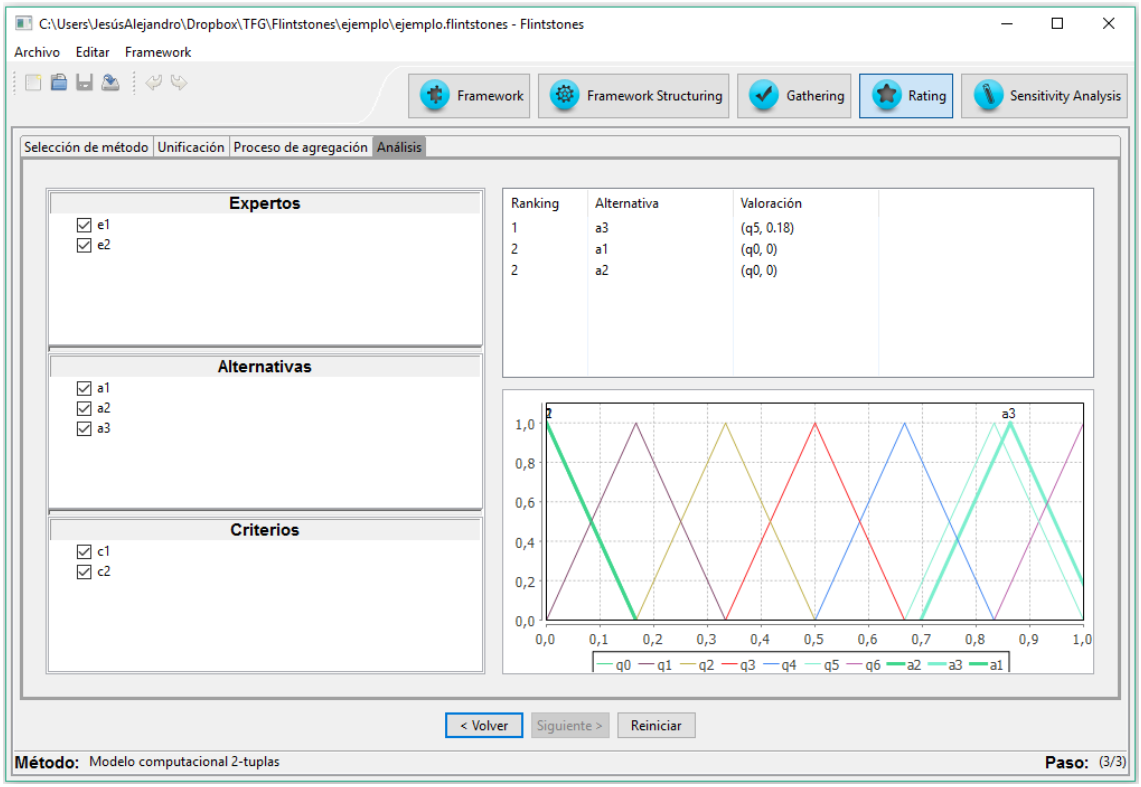

Figura A2.14 Modelo 2-Tuplas pantalla 3 análisis de alternativas.

El último paso es el análisis sensible donde vemos la consistencia de la solución.

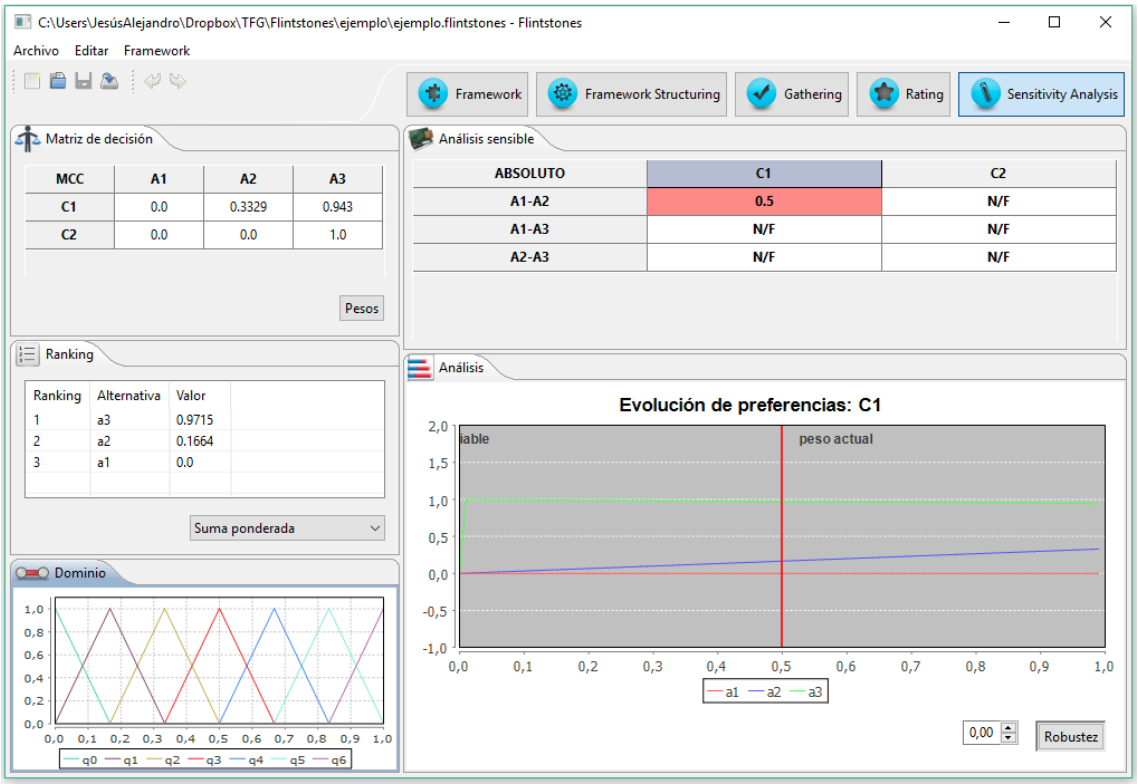

Figura A2.15 Análisis sensible.