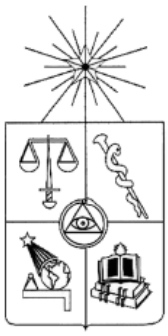


UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

FRAMEWORK PARA IMPLEMENTACIÓN DE INTERFACES EMPÁTICAS EN SOFTWARE EDUCATIVO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

GUILLERMO ANTONIO CAMPOS JORQUERA

PROFESOR GUÍA:
JOSÉ A. PINO URTUBIA

MIEMBROS DE LA COMISIÓN:
JAIME SÁNCHEZ ILABACA
JUÁN ÁLVAREZ RUBIO

SANTIAGO DE CHILE
ABRIL 2012

Resumen

El objetivo del presente trabajo es el desarrollo de un framework para la entrega de retroalimentación a estudiantes que utilizan software educativo. El mensaje a presentar a los alumnos estará basado en el rendimiento y uso del software educativo por parte de los estudiantes. Para lograrlo, se interpreta la información disponible de los usuarios y se evalúan indicadores con ella. Sobre estos indicadores se verifica que se cumplan varias condiciones (como por ejemplo, que su promedio de notas sea sobre 6), se selecciona una y se presenta retroalimentación referente a la misma. El trabajo se desarrolla para la empresa AutoMind, de la cual se obtiene el nombre "interfaz empática" para denominar a la componente responsable de presentar al usuario la retroalimentación.

La primera etapa en el desarrollo de este trabajo correspondió a la identificación de la información de interés sobre el rendimiento de los usuarios para ser utilizada en la generación de los mensajes. Se definieron además los lineamientos para el diseño e implementación de un framework con las características que se solicitan.

Para el diseño se utilizó el paradigma de programación orientada a objetos y se hizo uso de distintos patrones de diseño como por ejemplo: *observer*, *visitor*, *template method y strategy*. El objetivo fue que el framework fuera extensible y permitiera agregar contenido nuevo sin mayores complicaciones. Se identificaron cuatro elementos fundamentales para desarrollar este framework: mensajes, selección de mensajes, condiciones o reglas y fuentes de datos. A cada uno de estos elementos se le asignó un módulo en particular.

La implementación del diseño fue hecha en Java y dio paso a una etapa de evaluación del framework. Esta evaluación se llevó a cabo por medio del cálculo de métricas de software usando la herramienta JDepend y de un test de usabilidad del framework. El framework fue utilizado para hacer mejoras en la retroalimentación para los usuarios de un sistema de votación de la empresa. Finalmente, se transcribió el framework a Javascript con el fin de poder utilizarlo en aplicaciones web.

En síntesis, se construyó un framework para la entrega de retroalimentación a los usuarios de software educativo, utilizando su información de rendimiento y uso. El framework se encuentra implementado en Java y Javascript y fue evaluado por métricas de diseño y de usabilidad. Los resultados de la primera evaluación muestran que el diseño es satisfactorio, pero puede ser mejorado. Por su parte, los resultados de la evaluación por usuarios (desarrolladores), califican de forma positiva el framework destacando principalmente la facilidad de uso y su extensibilidad.

Agradecimientos

A mi familia por su apoyo incondicional. Especialmente a mis padres que siempre han confiado en mí: nada de esto hubiera sido posible sin que ustedes antes lo dieran todo por mí. Los quiero y los admiro más de lo que se imaginan. A mi hermana por estar ahí y soportarme en los momentos complicados, sé que siempre puedo contar contigo.

Espero algún día ser tan bueno como ustedes tres. Son mi vida.

Agradecerles también a mis amigos por hacer el paso por la universidad una alegría. Especialmente para ustedes: Patty que siempre confiaste en mí; Pablo que siempre fuiste mi más cercano; Pedro que siempre me contagiaba con su entusiasmo; Felipe que nunca falló a los asados. A ustedes, mis mejores amigos, que tomamos caminos distintos en la universidad, pero que siempre estuvimos cerca. Los quiero.

A mis amigos que conocí en la carrera, y que hicieron de esta experiencia más que diseñar e implementar código: Lissette, Javiera, Gabriel, Vijay. Gracias por todo.

Gracias a mi profesor guía, José A. Pino, por haberme enseñado tanto durante este proceso, por su buena disposición a responder mis dudas y por haberme apoyado en este cambio de tema. Gracias a mi profesor co-guía, Jaime Sánchez, por lo que aprendí trabajando en C5 y por recibirme con las puertas abiertas para hacer este trabajo.

Sic Parvis Magna

Tabla de Contenidos

RESUMEN ..... **I**

AGRADECIMIENTOS ..... **II**

TABLA DE CONTENIDOS ..... **III**

ÍNDICE DE FIGURAS ..... **VI**

CÓDIGOS VI
IMÁGENES VI
TABLAS VI

1 INTRODUCCIÓN **1**

1.1 ANTECEDENTES GENERALES 1
1.2 MOTIVACIÓN DEL TRABAJO 2
1.3 OBJETIVOS 3
1.3.1 Objetivo General 3
1.3.2 Objetivos Específicos 3
1.4 CONTENIDOS 3

2 ANTECEDENTES **5**

2.1 ELEMENTOS MOTIVACIONALES 5
2.1.1 Motivación Intrínseca 5
2.1.2 Motivación Extrínseca 5
2.1.3 Elementos motivacionales en matemáticas y ciencias 5
2.2 B-LEARNING 7
2.3 AGENTE COMPUTACIONAL 7
2.3.1 Definición 7
2.3.2 Clasificación 8
2.4 TOMA DE DECISIONES 8
2.4.1 Arquitectura genérica de toma de decisiones 8
2.4.2 Arquitecturas de Pizarrón 10
2.5 PROGRAMACIÓN ORIENTADA A OBJETOS 10

3 PRIMERA ETAPA: IDENTIFICACIÓN DE INFORMACIÓN EDUCACIONAL DE INTERÉS **13**

3.1 DESCRIPCIÓN 13
3.2 CONTEXTO 13
3.3 ESTUDIO Y SELECCIÓN DE LA INFORMACIÓN 14
3.3.1 Información Histórica 14
3.3.2 Información de la sesión 15
3.3.3 Estrategias motivacionales 15
3.3.4 Presentación de la retroalimentación y recepción por parte del usuario 15

4 SEGUNDA ETAPA: DISEÑO **17**

4.1 DESCRIPCIÓN 17
4.2 ELEMENTOS A CONSIDERAR EN EL DISEÑO 17
4.2.1 Objetivos del framework 17

4.2.2 Necesidades de información ... 17
4.2.3 Resumen ... 18
4.3 ARQUITECTURA DEL SISTEMA ... 18
4.3.1 Elementos principales del framework ... 18
4.3.2 Arquitectura del framework ... 19
4.3.3 Arquitectura del sistema en producción ... 20
4.4 PRIMER DISEÑO ... 20
4.4.1 Paquete Message ... 20
4.4.2 Paquete Rule ... 20
4.4.3 Paquete Arbiter ... 21
4.4.4 Paquete Data ... 22
4.5 SEGUNDO DISEÑO ... 24
4.5.1 Elementos eliminados ... 24
4.5.2 Elementos modificados ... 24
4.5.3 Elementos nuevos ... 25
4.6 PROTOTIPO ... 26
4.7 TERCER DISEÑO ... 26
4.7.1 Elementos eliminados ... 26
4.7.2 Elementos modificados ... 26
4.7.3 Elementos nuevos ... 27
4.7.4 Soporte para SQL ... 27
4.7.5 Diseño unificado para hacer consultas a fuentes de datos ... 30
4.8 MODELO DE DATOS PARA EL REGISTRO DE EVENTOS ... 31
4.8.1 Tipo de mensaje ... 31
4.8.2 Fuente del mensaje ... 31
4.8.3 Mensajes ... 31
4.8.4 Información del mensaje ... 32
4.8.5 Información del usuario objetivo ... 32
4.8.6 Información del origen del mensaje ... 32

5 TERCERA ETAPA: IMPLEMENTACIÓN ... 34

5.1 PROTOTIPOS ... 34
5.1.1 Prototipo inicial ... 34
5.1.2 Segundo prototipo ... 36
5.1.3 Prototipo Final ... 37
5.2 SEPARACIÓN DEL NÚCLEO DEL FRAMEWORK ... 37
5.3 PROGRAMACIÓN DE CLASES CON MÉTODOS PRE-IMPLEMENTADOS ... 38
5.3.1 Clases abstractas ... 38
5.3.2 Clases concretas y ejecución del framework ... 39
5.4 VISIÓN GENERAL DE FUNCIONALIDADES IMPLEMENTADAS ... 40
5.4.1 Manejo de fuentes de datos ... 40
5.4.2 Definición de reglas para presentar mensajes de retroalimentación ... 41
5.4.3 Priorización de elementos y presentación de mensajes ... 41
5.4.4 Registro de mensajes de retroalimentación ... 41

6 CUARTA ETAPA: EVALUACIÓN ... 42

6.1 IMPLEMENTACIÓN ... 42

6.1.1 Métricas de librería EmpathicCore 42
6.1.2 Métricas de librería EmpathicFw 42
6.1.3 Métricas de librería EmpathicCore y EmpathicFw 43
6.2 USABILIDAD 43
6.2.1 Resultados primera parte de la encuesta 44
6.2.2 Resultados segunda etapa 46

7 QUINTA ETAPA: MEJORAS A UN SOFTWARE EXISTENTE DE LA EMPRESA UTILIZANDO EL FRAMEWORK 50

7.1 DESCRIPCIÓN 50
7.2 INTEGRACIÓN DEL SISTEMA DE VOTACIÓN 50
7.2.1 Sistema de Votación 50
7.2.2 Estado del Sistema de Votación al comenzar el proyecto 53
7.2.3 Mejoras al Sistema de Votación ajenas al framework 53
7.3 MEJORAS AL SISTEMA DE VOTACIÓN UTILIZANDO EL FRAMEWORK 54
7.3.1 Contexto 54
7.3.2 Mejoras utilizando información histórica 55
7.3.3 Mejoras usando información de sesión 58

8 SEXTA ETAPA: TRANSCRIPCIÓN DEL TRABAJO A JAVASCRIPT 59

8.1 DESCRIPCIÓN 59
8.2 REDISEÑO E IMPLEMENTACIÓN 59
8.2.1 Consideraciones generales 59
8.2.2 Programación orientada a objetos en Javascript 59
8.2.3 De multiples threads a observer 60

9 CONCLUSIONES 62

9.1 RESULTADOS OBTENIDOS 62
9.2 TRABAJO FUTURO 63

10 REFERENCIAS 65

11 ANEXOS 66

11.1 EVALUACIÓN: ENCUESTA DE USABILIDAD 66
11.1.1 Primera parte 66
11.1.2 Segunda parte 67
11.2 DISEÑO: MODELO DE DATOS 68
11.3 DISEÑO: DIAGRAMAS DE CLASES 69
11.3.1 Diagrama completo 69
11.3.2 Clases principales: AbstractEmpathicKernel y IArbiter 70
11.3.3 Paquete data en EmpathicCore 71
11.3.4 Paquete rule en EmpathicCore 72
11.3.5 EmpathicFw 73

Índice de figuras

Códigos

Código 7-1: La clase VoteResult 54
Código 7-2: La clase SagdeMetadata 55
Código 7-3: Extensión de un criterio para filtrar datos 55
Código 7-4: Sobrescribir el método canEvaluate de la interfaz IRule 56
Código 7-5: Sobrescribir método evaluateImpl de la clase AbstractRule 57
Código 7-6: Extensión de la clase AbstractMessage 57
Código 7-7: Agregar regla al AbstractEmpathicPlugin 58
Código 8-1: Definición de un constructor en Javascript 59
Código 8-2: Definición de un método para un tipo en Javascript 59
Código 8-3: Simulación de programación orientada a objetos en Javascript 60
Código 8-4: Simulación de herencia en Javascript 60

Imágenes

Figura 2-1: Proceso de decisión 9
Figura 4-1: Estructura de clases del paquete message 21
Figura 4-2: Estructura de clases del paquete rule 21
Figura 4-3: Estructura de clases del paquete data 23
Figura 4-4: Estructura de clases del paquete arbiter 24
Figura 4-5: La clase EmpathicManagers 26
Figura 4-6: La anotación RuleMetadata 26
Figura 4-7: Diagrama de clases para consultas SQL 29
Figura 4-8: Clases descriptoras de consultas sql 30
Figura 5-1: Pantalla principal prototipo inicial 35
Figura 5-2: Mensaje de prueba de aplicación prototipo inicial 35
Figura 5-3: Agregar elementos en aplicación prototipo inicial 35
Figura 5-4: Mensaje de regla activa en aplicación prototipo inicial 35
Figura 5-5: Segundo prototipo. Mensaje de retroalimentación en Android. 36
Figura 5-6: Segundo prototipo. Mensaje de retroalimentación con consulta al usuario 37
Figura 5-7: El tipo predeterminado de fuentes de datos: MemoryDataSource<T> 40
Figura 7-1: Versión inicial del Sistema de Votación 51
Figura 7-2: Segunda versión del Sistema de Votación 52
Figura 7-3: Segunda versión del Editor de preguntas 53

Tablas

Tabla 2-1: Tipos de agentes 8
Tabla 4-1: Comparación de información histórica y de sesión 17
Tabla 4-2: Modelo de datos - Tipos de mensajes 31
Tabla 4-3: Modelo de datos - Fuentes de mensajes 31
Tabla 4-4: Modelo de datos - Mensajes 31
Tabla 4-5: Modelo de datos - Información del mensaje 32
Tabla 4-6: Modelo de datos - Usuario objetivo 32

TABLA 4-7: MODELO DE DATOS - INFORMACIÓN EDUCATIVA 33
TABLA 6-1: MÉTRICAS USANDO JDEPEND PARA EMPATHICCORE 42
TABLA 6-2: MÉTRICAS USANDO JDEPEND PARA EMPATHICFW 42
TABLA 6-3: MÉTRICAS USANDO JDEPEND PARA AMBAS LIBRERÍAS 43
TABLA 6-4: RESULTADOS TEST USABILIDAD - USO GENERAL DEL FRAMEWORK 44
TABLA 6-5: RESULTADOS TEST USABILIDAD - SISTEMA DE PLUGINS DEL FRAMEWORK 45
TABLA 6-6: RESULTADOS TEST USABILIDAD - PROGRAMACIÓN CON EL FRAMEWORK 45
TABLA 6-7: RESULTADOS TEST USABILIDAD - DOCUMENTACIÓN DEL FRAMEWORK 45
TABLA 6-8: RESULTADOS TEST USABILIDAD - NOMBRES DE LOS ELEMENTOS DEL FRAMEWORK 45
TABLA 6-9: RESULTADOS TEST USABILIDAD - SINTAXIS 46
TABLA 6-10: RESULTADOS TEST USABILIDAD - DISEÑO 47
TABLA 6-11: RESULTADOS TEST USABILIDAD - SELECCIÓN DE ELEMENTOS 47
TABLA 6-12: RESULTADOS TEST USABILIDAD - PREGUNTAS GENERALES 48
TABLA 8-1: COMPARACIÓN JAVA - JAVASCRIPT 59

1 Introducción

1.1 Antecedentes generales

Desde los comienzos de la computación la idea de su uso para asistir la educación ha estado presente. En la actualidad la mayor capacidad tecnológica tanto a nivel de redes de comunicación como la movilidad del hardware proveen un escenario en que la integración puede alcanzar lugares más específicos.

Automatización Industrial, de ahora en adelante AutoMind, es una empresa chilena que desde 1988 se dedica a proveer productos, servicios y capacitación a empresas, instituciones y profesionales en diversos países de Latinoamérica. Entre estos productos se encuentran servicios de análisis de vibraciones y sistemas de predicción de riesgo crediticio. En educación AutoMind trabaja con software que funciona al interior del aula y está desarrollando sistemas educacionales que otorguen más movilidad teniendo como objetivo el sistema operativo Android.

AutoMind ha trabajado ampliamente en programas educativos que incluyen el uso de tecnología, principalmente en la implementación de b-learning. B-learning o *blended learning* se refiere a la mezcla de distintos ambientes para la educación. Estos ambientes combinan métodos tradicionales de enseñanza cara a cara con actividades guiadas a través de tecnología, mayoritariamente por medio de computadores. Se le conoce con múltiples nombres: enseñanza integrada, enseñanza híbrida o enseñanza multi-metódica [1].

Uno de los desafíos más importantes con los que se ha encontrado AutoMind en estos programas educativos es el de encontrar formas de motivar a los usuarios de los distintos sistemas educacionales. El principal objetivo de incrementar la motivación de los estudiantes es buscar un impacto mayor de los contenidos y un mejor aprendizaje. En la literatura se reconocen diversas formas de motivar a las personas en actividades de distinto tipo. Cada una de estas estrategias tiene efectos diferentes dependiendo del individuo, pero se distinguen elementos motivacionales de dos tipos: intrínseco y extrínseco.

Un elemento motivacional intrínseco es aquel que proviene desde el interior del individuo y no de un medio o estímulo externo como el dinero o las calificaciones. La motivación intrínseca proviene de la satisfacción que se consigue por el desarrollo mismo o por la finalización de una tarea. Ejemplos de motivación intrínseca son descubrir un objeto oculto, encontrar simetrías y predecir eventos. Un elemento motivacional extrínseco, a diferencia de uno intrínseco, es aquel ajeno al individuo como por ejemplo un premio o una paga. La motivación extrínseca no proviene ni del desarrollo de la tarea ni por su finalización, aun cuando esta pudiese existir, pero implica que la persona seguirá realizando la tarea con tal de recibir el estímulo aun cuando la tarea en sí no le resulte atractiva.

En un ambiente de aprendizaje, la habilidad de mostrar emoción, empatía y entendimiento a través de expresiones faciales o lenguaje corporal es central para asegurar la calidad de la interacción entre estudiante y profesor [2]. En un contexto digital sin embargo, se sabe poco de cómo o inclusive cuándo las emociones pueden ser transmitidas efectivamente por el software

[2]. Se han hecho aproximaciones al tema usando avatares que tienen capacidades expresivas limitadas pero que enriquecen la interacción entre el usuario y el software [2].

Existen trabajos que han desarrollado agentes o tutores empáticos utilizando diversas metodologías: como eye-tracking [3] y cumplimiento de objetivos [4]. Otros trabajos se han enfocado en poder definir un lenguaje para representar las emociones [2] y también en el desarrollo de frameworks para el diseño de agentes empáticos que acompañen al usuario durante el uso de sistemas educacionales [5].

Parte de la motivación de los trabajos recién mencionados es la importancia de que el usuario reciba una retroalimentación que mantenga su interés y concentración en la actividad que desarrolla, generalmente en un contexto educativo.

1.2 Motivación del Trabajo

Existe la necesidad de hacer las actividades más atractivas para los estudiantes que utilizan los distintos sistemas educacionales de AutoMind. Se busca además que al hacer las actividades más atractivas, el desempeño de los alumnos mejore y por lo tanto el aprendizaje obtenido al usar los sistemas aumente. Para abordar este problema se ha hecho uso de interfaces que entregan mensajes positivos de retroalimentación a los alumnos basados en aspectos de su rendimiento y estadísticas de uso. La empresa ha denominado a estas interfaces: interfaces empáticas.

En el pasado, AutoMind ha desarrollado interfaces empáticas de distinta complejidad en algunos de sus proyectos como Sorpresas Mágicas, juego que se ha utilizado en los "Torneos Nacionales de Juegos Online de Matemáticas". Todas las interfaces desarrolladas anteriormente han sido específicas para cada programa y no han ocupado un framework o librería común. Se reconoce que es inevitable hacer desarrollos que sean propios para cada programa, sobre todo considerando que tienen objetivos y enfoques educativos distintos y por lo tanto, diferentes necesidades. Sin embargo, es deseable que exista un punto de partida base para el desarrollo de las interfaces empáticas, de manera de disminuir el esfuerzo requerido en su implementación. El hecho de que la mayoría de los sistemas educacionales de AutoMind estén desarrollados en Java y que actualmente se esté comenzando el desarrollo de sistemas para Android, abre la posibilidad de elaborar una librería que cumpla el rol de base tanto para plataformas de escritorio como plataformas móviles.

Se busca desarrollar un framework para implementar lo que AutoMind denomina como un "motor empático". Este motor empático es un agente capaz de decidir la retroalimentación que debe dar a los usuarios del software en base a distintos estímulos. Ejemplos de estímulos a los que debiese responder el motor son: cambios en valores de variables de la aplicación, rendimiento del usuario o llamada directa por parte del tutor. El framework debe ser extensible, de forma de poder agregar de manera sencilla nuevos elementos como nuevas fuentes de datos y su interpretación y evaluación. La importancia de realizar un framework con estas características radica en que se puede contar con una forma estructurada y común para el desarrollo de herramientas que presenten retroalimentación al usuario. Una solución de este tipo permite evaluar de manera más ordenada cuales son los resultados obtenidos con la retroalimentación entregada. Además, provee un mecanismo para hacer desarrollos que requiere menos esfuerzo

que el que se hace en la actualidad, en que para cada entorno hay que conocer un sistema distinto y por consiguiente las diferentes componentes que componen a dichos sistemas.

Se conocen dos trabajos que han abordado el problema de desarrollar un framework para modelar agentes empáticos. Uno de estos frameworks está orientado al entrenamiento de agentes empáticos basado en la interacción entre personas [5]. El otro framework está pensado en el desarrollo de agentes empáticos para aplicaciones en ambientes virtuales colaborativos [2]. Ambos trabajos, aunque distintos a lo que se busca, presentan varias consideraciones de diseño que son útiles para el desarrollo de un framework para entregar información y retroalimentación en programas educacionales.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un framework para la entrega de retroalimentación a estudiantes que utilizan software educativo. La retroalimentación a presentar debe estar basada en información de rendimiento y uso del software educativo por parte de estudiantes. Para lograr esto, se interpreta la información disponible de los usuarios y se evalúan indicadores con ella. Sobre estos indicadores se verifica que se cumplan condiciones; se prioriza una de las condiciones que se cumplen. Finalmente se presenta retroalimentación basada en la condición seleccionada.

1.3.2 Objetivos Específicos

- Diseñar un framework que permita evaluar indicadores sobre información de uso y rendimiento de software educativo y para verificar condiciones (o reglas) sobre estos indicadores.
- Incluir en el diseño del framework un mecanismo para priorizar una de las condiciones (mencionadas en el punto anterior) que se cumplan y presentar retroalimentación referente a la misma.
- Utilizar el framework para hacer mejoras en la retroalimentación de al menos uno de los sistemas educacionales de AutoMind.
- Evaluar la extensibilidad del framework usando métricas de software y un test de usabilidad.

1.4 Contenidos

A continuación se listan las distintas secciones del presente trabajo y se hace una breve referencia al contenido de éstas.

1. Introducción. Se muestra una mirada general al tema del presente trabajo y la motivación. Se exponen los objetivos del trabajo a realizar.
2. Antecedentes. Se presenta el marco teórico y elementos necesarios para el desarrollo de este trabajo.
3. Primera etapa: Identificación de información educacional de interés. Se hace un reconocimiento de las principales necesidades de información a utilizar por el trabajo a desarrollar.

4. Segunda etapa: Diseño. Se expone el diseño de la estructura de clases del presente trabajo. Se presenta también el modelo de datos para almacenar información sobre la retroalimentación entregada. Esta segunda etapa se divide en tres sub-etapas de diseño incrementales.
5. Tercera etapa: Implementación. Se muestran detalles de la implementación del trabajo.
6. Cuarta etapa: Evaluación. Se presentan la evaluación del trabajo, compuesta de dos partes: cálculo automatizado de métricas de software de interés; y test de usabilidad a usuarios finales del trabajo, en este caso, desarrolladores.
7. Quinta etapa: Mejoras a un software existente. Se exponen las mejoras realizadas a un software de existencia previa en la empresa. Estas mejoras se llevan a cabo usando la herramienta desarrollada.
8. Sexta etapa: Transcripción del trabajo a Javascript. Se muestra el trabajo realizado para migrar el trabajo realizado a Javascript, incluyendo cambios de enfoque necesarios para poder lograrlo.
9. Conclusiones: Se exponen los resultados obtenidos, incluyendo conclusiones y posible trabajo futuro a realizar.
10. Referencias: Se presentan las fuentes de información bibliográfica utilizadas en relación al marco teórico y trabajos relacionados sobre el tema en estudio.
11. Anexos: Se muestra información adicional y de interés sobre el desarrollo de la memoria.

2 Antecedentes

En esta sección se recopila información relevante para realizar el presente trabajo. Para cumplirlo se considera importante tener información de elementos motivacionales generales y para estudiantes, del modelo educativo ("B-Learning") y elementos de toma de decisiones. Las siguientes subsecciones abarcan dichos contenidos.

2.1 Elementos motivacionales

El objetivo principal de este trabajo (ver sección 1.3 - Objetivos), responde a la necesidad de motivar a los estudiantes en el uso de software educativo. Se reconocen dos tipos de motivación: intrínseca y extrínseca. Adicionalmente del trabajo del profesor Roberto Araya Schulz para la OEA [6] [7], se reconocen distintos elementos motivacionales posibles de encontrar en matemáticas y ciencias.

2.1.1 Motivación Intrínseca

Un elemento motivacional intrínseco es aquel que proviene desde el interior del individuo y no de un medio o estímulo externo como el dinero o las calificaciones. La motivación intrínseca se genera por el desarrollo y/o la finalización de una tarea. Este tipo de motivación refleja el deseo de hacer algo únicamente porque se le considera disfrutable. Para una persona intrínsecamente motivada, el placer que experimenta al realizar una actividad es suficiente para querer volver a realizarla en el futuro. Ejemplos de motivación intrínseca son descubrir un objeto oculto, encontrar simetrías y predecir eventos.

2.1.2 Motivación Extrínseca

Un elemento motivacional extrínseco es aquel ajeno al individuo como por ejemplo un premio o una paga. La motivación extrínseca no proviene ni del desarrollo de la tarea ni por su finalización, aun cuando esta pudiese existir, pero implica que la persona seguirá realizando la tarea con tal de recibir el estímulo aun cuando la tarea en sí no le resulte atractiva.

Existe una amplia gama de ejemplos de motivación extrínseca. Se puede encontrar entre estos casos a la gente que participa en concursos únicamente por el premio que puede obtener en ellos, o que realiza trabajos que no le agradan, pero que le permiten una retribución económica superior a la de un trabajo en el que sí se siente motivada.

2.1.3 Elementos motivacionales en matemáticas y ciencias

Del trabajo del profesor Roberto Araya Schulz para la OEA [6] [7], se reconocen distintos elementos motivacionales posibles de encontrar en matemáticas y ciencias.

1. **Alcanzar una meta**. Cada vez que se completa un desafío que requirió esfuerzo se experimenta un sentimiento de agrado. Si el desafío es pequeño, entonces seguramente se experimenta un pequeño momento de triunfo. Si el desafío es grande, en cambio, la experiencia de satisfacción puede ser mucho mayor.
2. **Encontrar la esencia de algo**. La identificación de formas y/o patrones que es una labor importante para la supervivencia. Por esa razón, el cerebro posee mecanismos especializados en estos elementos y producir satisfacción al hacerlo. Al detectar la esencia

de algo no hay ruido o elementos distractores que perturben o recarguen el procesamiento de información. La atracción por la esencia, se observa en el gusto por las maquetas o las caricaturas de personas.

3. **Armar y Componer**. Niños y niñas de menos de dos años ya muestran un gran interés y un fuerte impulso por armar estructuras con bloques. Esta misma sensación se observa en adolescentes y adultos en la construcción de casas, edificios y maquinarias. Algo similar ocurre al armar un edificio conceptual (un modelo teórico, un modelo computacional o un sistema axiomático) sobre el cual se construyen conceptos y estructuras más complejas.
4. **Descubrir objeto oculto**. El ser humano está biológicamente determinado para buscar y detectar rápidamente caras y otros animales ocultos en un escenario de información visual desordenado y confuso. Muchos problemas matemáticos muestran esta misma característica. Por ejemplo, encontrar objetos o relaciones ocultas en información numérica o encontrar estructuras ocultas en un conjunto de elementos conectados y relacionados entre sí.
5. **Encontrar metáforas**. Una metáfora es una analogía para explicar un fenómeno nuevo o complejo sobre la base de similitudes con otros más conocidos y/o naturales para la persona. Encontrar una metáfora apropiada, con gran capacidad de representar el fenómeno complejo y parte de su estructura, genera una experiencia de satisfacción.
6. **Encontrar similitudes en diferentes objetos o fenómenos**. Identificar elementos comunes o parecidos en diferentes objetos o fenómenos y agruparlos según factores o componentes similares es un proceso muy importante para sintetizar información. Este proceso es muy parecido al de encontrar una metáfora.
7. **Conectar diferentes representaciones de un mismo fenómeno**. Un mismo fenómeno puede admitir varias representaciones. Por ejemplo, una representación visual y manual puede ejemplificar propiedades de un concepto o idea abstracta. Este tipo de conexión es atrayente y despierta interés. Mientras mayor sea la diferencia en el nivel de abstracción entre la representación simbólica y la representación concreta, mayor es la atracción que genera.
8. **Encontrar simetrías e invariancias**. La atracción por la simetría se da en materias tan cotidianas como en los rostros de las personas, como en otras más abstractas que van desde mosaicos utilizados en arquitectura hasta simetrías en ecuaciones y relaciones simbólicas en matemáticas y física.
9. **Experimentar emergencia en patrones**. Observar o presenciar cómo a partir de elementos simples emerge un fenómeno nuevo más complejo es fuente de atracción intelectual. El fenómeno que emerge puede ser de una naturaleza muy distinta de las que le dieron origen y mientras mayor sea esa diferencia más atractivo provoca. Por ejemplo, del uso de dos espejos emerge un fenómeno infinito al colocarlos paralelos.
10. **Encontrar punto de vista genérico y no meras coincidencia**s. Comprender que un fenómeno dado no es pura coincidencia sino que es un caso particular de una situación general o de reglas generales es motivante intelectualmente.
11. **Predecir (apostar) y aceptar**. La mente realiza predicciones constantemente. Cuando éstas se hacen de forma consciente y con cuidado se dice que se planifica. Esto en sí es una actividad atractiva, pero, al acertar, el grado de satisfacción es aún mayor. Muchos aspectos de la matemática tienen esta estructura. Por ejemplo, la predicción de la posición de un planeta en un cierto día o la predicción de un eclipse. Otro ejemplo, más común esta vez, es intentar predecir los resultados de partidos de fútbol utilizando los resultados de partidos anteriores.

12. **Explicitar procesos y explicar**. Existe un gran atractivo en poder explicarse uno mismo y en explicar a otros las causas de un fenómeno. Aquí reside, en parte, la causa del gusto por enseñar. Esta atracción puede también observarse en el caso opuesto, en el desagrado y mal rato que se produce si alguien hace ver que una explicación es incorrecta.
13. **Experimentar violaciones a la intuición**. Existen cinco elementos en esta materia: la causalidad, la lógica intuitiva, las nociones espaciales, la física intuitiva y la biología intuitiva. Ejemplos típicos para estos elementos son frases como "esta frase miente" para la lógica intuitiva e ilusiones ópticas con objetos y sus proporciones para noción espacial.

2.2 B-Learning

B-learning o *blended learning* se refiere a la mezcla de distintos ambientes para la educación. Esta forma de educar combina métodos tradicionales de enseñanza cara a cara con actividades guiadas a través de tecnología, mayoritariamente por medio de computadores. Se le conoce con múltiples nombres como enseñanza integrada, enseñanza híbrida o enseñanza multi-metódica [1].

Existen diversos estudios con resultados favorables y desfavorables para distintas combinaciones en *b-learning*. Estas combinaciones difieren en qué tanta prioridad, o qué tanto se depende de educación tradicional o educación en línea. Para distinguir a los distintos niveles de importancia que se da a cada uno de estos elementos se utiliza el término 'Continuo del b-learning'.

En [8] se distinguen varios niveles o modelos dentro del continuo del *b-learning*. Estos modelos varían desde más participación guiada a través de tecnología y menos participación tradicional cara-a-cara, a menos participación guiada a través de tecnología y más participación tradicional cara-a-cara, tal como se presenta a continuación.

1. Modelo 1: Currículum totalmente en línea con actividades cara a cara tradicionales.
2. Modelo 2: El currículum es mayormente o totalmente currículum en línea. En este modelo las actividades cara a cara no son opcionales sino requeridas. Estas se pueden hacer en la sala de clases o en el laboratorio de computación.
3. Modelo 3: La mayoría o casi todo el currículum se hace en línea. Los estudiantes se reúnen diariamente en la sala de clases o en el laboratorio de computación.
4. Modelo 4: La instrucción se hace en la sala de clases y requiere de un uso sustancial de componentes en línea. El uso de estas componentes se extiende más allá de la sala de clases y/o el horario de clases.
5. Modelo 5: La instrucción se realiza en la sala de clases e incluye algún grado de recursos en línea. Sin embargo, el uso de estos recursos es limitado o inclusive nulo.

2.3 Agente Computacional

2.3.1 Definición

Un agente computacional es un programa que trata de cumplir una serie de objetivos en un ambiente complejo y dinámico [9]. Los objetivos de un agente pueden ser muy diversos. Un agente puede tratar de alcanzar un estado determinado mientras otro busca maximizar un "premio".

En [10] se da la siguiente definición para un agente autónomo: Un agente autónomo es un sistema situado al interior y es parte de un ambiente. El agente siente el ambiente y actúa en él a través del tiempo con el fin de cumplir un objetivo y de esta forma puede afectar lo que sentirá en el futuro.

2.3.2 Clasificación

La definición de agente es muy amplia y para que sea práctica es necesario hacer una clasificación más detallada de agentes [10]. En la Tabla 2-1: Tipos de agentes, se propone una clasificación de agentes con una breve descripción de éstos.

Tabla 2-1: Tipos de agentes

Clasificación	Otros nombres	Descripción
Reactivo	Sensible	Actúa en respuesta a cambios en el ambiente
Autónomo		Ejerce control sobre sus propias acciones
Orientado a objetivos	Pro-activo	No actúa simplemente en respuesta al ambiente
Continuo		Es un proceso que se ejecuta de forma continua
Comunicativo	Socialmente capaz	Se comunica con otros agentes, quizás también con gente
Aprendedor	Adaptativo	Cambia su conducta de acuerdo a su experiencia previa
Móvil		Es capaz de transportarse a sí mismo de una máquina a otra
Flexible		Sus acciones no están pre-escritas
Con carácter		Tiene una personalidad creíble y un estado emocional

2.4 Toma de decisiones

2.4.1 Arquitectura genérica de toma de decisiones

En un proceso genérico de toma de decisiones se reconocen los siguientes elementos [11]:

- Ambiente: Es el contexto dentro del cual un determinado elemento (posiblemente un agente) se encuentra enmarcado.
- Evento: Corresponde a una situación ocurrida al interior del ambiente.
- Actor: Es un elemento capaz de interactuar con el ambiente. Un actor posee una memoria propia que utiliza para identificar a otros actores y cambios dentro del ambiente.
- Actor autónomo: Corresponde a un actor independiente. Un actor autónomo necesita una arquitectura para tomar sus decisiones.
- Acción: Evento generado por un actor al interactuar con el ambiente.

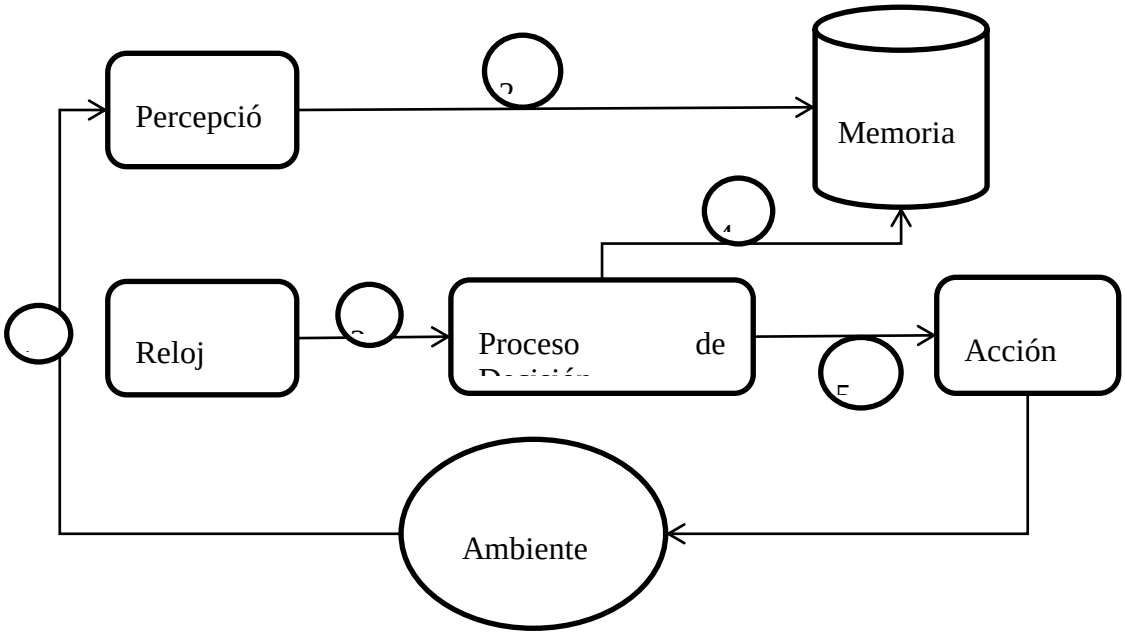
En la arquitectura propuesta en [11], los objetos claves corresponden a los eventos. Cuando un actor interactúa con el ambiente a través de acciones, el resultado de éstas se propaga

a otros actores que las perciben como eventos. Un evento contiene información acerca del tipo de evento, el actor fuente y el acto objetivo. El actor que percibe el evento verifica en su lista de eventos si cuenta con alguna forma de manejarlo y de ser así, el actor trata de identificar a los actores fuente y objetivo en su memoria interna. El manejador del evento en esta instancia puede tomar dos caminos: gatillar directamente una llamada en el actor fuente y objetivo, o actualizar la memoria interna de ambos actores respecto del evento. Ambas decisiones tienen diferentes implicaciones sobre el rendimiento de la toma de decisión.

En el enfoque activo (realizar una llamada) se gana en tiempo de respuesta, pero es muy exigente en ambientes donde ocurren muchos eventos y puede acarrear problemas de rendimiento de la aplicación. En el enfoque pasivo (actualizar memoria) se requiere de una revisión recurrente de la memoria interna para tomar la decisión, perdiéndose posiblemente rapidez en la respuesta, pero con una ventaja en escalabilidad cuando existen demasiados eventos. La arquitectura propuesta en [11] se inclina por la segunda opción.

Teniendo esto en cuenta, el proceso responsable de verificar la memoria determina también cuál es la conducta apropiada a seguir. En ningún caso se implica hasta ahora como se realiza la decisión, dejando esta componente del sistema abierta a distintas formas de implementación. En la Figura 2-1 se grafican las diferentes etapas que forman parte del proceso de toma de decisiones en un actor.

Figura 2-1: Proceso de decisión



1. El actor percibe cambios en el ambiente como eventos.
2. Estos eventos cambian la memoria del ambiente del actor.
3. Independientemente, el actor realiza un proceso programado de toma de decisiones (posiblemente con un reloj).
4. El proceso de decisión accede a la memoria interna del actor.
5. El proceso de decisión determina qué acción realizar.
6. El proceso de acción causa cambios en el ambiente.

2.4.2 Arquitecturas de Pizarrón

Un tema relevante a la hora de tomar decisiones es la coordinación de las distintas partes involucradas en el proceso. La arquitectura de pizarrón es uno de los posibles medios para poder manejar coordinación. A pesar de no ser compleja de implementar, la arquitectura ha probado ser suficientemente elegante y poderosa como para ser útil en programas de distintos tipos [12].

Esta arquitectura está construida alrededor de una metáfora: El pizarrón tradicional de un colegio era una herramienta de resolución de problemas. La utilidad del pizarrón radica en que la información está disponible para todos quienes observen el pizarrón. De esta forma se puede también dividir el trabajo y los observadores pueden colaborar por medio del pizarrón. En el Anexo se encuentra un diagrama con la estructura genérica de la arquitectura de pizarrón.

Del trabajo de [12] se identifican los siguientes elementos:

- Pizarrón (*Blackboard*): Un fuente pública (en el contexto de la aplicación) de información legible y escribible. Típicamente, un pizarrón contiene una lista de afirmaciones lógicas por las cuales sus componentes operan. Estas afirmaciones pueden estar organizadas en una fuente común, pero es recomendable usar una estructura para organizarlas. De esta forma, las componentes pueden buscar en lugares específicos del pizarrón.
- Fuentes de Conocimiento – FC (*Knowledge Source* – KS): En torno al pizarrón existe una serie de componentes que son capaces de operar con la información que éste contiene. Estos son los elementos que colaboran en la resolución del problema. Las FC usualmente están inactivas, esperando que un conjunto de precondiciones se cumplan. Una FC puede presentar un índice de relevancia o "deseo de correr" indicando su nivel de aplicabilidad respecto a los contenidos actuales del pizarrón.
- Árbitro (*Arbiter*): Dado un solo vistazo al pizarrón y sus contenidos, es posible que un número de FCs presenten una relevancia mayor que cero. El árbitro es la componente encargada de decidir cuáles FCs son relevantes para ser ejecutadas. Este paso es crítico y constituye la totalidad de la estrategia para controlar el pizarrón.

2.5 Programación Orientada a Objetos

La programación orientada a objetos es un paradigma de programación basada en la representación de entidades y las interacciones que se dan entre ellas a través de lo que se denomina objetos. Los objetos poseen un estado representado por atributos, un comportamiento representado por sus métodos, y una identidad o identificador que lo diferencia de otros objetos [13].

Dado que el diseño de aplicaciones basadas en este paradigma de programación es de uso amplio y existen escenarios con interacciones típicas entre distintos objetos, se propone abordar el problema mediante el uso de patrones para el diseño. Los patrones de diseño representan soluciones a problemas habituales en el diseño de una estructura de clases y su uso permite generar código reusable, fácil de re implementar y extender. Se busca por ende definir relaciones estables entre elementos y generar objetos extensibles. Mediante el uso de estas técnicas se logra que el diseño desarrollado en particular para el problema presentado, sea lo bastante general como para agregar funcionalidades a futuro [13].

A continuación, se presentan algunos patrones de diseños que se considera podrían ser de utilidad en el presente trabajo:

- Abstract Factory: Proporciona una interfaz que permite crear familias de objetos relacionados sin especificar sus clases concretas. Se utiliza cuando se quiere definir sistemas independientes de cómo se crean, componen y representan cada una de sus partes y, cuando se desea proveer una librería de clases, revelando sólo la interfaz pero no su implementación.
- Chain of Responsibility: Este patrón permite definir una serie de objetos encargados de procesar una acción. Mediante la definición de esta serie se establece una prioridad de quien es responsable del procesamiento. Si un elemento de la cadena no es capaz de atender a la solicitud o lo hace de manera incompleta, pasa la responsabilidad al elemento siguiente.
- Composite: Compone objetos en estructuras de árbol para representar jerarquías generales o parciales. Permite al cliente tratar objetos individuales y composiciones de objetos de forma uniforme. Sirve para representar jerarquías, y permite ignorar la diferencia entre la composición de los objetos y el objeto individual.
- Facade: Provee una interfaz unificada para un conjunto de interfaces pertenecientes a un subsistema. La idea de esta interfaz es proveer una abstracción de alto nivel que interactúe con el subsistema de forma más sencilla. Es de gran utilidad en presencia de subsistemas completos pues permite mantenerlos aislados y manejarlos de forma independiente de los elementos que interactúan con él.
- Factory Method: Define una interfaz para crear un objeto pero la subclase decide qué clase instanciar. Permite a la clase aplazar la instanciación de la subclase. Se utiliza cuando una clase no puede anticipar el tipo de objeto que se necesitará crear, o cuando la clase delega en subclases las responsabilidades de ciertas tareas.
- Iterator: Proporciona una forma de acceso a los elementos de un objeto de agregación secuencialmente sin exponer su representación. Permite proveer una interfaz uniforme para recorrer distintas estructuras agregadas.
- Observer: Define una o más dependencias entre objetos de modo que cuando uno de los objetos cambia de estado, todas las dependencias de él son notificadas y actualizadas automáticamente. Es útil cuando un objeto requiere cambiar a los demás y no sabe quiénes son ni cuántos hay.
- Singleton: Asegura que una clase sólo posee una instancia, o un número limitado y administrado de ellas, y proporciona un punto de acceso global a ella(s).
- Strategy: Define una familia de algoritmos, encapsulando cada uno y haciéndolos intercambiables. Strategy permite al algoritmo variar independientemente de los clientes que lo usan. Es útil cuando se tienen clases relacionadas que sólo difieren en su comportamiento, o se necesitan muchas variantes del mismo algoritmo. Permite además, evitar la exposición de estructuras complejas y ocultar los datos del algoritmo.
- Template Method: Define el esqueleto de un algoritmo en una operación, diferenciando algunos pasos en subclases. Permite a las subclases redefinir ciertos pasos del algoritmo sin cambiar la estructura propia de dicho algoritmo. Permite evitar la duplicación de código, cuando un grupo de subclases tienen un comportamiento común que se centraliza en una sola clase.

- Visitor: Este patrón define una forma de separar un algoritmo de la estructura sobre la cual opera. Esta separación permite agregar nuevas funcionalidades a objetos existentes sin modificar sus estructuras. O disponer de una implementación diferente del algoritmo en un contexto distinto.

3 Primera etapa: Identificación de información educacional de interés

3.1 Descripción

La primera etapa del trabajo corresponde a la identificación de las distintas preguntas frecuentes que se pueden realizar al sistema, información educacional clave que se solicite de forma recurrente y elementos motivacionales claves. En esta categoría se encuentran, por ejemplo: alumno con mejor rendimiento en un curso, curso con mejor rendimiento, etc. Se estudian, además, fuentes motivacionales propias de matemáticas y ciencias: fuentes cognitivas, herramientas e interpersonales.

El objetivo principal de esta etapa es poder identificar detalladamente cuáles son las necesidades de información existentes. Una vez hecho esto es posible dar paso a las siguientes etapas consistentes en el diseño y la implementación.

3.2 Contexto

Tal como se indica en secciones anteriores de este trabajo, AutoMind provee un servicio de B-Learning en varios colegios. En cada institución que contrata el servicio, se instala un servidor que aloja los distintos sistemas educacionales (como SAGDE) y las bases de datos correspondientes.

En las bases de datos se encuentra información de los alumnos que hacen uso del sistema y también datos de la institución contratante. Se almacena información de diverso tipo como por ejemplo datos personales, datos correspondientes al uso de los programas y datos de configuración de los distintos programas.

El modelo educativo está basado en el aprendizaje mediante la ejercitación en los sistemas educacionales. Se realiza trabajo en el laboratorio de los colegios, ejecutándose en promedio diez ejercicios por cada bloque de una hora pedagógica (cuarenta y cinco minutos).

Cada ejercicio que se realiza está etiquetado con información clave que permite hacer seguimiento del progreso de los estudiantes, de un curso o de la institución en una materia en particular. Dentro de la información se reconocen tres elementos principales:

- Sector educativo: Es el análogo a un ramo o asignatura, como por ejemplo, matemáticas.
- Eje educativo: Corresponde a distintas áreas de conocimiento que se comprenden dentro de un determinado sector educativo. Es así como un sector está compuesto por un conjunto de ejes. Complementando el ejemplo anterior, el sector de matemáticas está compuesto por los ejes de geometría, probabilidades y álgebra, entre otros.
- Contenidos mínimos obligatorios (CMO): Basados en el programa educativo entregado por el Ministerio de Educación que específica Contenidos Mínimos Obligatorios para un determinado curso y asignatura. En el caso de AutoMind, cada CMO está asignado a un nivel educativo (curso) y a un eje educativo específico (y por consiguiente a un sector).

Además de la información anterior se distinguen ejercicios hechos como ‘Tarea’ y ‘Prueba’, la fecha en que se realizaron, el puntaje correspondiente y los errores cometidos.

Recientemente se agregó al sistema SAGDE una opción que permite a los estudiantes solicitar la ayuda del profesor desde el mismo programa (a estas solicitudes de ayuda se les llamará de ahora en adelante simplemente ayudas). Si el alumno no solicita ayuda, pero no progresa, el sistema avisa a los profesores y también se genera una instancia de ayuda.

3.3 Estudio y selección de la información

Durante las semanas tres y cuatro del trabajo se definió cuál sería la información de interés para ser usada en el framework. Esta información será la que dará paso a indicadores que a su vez darán paso, a través del framework, a retroalimentación que se entregará al usuario.

La selección de información de interés se hizo teniendo en consideración principalmente su transversalidad. La información debe ser válida para cualquiera de los sistemas educacionales de AutoMind, esto para reforzar el hecho de que con este trabajo se busca establecer un punto de partida común para los mismos. Otro factor a considerar fue que se busca que los usuarios reciban incentivos intrínsecos a la actividad. Por este motivo, se orienta la retroalimentación (y por consiguiente la información necesaria para presentarlo) a elementos relativos al rendimiento y progreso del estudiante.

Esta etapa del trabajo se llevó a cabo en forma conjunta con el profesor Roberto Araya, investigador asociado del Centro de Investigación Avanzada en Educación. El profesor cuenta con experiencia en el área de informática educativa y ha presentado trabajos relativos a elementos motivacionales en aprendizaje. Es autor también del libro Inteligencia Matemática. A continuación se explica cuáles fueron los elementos que se consideraron.

3.3.1 Información Histórica

En primer lugar se considera la información histórica del alumno. Se distinguieron los siguientes elementos como claves para obtener información.

1. Desempeño: Se decidió dar énfasis en el desempeño del estudiante por CMO y por modo (Tarea o Prueba). Por desempeño se entiende qué tan correcto es el trabajo que está desarrollando el estudiante. Se considera necesario el agregar una atenuación en los requisitos para mostrar retroalimentación a los usuarios con resultados más bajos, de manera que reciban mensajes de motivación.
2. Esfuerzo: Corresponde a cuántos ejercicios ha realizado el estudiante sin importar su resultado. Esta información se agrega con el objetivo de reflejar qué tanto está tratando de realizar ejercicios el estudiante.
3. Cobertura: Refleja cuantos CMOs distintos ha trabajado el estudiante.
4. Ayudas: Se decidió importante agregar información correspondiente a las ayudas que el usuario ha solicitado/recibido para un determinado CMO.
5. Retroalimentación entregada: Se consideró que es importante conocer cuál ha sido la impresión que ha tenido el usuario de los mensajes que se le han presentado. Esto se hace con el objetivo de darle un grado de personalización a la retroalimentación que recibirá en el futuro.

3.3.2 Información de la sesión

Análogamente a lo hecho para la información histórica, se definieron elementos importantes a considerar que son propios de una sesión de trabajo. Los elementos son básicamente los mismos: Desempeño, esfuerzo, ayudas y retroalimentación. Se reconoció que para este nivel es más importante definir requerimientos propios para cada sistema, teniendo siempre por objetivo desarrollarlos usando el framework. Considerando esto, se decidió que cualquier avance en este punto se encuentra fuera del alcance de este trabajo.

3.3.3 Estrategias motivacionales

Una vez identificada la información de interés se pensó en cuales iban a ser las estrategias para presentarlas. Se definieron dos estrategias que agrupan varios tipos de posibles mensajes motivacionales. Estas estrategias abarcan elementos de información tanto histórica como de la sesión.

La primera estrategia consiste en destacar el desempeño propio del estudiante. El enfoque en estos mensajes se da hacia el rendimiento por CMO. Los mensajes agrupados en esta estrategia corresponden a:

- Desempeño en un CMO: Muestra un mensaje aludiendo al buen rendimiento del estudiante en un determinado CMO. Este tipo de mensajes puede ser extendido para ejes y/o sectores.
- Crecimiento propio del desempeño en un CMO: Felicita al estudiante por las mejoras que puede haber presentado con el tiempo en el desarrollo de ejercicios.
- Aciertos consecutivos en ejercicios: Muestra un mensaje al estudiante diciéndole que ha realizado una serie de ejercicios de forma correcta.

La segunda estrategia consiste en comparar el desempeño del estudiante respecto a sus pares. Hay que destacar que existe un balance delicado en este punto: el objetivo de motivar a los estudiantes debe conjugarse con el que no sean objeto de burla por parte de los demás. Se hizo especial énfasis en que los mensajes que usen esta estrategia deben ser presentados siempre de forma positiva. Los mensajes agrupados en esta estrategia corresponden a:

- Desempeño en un CMO comparado con terceros.
- Crecimiento histórico comparado con terceros.
- Desempeño en un ejercicio en particular.
- Cobertura respecto a terceros.

3.3.4 Presentación de la retroalimentación y recepción por parte del usuario

En este punto se definieron algunos lineamientos para la presentación de los mensajes motivacionales. Se abordaron elementos acerca de la frecuencia en que se presentan y la manera de saber si están siendo efectivos. Se decidió:

- Los mensajes motivacionales deben obstruir lo menos posible a los elementos importantes y/o necesarios para poder ocupar los sistemas y realizar las actividades. Esto basado en el principio de diseño de interfaces llamado visibilidad.
- Los mensajes no deben ser ni muy frecuentes ni repetitivos.

- Los mensajes deben ser presentados ojalá en momentos de pausas o donde el usuario no sea distraído por ellos.
- Es necesario consultar al usuario sobre la retroalimentación que recibe de forma que en el futuro se pueda conocer la efectividad de los mismos.
- No se debe consultar muy seguido al usuario de forma de no saturarlo ni quitarle el foco de las actividades pedagógicas.
- Se debe llevar un registro de los mensajes presentados especificando: Usuario objetivo, sistema en que se originó, sector educativo, eje educativo, CMO, mensaje usado, estrategia motivacional, evaluación del usuario (si es que se le consultó) .

4 Segunda etapa: Diseño

4.1 Descripción

La segunda etapa corresponde al diseño de la estructura de clases de la API. Esta etapa se llevó a cabo inmediatamente después de la etapa de toma de requisitos. De dicha etapa se extrajeron elementos a ser considerados en el diseño.

El diseño se realiza de forma iterativa, con sub-etapas consistentes en un diseño (o rediseño, según corresponda) de la estructura de clases. Al final de cada una de estas etapas se evalúa si el desarrollar una regla y usar una fuente de datos es demasiado costoso en términos de implementación. Cabe recordar que antes de este trabajo ya existían implementaciones de 'interfaces empáticas' en proyectos de la empresa. Dado esto, evaluar el costo de implementación de las 'interfaces empáticas' permite ver no sólo la calidad del diseño, sino también la reducción de los costos de implementación respecto a las soluciones existentes. El diseño se lleva a cabo haciendo uso del paradigma de programación orientada a objetos y de diversos patrones de diseño, con el objetivo de que el código que se desarrolle en la etapa de implementación sea de fácil mantención y extensión.

4.2 Elementos a considerar en el diseño

4.2.1 Objetivos del framework

El objetivo principal del framework es la entrega de retroalimentación para estudiantes que utilizan software educativo. La retroalimentación que se presenta debe estar basada en información de rendimiento y uso del software educativo por parte de estudiantes. Para lograr esto, se interpreta la información disponible de los usuarios y se evalúan indicadores con ella. Sobre estos indicadores se verifica que se cumplan condiciones, priorizándose una de ellas. Finalmente se presenta retroalimentación basada en la condición seleccionada. De este objetivo se extraen dos elementos a considerar en el diseño:

1. El framework debe proveer un mecanismo para evaluar los distintos indicadores y condiciones sobre éstos para luego priorizarlas.
2. Dado que el mecanismo para priorizar las condiciones sobre los indicadores no tiene por qué ser fijo, el framework debe permitir remplazar este mecanismo por otro.

4.2.2 Necesidades de información

En la etapa de Primera etapa: Identificación de información educacional de interés, se pudo apreciar la necesidad de contar con mecanismos para la extracción de información. Dicha información, a su vez, fue clasificada en dos tipos: histórica y de sesión. Ambos tipos tienen características distintas que se resumen en la Tabla 4-1.

Tabla 4-1: Comparación de información histórica y de sesión

Característica	Información Histórica	Información de Sesión
Fuente de datos	Totalmente externa.	Interna para datos propios. Externa para datos de terceros.

Durante una sesión de trabajo, se revalúa	Nunca.	Constantemente.
Persistencia	Persistente, en los servidores de la empresa.	Persistente sólo para aquella que se guardará en los servidores.
Requiere acceso a fuente externa	Totalmente.	Parcialmente.

Se puede notar que la información histórica se obtiene siempre desde una fuente externa al programa en ejecución y que no necesita ser recalculada durante la sesión. Estas características hacen necesario que el framework provea un mecanismo para conectarse a una fuente de datos externa y que, de manera menos importante, permita almacenar la información que extraiga de dichas fuentes. De esta forma se evita solicitar reiteradas veces información que no variará.

Respecto a la información de sesión se extrae que, al igual que en la información histórica, existe la necesidad de que el framework provea facilidades de conectarse a una fuente de datos externa. Sumado a esto, se tiene que parte de la información se obtiene sin necesidad de una conexión como la del punto anterior, por lo que es deseable que el framework permita ser ocupado sin requerir el uso de una fuente externa.

4.2.3 Resumen

Tomando en cuenta todos los puntos expuestos en un comienzo, se procedió a definir tres requerimientos indispensables para el framework:

1. El framework debe permitir evaluar indicadores con la información disponible de los usuarios.
2. El framework debe proveer un mecanismo para verificar y priorizar condiciones sobre los indicadores.
3. El framework debe permitir conectarse a fuentes de datos externas.
4. El framework debe permitir poder funcionar sin conexión.

4.3 Arquitectura del sistema

4.3.1 Elementos principales del framework

Dentro del proceso de selección y presentación de retroalimentación para el usuario se distinguen cuatro elementos principales: la retroalimentación que se le da al usuario; la información del usuario y de donde se obtiene ésta; la evaluación de los indicadores que se pueden extraer de la información y la evaluación de las condiciones sobre éstos; y el mecanismo para seleccionar la retroalimentación que se dará al usuario, basándose en la información del mismo. El framework debe ser capaz de representar a dichos elementos y sus interacciones.

En trabajos anteriores desarrollados en la empresa, estos cuatro elementos se han identificado como: mensaje empático, fuente de datos, reglas empáticas y criterios empáticos. No se sabe si estos nombres tienen un origen específico en la literatura y al momento de realizar este

trabajo dicho origen no se encontró. Por simplicidad, de aquí en adelante, se les denotará como mensaje, fuentes de datos, reglas y criterios.

4.3.2 Arquitectura del framework

Para desarrollar el framework se considera una arquitectura basada en la metáfora del pizarrón. Esta decisión se toma considerando las diversas similitudes que existen entre el problema a resolver y esta arquitectura. A continuación se presentan los alcances de cada una de las componentes del sistema y las razones por las cuales una arquitectura de pizarrón es útil tanto para cumplir este alcance como para poder extenderlo en el futuro.

En primer lugar, se necesita evaluar distintas reglas para mostrar retroalimentación, como por ejemplo "progreso del jugador en el ranking respecto al último turno". Estas reglas se calculan basándose en indicadores que se obtienen con la información propia de la aplicación u otra fuente externa. A su vez, una regla retorna un valor (indicador). En principio, ninguna de estas reglas necesita interactuar con los demás, aunque sí pueden compartir información con otras. La labor anterior puede considerarse análoga a la que realiza una fuente de conocimiento y a su índice de relevancia. Una arquitectura de pizarrón permite agregar nuevas fuentes de conocimiento, en este caso reglas, al sistema sin afectar a las anteriores. Es, además, suficientemente dinámica como para que en el futuro dichas fuentes tengan más capacidad de acción, pudiendo inclusive ser agentes por sí mismas.

En segundo lugar está la necesidad de una fuente de datos interna de la aplicación. Esta fuente de datos se utiliza para manejar información propia de la ejecución del programa, como por ejemplo la posición del jugador en el ranking o las respuestas correctas durante una prueba. Hay diversos motivos para desear tener una fuente interna común de datos:

1. Muchas de las fuentes de conocimiento (reglas en este caso) comparten información.
2. Se puede evitar redundancia en la información, previniendo evaluaciones erróneas en las fuentes de conocimiento.
3. Se requiere menos espacio en memoria para almacenar los datos, esto es especialmente importante si se busca utilizar el sistema en equipos móviles.

Este rol puede ser cumplido por un pizarrón como el de una arquitectura del mismo nombre. El pizarrón permite ser usado además como un medio de coordinación entre fuentes de conocimiento. Si bien, la coordinación entre reglas no es un objetivo, la arquitectura de pizarrón lo soporta.

Finalmente, en este trabajo la única componente que realmente toma decisiones es aquella que identifica cuál de las reglas es la óptima para mostrar una retroalimentación. En la toma de esta decisión se pueden definir varios criterios para escoger la regla más adecuada. Este proceso es análogo al que realiza un árbitro en un pizarrón respecto a las distintas fuentes de conocimiento. Una vez que el árbitro escoge una regla, se le consulta (a la regla) cuál es la retroalimentación que se debe mostrar. Dado que se quiere aportar a la motivación de los usuarios de los sistemas, se espera que la retroalimentación sea variada y poco repetitiva. El criterio para elegir la fuente de conocimiento, entonces, no debiese ser trivial ni fácil de inferir.

4.3.3 Arquitectura del sistema en producción

El sistema en producción debe no sólo ser capaz de manejar una memoria interna. En algunos casos la información que se requiere no está disponible al interior de la aplicación. En otros casos el volumen de información se torna inconveniente como para hacer una copia completa en la memoria del cliente y/o enviarlo por la red a todos los clientes. Para este tipo de situaciones es necesario que el sistema sea capaz de solicitar información a fuentes de datos externas.

Se puede elegir entre varios enfoques para la conexión y extracción de los datos. Uno de los enfoques es el de conexión directa a las bases de datos, el otro es envolver el proceso de la consulta a la base de datos en un servicio web. Se opta por esta segunda opción principalmente para no duplicar trabajo. Actualmente no existe forma en Android de, por ejemplo, conectarse directamente a una base de datos Postgresql. Dado que se busca que el sistema sea compatible con equipos móviles es necesario buscar otro enfoque. El problema de la base de datos se aborda generalmente por medio de la implementación de un servicio web que conteste a las consultas. Si se desarrolla dicho servicio para equipos móviles se puede reutilizar para las plataformas fijas. En el anexo se adjunta un esquema representando esto.

4.4 Primer diseño

En la sección 4.3.1-Elementos principales del framework se distinguen los elementos principales del presente trabajo. Estos elementos se reflejaron en la distribución de paquetes al interior del framework que se presenta en las siguientes subsecciones.

4.4.1 Paquete Message

Uno de los elementos identificados en la sección 4.3.1 es el de la retroalimentación que se entrega a los usuarios. El paquete *message* agrupa los elementos necesarios para poder presentar estos mensajes y se compone de tres clases (ver Figura 4-1):

- AbstractEmpathicMessage: La clase base para los mensajes.
- EmpathicMessageArgs: La clase que representa la información necesaria para construir una instancia de la clase `AbstractMessage`.
- EmpathicMessageFactory: La clase encargada de la instanciación de objetos de tipo `AbstractEmpathicMessage`. Para la construcción del mensaje usa como argumento un objeto de tipo `EmpathicMessageArgs`.

4.4.2 Paquete Rule

En la sección 4.3.1 se identificó como una componente importante la evaluación de los indicadores que se pueden extraer de la información y la evaluación de las condiciones sobre éstos. El paquete *rule* contiene las clases relativas a las reglas. Estas clases son sólo dos (ver Figura 4-2):

- AbstractEmpathicRule: La clase base para las reglas.
- EmpathicRuleFactory: La clase encargada de la instanciación de objetos de tipo `AbstractEmpathicRule`. Para la construcción del mensaje usa como argumento un `String`. Esta *factory* está pensada para usar un `Map<String, AbstractRule>` y retornar una única

instancia de una regla que esté asignada a un 'nombre'. Esta decisión se justifica en el hecho de que cada regla es individual y no tiene mayor sentido tener dos instancias de una misma regla durante la misma ejecución. Esto básicamente porque el resultado de su evaluación no debiese ser diferente.

Figura 4-1: Estructura de clases del paquete message

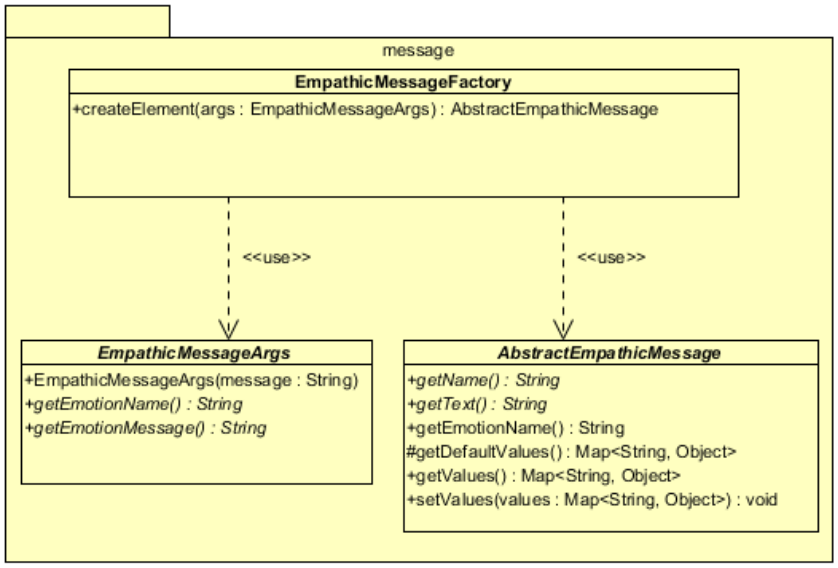
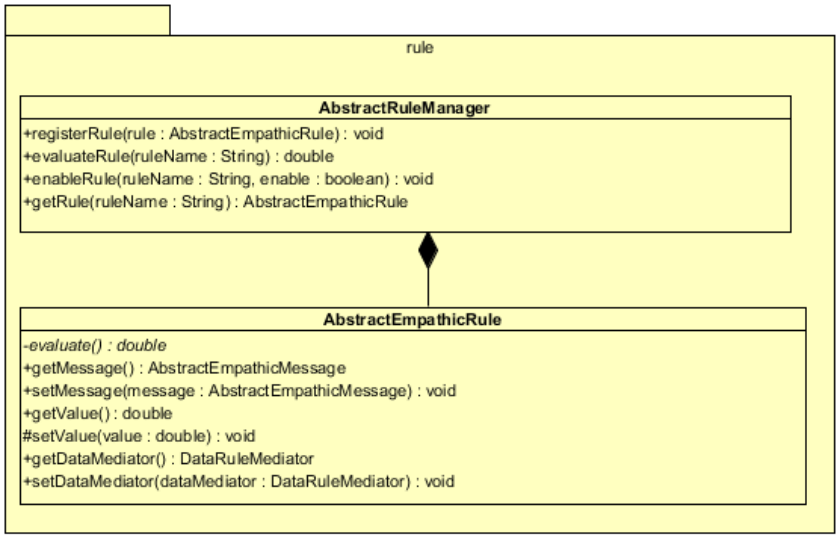


Figura 4-2: Estructura de clases del paquete rule



4.4.3 Paquete Arbiter

El paquete *arbiter* contiene los mecanismos para seleccionar la retroalimentación que se dará al usuario, basándose en la información del mismo, elemento identificado en la sección 4.3.1. Se compone de las clases *Arbiter* y *ArbiterCriterion* (ver Figura 4-4).

- Arbiter: La clase encargada de determinar cuál es la regla adecuada para presentar los datos. El funcionamiento de esta clase está pensado para ser principalmente mediante el patrón de diseño *Template Method*.
- ArbiterCriterion: La labor de seleccionar la regla se lleva a cabo por medio de un `ArbiterCriterion`. Está pensado que los objetos de este tipo se comuniquen con un objeto de tipo `Arbiter` usando el patrón de diseño *Visitor*. De esta forma no es necesario cambiar la ejecución del `Arbiter` y se pueden designar nuevos criterios para la selección de reglas.

4.4.4 Paquete Data

El paquete más grande del diseño es el referente a las fuentes de datos. Este paquete incluye los mecanismos para poder almacenar información de forma interna y consultar a fuentes de datos externas. Se compone de una interfaz y cuatro clases principales (ver Figura 4-3):

- IDataSource<T>: Corresponde a la interfaz para una fuente de datos. Provee métodos para acceder a los valores y modificarlos. Se hace uso de *generics* para aceptar datos de distintos tipos, tal como se hace con clases como Vector. Un IDataSource<T> sólo acepta datos de tipo `T`.
- WebDataSpace<T>: Una implementación de un `IDataSource<T>` para la comunicación con fuentes externas. Como se explica anteriormente la idea es no requerir de mecanismos de conexión a bases de datos pensando en el uso del framework en equipos móviles.
- LocalDataSpace<T>: Una implementación de un `IDataSource<T>` para el almacenamiento de objetos del tipo T dentro de la memoria de la aplicación.
- DataEntry<T>: Los datos que se almacenan en una implementación de `IDataSource<T>` se hacen usando *boxing* en un `DataEntry<T>`. Esta forma de almacenar los datos permite agregar metadata, como *ids* y/o versiones del objeto, los cuales no tiene por qué conocer el cliente y que se pueden usar para administración dentro de los `IDataSource<T>`.
- AbstractDataManager: La clase encargada de manejar los distintos `IDataSource<T>` y de manejar las peticiones que se realizan a éstos, ya sea de agregar, modificar y/o eliminar entradas. Esta clase está pensada para ser utilizada como una *Factory*. Al igual que en el caso de `EmpathicRuleFactory`, esta *factory* está pensada para usar un `Map` (en este caso `Map<String, IDataSource<?>>`) y retornar una única instancia de una fuente de datos. Esto pues se supone que no se necesitan dos fuentes de datos idénticas.

Figura 4-3: Estructura de clases del paquete data

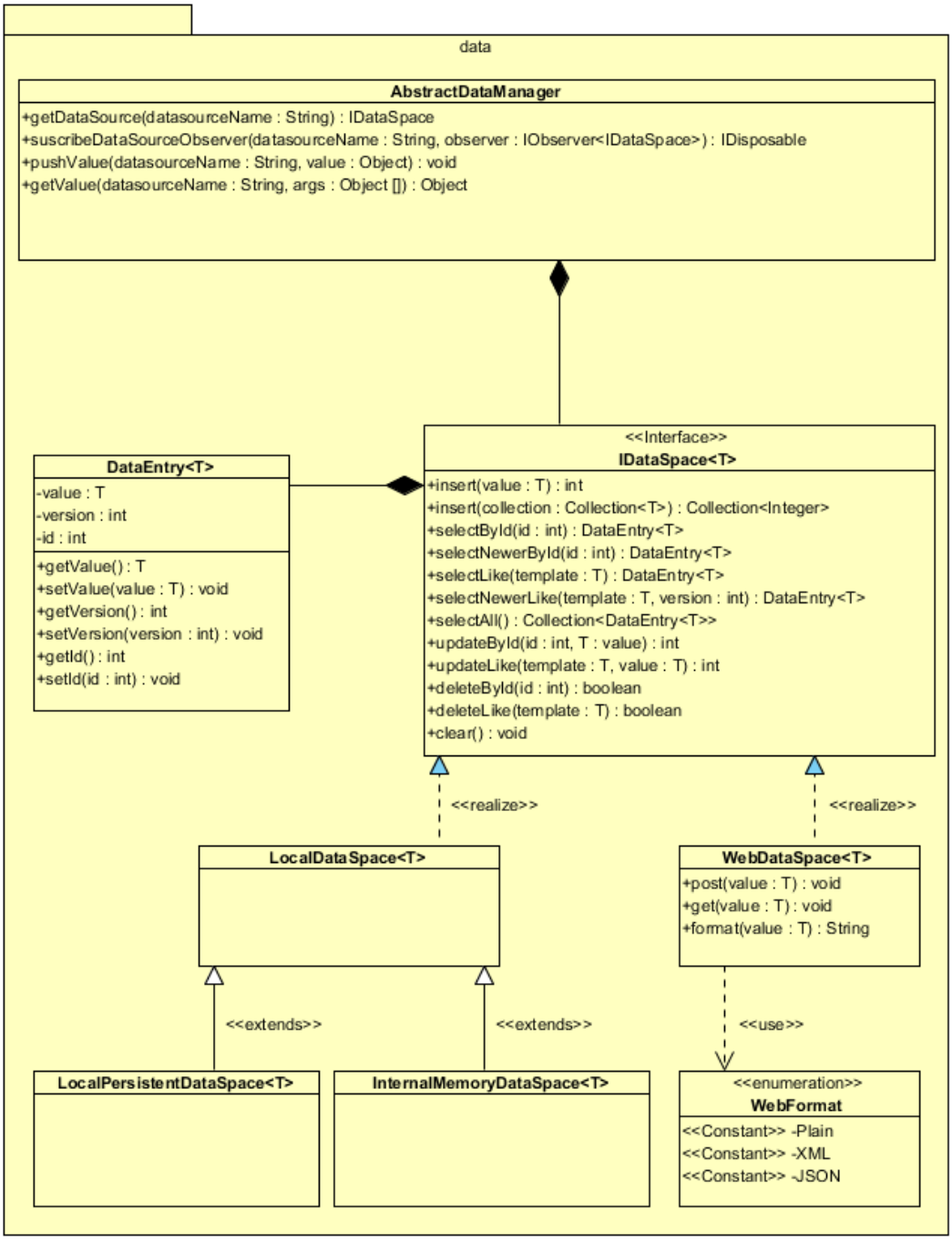
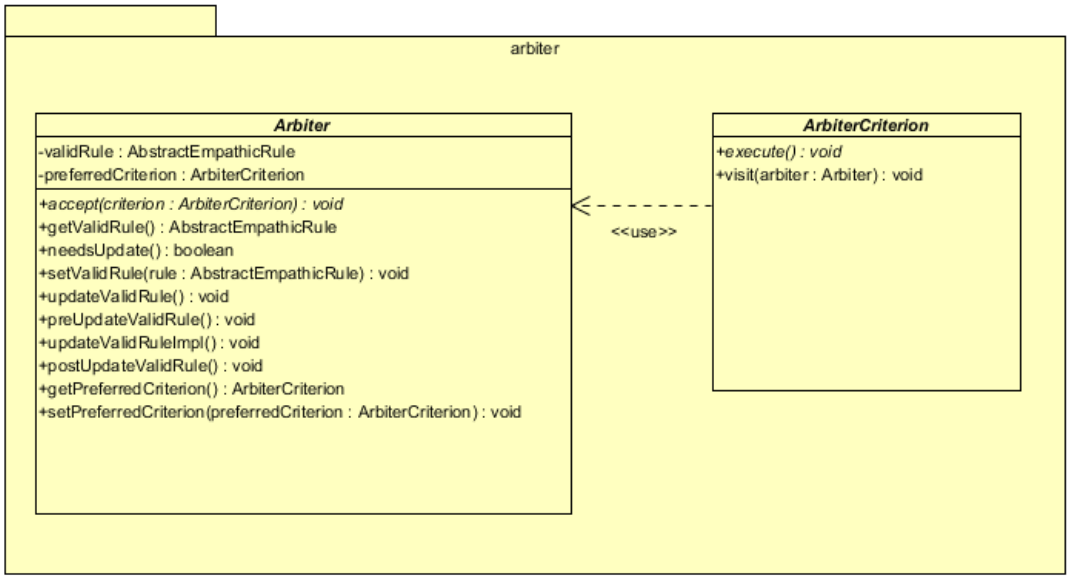


Figura 4-4: Estructura de clases del paquete arbiter



4.5 Segundo diseño

El primer diseño realizado de este trabajo fue evaluado respecto al costo asociado a la implementación de 'reglas empáticas'. Se decidió que algunos elementos eran redundantes o innecesarios y que otros necesitaban ser rediseñados y/o agregados. Dichas decisiones conllevaron los cambios en el diseño que se detallan a continuación, en las siguientes subsecciones.

4.5.1 Elementos eliminados

1. Dentro del paquete *Message* se decidió eliminar las clases `EmpathicMessageFactory` y `EmpathicMessageArgs`. Esta decisión se justifica en el hecho de que cada regla (`AbstractRule`) tiene asociado un mensaje. No existe necesidad de que la solicitud de creación de mensajes se pase a una *Factory* si la relación `AbstractMessage – AbstractRule` es 1 a 1. Eliminar `EmpathicMessageFactory` tiene como consecuencia que ya no es necesario usar el objeto de tipo `EmpathicMessageArgs`. Este tipo se pasaba a `EmpathicMessageFactory` como parámetro en la construcción de un `AbstractMessage`.
2. Se eliminan los métodos `selectLike`, `selectNewerLike`, `selectNewerById`, `updateLike` y `deleteLike` de la interfaz `IDataSource<T>`. Estos métodos se remplazan por un nuevo parámetro que indica el tipo de consulta que se está realizando.

4.5.2 Elementos modificados

1. Cambia la firma del método `getMessage` de la clase `AbstractRule`, ahora retorna un objeto de tipo `AbstractMessage` y no un `String`. Se elimina con esto un paso intermedio innecesario en la instanciación de los mensajes.
2. Se cambia la estructura de las reglas (`AbstractRule`). En un principio se consideraba que una regla podía tener dos estados distintos: 'Evaluable' y 'No Evaluable'. Esto se determina mediante el método `canEvaluate`. Se decide agregar un tercer estado:

'Seleccionable'. Una regla se dice seleccionable si es evaluable y el resultado de su evaluación la hace elegible para presentar un mensaje de retroalimentación.

3. La clase `AbstractEmpathicKernel` ya no tiene campos de tipo `AbstractRuleManager`, `AbstractDataManager` ni `AbstractUiManager`. Estos campos se agrupan dentro de una nueva clase: `Managers`.

4.5.3 Elementos nuevos

La mayor parte de los cambios del diseño se encuentran en la categoría de elementos nuevos.

1. Se agrega la clase `Managers` para envolver a los objetos de clase `AbstractRuleManager`, `AbstractDataManager` y `AbstractUiManager` (ver Figura 4-5: La clase EmpathicManagers).
2. Se agrega la opción de pasar un `Map<String, Object>` a un objeto de tipo `AbstractMessage`. Este `Map<String, Object>` se usa para reemplazar palabras claves (*keywords*) por un valor específico. Dado un `Map.Entry<String, Object>` en el `Map<String, Object>` se remplazan las apariciones del `String` *key* (antecedido por el carácter '@') por `value.toString()`. Por ejemplo: Si se tiene que, key = "valor" y value.toString() = "texto reemplazado". Al hacer el remplazo sobre el mensaje "El texto es: @valor", se obtiene: "El texto es: texto reemplazado".
3. Se agrega la clase `DataRuleMediator` aplicando el patrón de diseño *Mediator*. El objetivo de esta nueva clase es lograr que la interacción entre las fuentes de datos (`IDataSource<T>`) y las reglas (`AbstractRule`) esté encapsulada. Se evita con esto que las reglas necesiten conocer información adicional para su comunicación con los datos.
4. Se agrega a los objetos de tipo `AbstractRule` una variable de instancia de tipo `DataRuleMediator`.
5. Se agregó un sistema de *plugins* para el framework. Los objetos de tipo `AbstractEmpathicKernel` deben implementar el método `loadPlugins` que retorna una colección de *plugins* (`Collection<AbstractEmpathicPlugin>`). Un *plugin* (`AbstractEmpathicPlugin`) se compone de métodos que retornan elementos que se usan para constituir el framework. Estos objetos son:
    a. Objetos de tipo `AbstractRuleManager`, `AbstractDataManager` y `AbstractUiManager`.
    b. Objetos de tipo `Arbiter` y `ArbiterCriterion`.
    c. Una colección de reglas (`Collection<AbstractRule>`).
    d. Una colección de fuentes de datos (`Collection<IDataSource<?>>`).
6. Se agregó la anotación `RuleMetadata` para indicar *metadata* correspondiente a una regla. El framework debe procesar esta *metadata* y administrarla con un objeto de tipo `AbstractRuleManager`, con el fin de configurar automáticamente un objeto de tipo `AbstractRule` (ver Figura 4-6: La anotación RuleMetadata).
7. Se provee una línea de ejecución predeterminada. Esta forma de ejecución se basa en el uso de memoria interna para almacenar los datos y no requiere más configuración que iniciar un objeto de tipo `DefaultEmpathicKernel` y registrar las reglas. El criterio de selección de reglas (`ArbiterCriterion`) predeterminado es uno que ocupa la regla en estado 'seleccionable' menos ocupada (`LessUsedCriterion`).

Figura 4-5: La clase EmpathicManagers

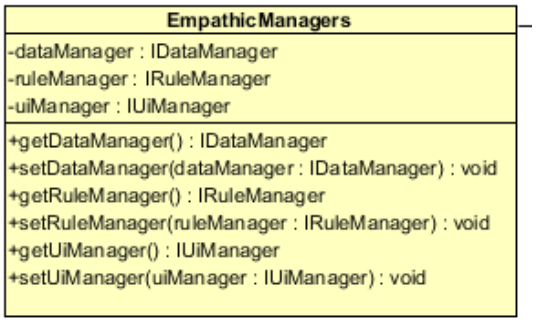
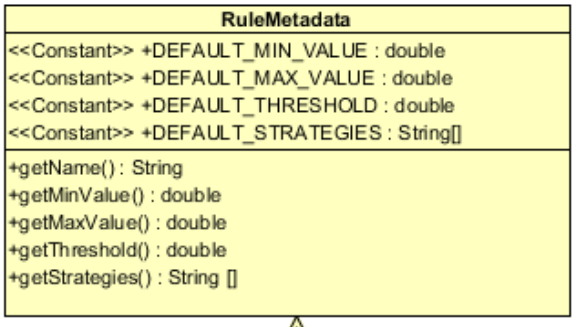


Figura 4-6: La anotación RuleMetadata



4.6 Prototipo

Entre la segunda y tercera fases de diseño, se implementó un prototipo del framework con el objetivo de ser presentado en la empresa. De dicha ocasión se obtuvo retroalimentación que se utilizó para hacer mejoras al segundo diseño. En primer lugar se solicitó una forma más unificada de realizar las distintas acciones en una fuente de datos. Finalmente, uno de los ingenieros de la empresa solicitó el soporte para bases de datos relacionales, mencionando que algunas de las aplicaciones actualmente usan HSQLDB. Para más detalles de este punto, ver sección 5.1.

4.7 Tercer diseño

La tercera etapa de diseño del framework fue llevada a cabo de forma posterior a la presentación del prototipo del mismo. El diseño realizado es el diseño final del framework antes de pasar a la etapa de implementación definitiva. Es importante destacar que para esta etapa el prototipo fue presentado en la empresa. Durante la presentación, se mostraron las distintas funcionalidades, además de algunos ejemplos y su código fuente. Los demás ingenieros que trabajan en desarrollo solicitaron que para la versión de escritorio se agregara soporte para HSQLDB, una base de datos relacional escrita en java. Se decidió agregar soporte para SQL.

4.7.1 Elementos eliminados

No se eliminaron elementos durante esta etapa de diseño.

4.7.2 Elementos modificados

Diversas clases abstractas dejaron de ser el comienzo de la jerarquía y fueron remplazadas por interfaces. Las clases existentes desde este momento implementan dichas interfaces. El rol de

las clases abstractas pasó a ser el de clases auxiliares que tienen como objetivo reducir el costo de implementación de los distintos elementos, poseyendo un conjunto de métodos ya implementados. Las clases modificadas fueron: `AbstractRule`, `AbstractRuleManager`, `AbstractDataManager`, `AbstractUiManager`, `Arbiter`, `ArbiterCriterion`.

4.7.3 Elementos nuevos

1. A petición del cliente se agregó soporte para fuentes de datos de tipo SQL. Dada la extensión de este cambio se agrega más información en una sección aparte.
2. Se agrega un criterio unificado para realizar consultas a los distintos tipos de fuentes de datos. Dada la extensión de este cambio se agregan los detalles en una sección aparte.
3. Se agregaron interfaces al comienzo de la jerarquía para remplazar a algunas de las clases abstractas existentes: `AbstractRule` ahora implementa `IRule`; `AbstractRuleManager` implementa `IRuleManager`; `AbstractDataManager` implementa `IDataManager`; `AbstractUiManager` implementa `IUiManager`; `Arbiter` implementa `IArbiter`; `ArbiterCriterion` implementa `IArbiterCriterion`.

4.7.4 Soporte para SQL

Durante la presentación del prototipo del framework en la segunda etapa, surgió la solicitud por parte de la gente de la empresa de agregar soporte para HSQLDB. Al principio del diseño se había decidido dejar de lado el desarrollo para SQL considerando que los dispositivos móviles en general no contaban con los drivers necesarios para conectarse a bases de datos externas. Dado que se apreció interés en el uso de esta funcionalidad para los desarrollos, se decidió reincorporarla al diseño. (Para ver el modelo completo, ir a Figura 4-7: Diagrama de clases para consultas SQL)

4.7.4.1 Elementos necesarios

Para poder dar un soporte para SQL que facilite el trabajo del desarrollador, se deben dar facilidades para definir cuatro elementos:

1. Información de conexión (Dirección del servidor, nombre de la base de datos, etc.)
2. Modelo de datos.
3. Consultas personalizadas.
4. (Opcionalmente) Generación automática de algunas consultas.

4.7.4.2 Enfoque utilizado

Se decidió dar un enfoque orientado al uso de anotaciones de Java. Este enfoque es usado por distintas librerías de mapeo objeto-relacional (ORM por sus siglas en inglés), como *Hibernate*. Se optó por utilizar anotaciones idénticas a las de *Hibernate*, principalmente por el amplio uso de este sistema de ORM, a fin de reducir el tiempo de aprendizaje e inclusive reutilizar código existente cambiando los *imports* respectivos.

4.7.4.3 Elementos agregados

Se agregaron tres interfaces al comienzo de la jerarquía:

1. ISqlDataSource<T>: Esta interfaz extiende `IDataSource<T>`, incluye tres métodos adicionales:
    a. executeNamedQuery: recibe un `String` con el nombre de una consulta personalizada, y un conjunto de pares nombre-valor. Ejecuta la consulta del nombre especificado y retorna una lista de objetos del tipo de la fuente de datos.
    b. parse: recibe un objeto de tipo `ResultSet` y retorna una lista de objetos del tipo almacenado en la fuente de datos.
    c. toPairs: recibe un objeto del tipo almacenado en la fuente de datos y lo convierte en una colección de pares nombre-valor.
2. ISqlConnector: Está orientada al manejo de la conexión a la base de datos.
3. ISqlConnectionInfo: Posee la información necesaria para conectarse a la fuente de datos.

Se agregaron además clases que implementan las interfaces anteriores:

4. AbstractSqlDataSource<T>: Implementa la mayor parte de `ISqlDataSource<T>`. Genera automáticamente las consultas para hacer inserciones, búsquedas, actualizaciones y eliminar elementos en las tablas. Además interpreta una anotación de tipo `SqlMetadata` (que se explica más adelante) y las consultas personalizadas que se especifican ahí. Se apoya en las clases `SqlDescriptor` y `QueryDescriptor` (que se describen más adelante).
5. DefaultSqlConnector: Implementa `ISqlConnector`.
6. AbstractSqlConnectionInfo: Implementa `ISqlConnectionInfo`. Provee la opción de cargar la información de la fuente de datos desde un objeto de tipo `Properties` (y por lo mismo desde un archivo de tipo .properties).

Se agregaron dos clases para manejar las consultas SQL (ver Figura 4-8: Clases descriptoras de consultas sql).

1. SqlDescriptor: Esta clase almacena las distintas consultas que posee un objeto de tipo `ISqlDataSource<T>`. Asocia un nombre de consulta a un determinado `QueryDescriptor`. De forma predeterminada, un `AbstractSqlDataSource<T>` cuenta con un `SqlDescriptor` con las consultas para realizar inserciones, búsquedas, actualizaciones y eliminar datos de las tablas en la base datos.
2. QueryDescriptor: Almacena la descripción de una consulta SQL. Para cada parámetro de una consulta, tiene un registro de la posición o las posiciones que le corresponden. Cuando un parámetro es pasado a la consulta, el `QueryDescriptor` automáticamente lo asigna en todas las posiciones correspondientes.

Se agregaron cinco anotaciones. Dos de estas anotaciones son para describir el modelo de datos, mientras que las tres anotaciones restantes permiten definir consultas personalizadas. Los elementos agregados son los siguientes:

1. Id: Marca un campo (variable de instancia) de un modelo de datos como el elemento que corresponde al id en su homólogo en el modelo relacional (en caso de existir). Permite personalizar el nombre de la columna, el predeterminado es ‘id’.
2. Column: Identifica un campo del modelo de datos como un elemento con un homólogo en el modelo relacional. Al igual que en el caso de `Id`, permite personalizar el nombre, si no se especifica uno, lo infiere del nombre de la variable de instancia.
3. NamedQuery: Permite definir una consulta con un nombre especifico.

4. NamedQueries: Es un envoltorio para entregar un arreglo de `NamedQuery` a un `SqlMetadata`.
5. SqlMetadata: Permite nombrar una `ISqlDataSource<T>`, y entregarle un `NamedQueries` que contiene un arreglo (posiblemente vacío de `NamedQuery`).

Figura 4-7: Diagrama de clases para consultas SQL

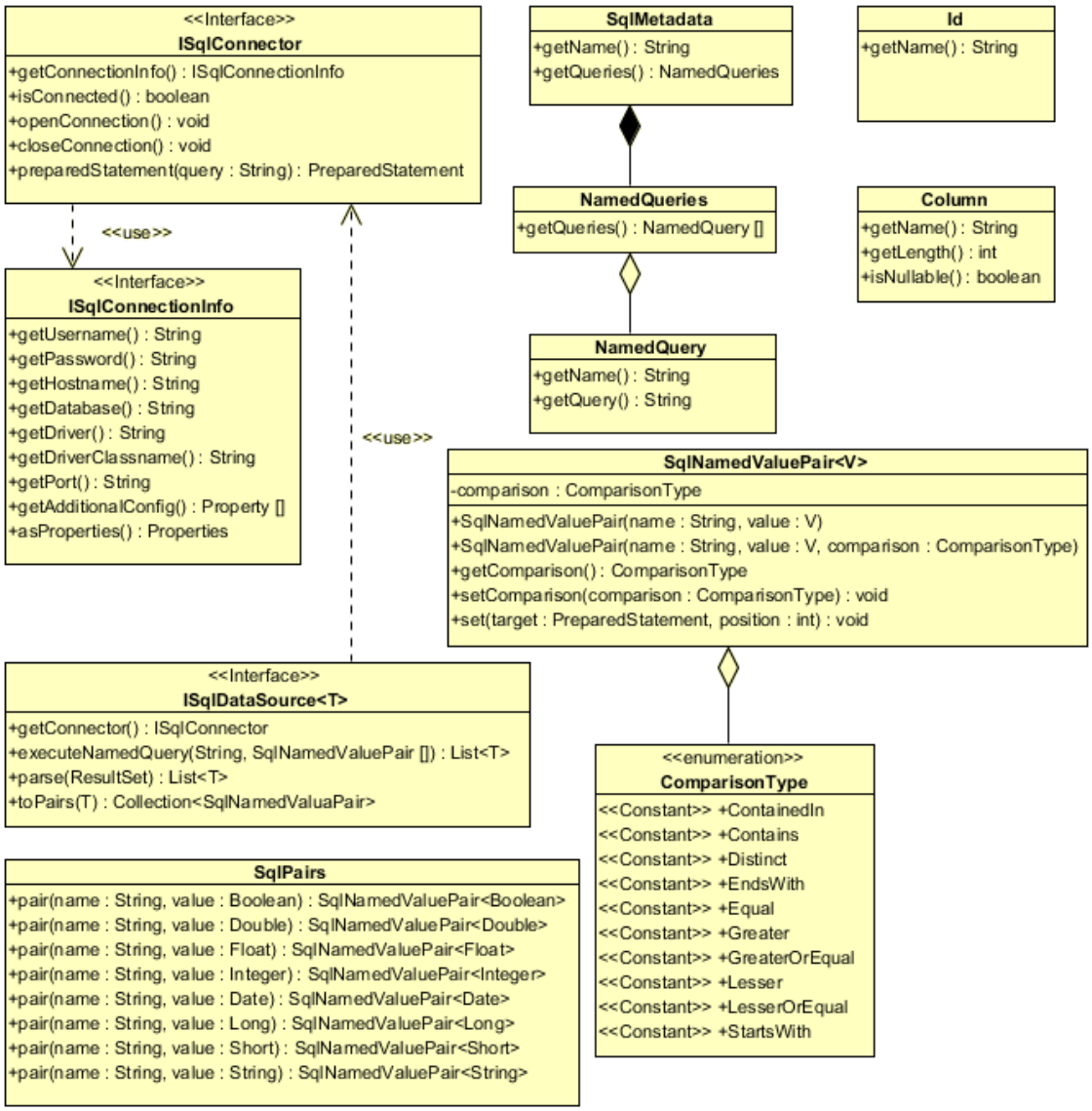
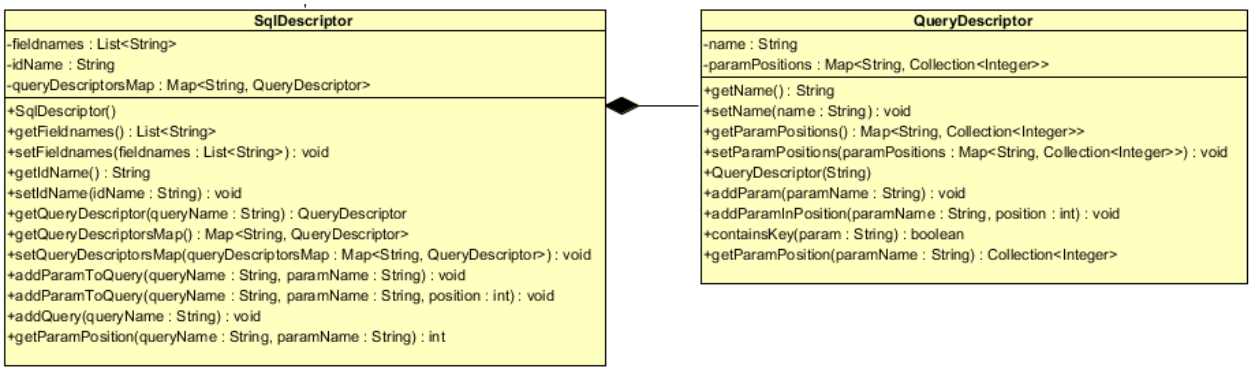


Figura 4-8: Clases descriptoras de consultas sql



4.7.5 Diseño unificado para hacer consultas a fuentes de datos

Dentro del diseño de las fuentes de datos, se considera que éstas pueden realizar todas o parte de las siguientes cuatro actividades: búsqueda, inserción, actualización y eliminación de datos. En un principio se decidió que para estas actividades se agregarían métodos diferentes para cada una de las formas de filtrar los datos, vale decir: por identificador del dato, por similitud, etc. Este enfoque hacía que implementar una fuente de datos requiriera sobrescribir muchos métodos que, además, recibían varios parámetros, lo que volvía más difícil el recordar la interfaz a la hora de desarrollar.

En la segunda etapa de diseño se decidió dar paso a un solo método por tipo de consulta, vale decir, un método para inserciones, uno para actualizaciones, etc. Dicho método tiene un parámetro para indicar la forma de filtrar los datos y un parámetro correspondiente a un arreglo de objetos. Esta forma de abordar el problema reduce la cantidad de métodos a implementar, pero aumenta considerablemente la complejidad de la implementación del método único.

En la tercera etapa de diseño se decidió mejorar la estructura de los parámetros que se dan a las consultas. Se crearon dos interfaces y una clase nueva: `IQueryOption<T>`, `IQueryCriterion<T>` y `NamedValuePair<T>`. La primera interfaz (`IQueryOption<T>`) almacena el tipo de filtro que se usará y un valor entero extra. Este valor se pasa para determinar un número identificador o la cantidad de elementos máximo que se quiere, por ejemplo. La segunda interfaz (`IQueryCriterion<T>`) representa un filtro. Los objetos a ser consultados deben pasar por este filtro para ser buscados, insertados, actualizados o eliminados. El método que filtra recibe como parámetro un arreglo de objetos de tipo `NamedValuePair<?>`. La clase `NamedValuePair<?>` almacena un par nombre – valor de un determinado tipo.

Existen leves diferencias en cómo se hace el filtro en los distintos tipos de fuentes de datos. En el caso de las fuentes de datos internas se hace uso directamente del objeto de tipo `IQueryCriterion<T>` y de su método `apply`. En las fuentes de datos web, todos los `NamedValuePair<?>` que se pasan se transforman en pares nombre valor como en una petición web estándar. En las fuentes de datos SQL, todos los `NamedValuePair<?>` se insertan en la consulta para la base de datos, por medio de un objeto de tipo `SqlDescriptor`. Se utiliza el nombre para determinar la posición dentro de la consulta.

4.8 Modelo de datos para el registro de eventos

Para medir la efectividad de los mensajes entregados es necesario llevar un registro de ellos y su evaluación. Es en este contexto que se diseñó un modelo de datos para almacenar los distintos eventos en que se presente un mensaje.

El modelo consiste de seis entidades distintas. Cuatro de estas entidades son indispensables, dos son opcionales y están ajustadas a las necesidades de la empresa. A continuación se presenta en detalle las entidades. Se adjunta un diagrama con el modelo de datos en los anexos.

4.8.1 Tipo de mensaje

La primera entidad representa los varios tipos de mensajes. El objetivo es poder diferenciar la estrategia educativa, o motivo por el cuál ese mensaje está siendo presentado. El modelo de datos se presenta en la Tabla 4-2: Modelo de datos - Tipos de mensajes.

Tabla 4-2: Modelo de datos - Tipos de mensajes

Entidad 1: empathy_types		
Nombre	Tipo	Descripción
type_name	Varchar(50)	El nombre del tipo de mensaje.
description	Varchar(150)	Descripción breve del tipo específico.

4.8.2 Fuente del mensaje

La segunda entidad representa las distintas fuentes posibles del mensaje. El objetivo es poder determinar específicamente cual fue la aplicación que lo originó. El modelo se presenta en la Tabla 4-3: Modelo de datos - Fuentes de mensajes

Tabla 4-3: Modelo de datos - Fuentes de mensajes

Entidad 2: empathy_sources		
Nombre	Tipo	Descripción
type_name	Varchar(50)	El nombre de la fuente del mensaje.
description	Varchar(150)	Descripción breve de la fuente del mensaje.

4.8.3 Mensajes

La tercera entidad representa las diversas instancias de un mensaje de retroalimentación. Se almacena el evento y la recepción (evaluación) del usuario. Esta tabla permite conocer cuánta retroalimentación han recibido los distintos usuarios y qué tan positivamente la han evaluado. El modelo de datos se presenta en la Tabla 4-4: Modelo de datos - Mensajes

Tabla 4-4: Modelo de datos - Mensajes

Entidad 3: empathy		
Nombre	Tipo	Descripción
id	Integer	Llave primaria. El identificador de la instancia de mensaje.
idsource	Integer	Referencia al campo id de empathy_sources. Identifica la fuente del mensaje.
Idtype	Integer	Referencia al campo id de empathy_types. Identifica el

		motivo del mensaje.
created_at	Date	La fecha en la que se presentó el mensaje al usuario.
evaluated	Boolean	Verdadero si con este mensaje se solicitó al usuario que dijera que le pareció éste.
answered	Boolean	Si se le solicitó evaluación al usuario, este valor es verdadero cuando contestó. Si no se le solicitó, se le solicitó pero no contestó o el mensaje desapareció en el intertanto este campo es falso.
liked	boolean	Verdadero si el usuario contestó que el mensaje le gustó. Es falso si: el usuario contestó que no le gustaba o el usuario no contestó (no se le preguntó, no alcanzó a contestar).

4.8.4 Información del mensaje

La cuarta entidad presenta información sobre los medios audiovisuales utilizados para presentar el mensaje al usuario. El modelo de datos se expone en la Tabla 4-5: Modelo de datos - Información del mensaje

Tabla 4-5: Modelo de datos - Información del mensaje

Entidad 4: empathy_info_message		
Nombre	Tipo	Descripción
idempathy	Integer	Referencia a una instancia de mensaje (campo id de empathy).
message	Varchar(250)	El mensaje o su descripción.
uses_text	Boolean	Verdadero si el mensaje muestra texto.
uses_audio	Boolean	Verdadero si el mensaje reproduce un sonido.
uses_image	Boolean	Verdadero si el mensaje presenta una imagen.
uses_video	Boolean	Verdadero si el mensaje muestra un video.

4.8.5 Información del usuario objetivo

La quinta entidad contiene información sobre el usuario al que se le presenta la retroalimentación. Su modelo se muestra en la Tabla 4-6: Modelo de datos - Usuario objetivo

Tabla 4-6: Modelo de datos - Usuario objetivo

Entidad 5: empathy_info_sagde_user		
Nombre	Tipo	Descripción
idempathy	Integer	Referencia a una instancia de mensaje (campo id de empathy).
idestablishment	Integer	Referencia a la entidad educativa a la que asiste el estudiante.
idcourse	Integer	Referencia al curso al cual pertenece el alumno.
iduser	Integer	Referencia al identificador del usuario en su institución educativa.

4.8.6 Información del origen del mensaje

La última entidad muestra información específica del origen del mensaje de retroalimentación. Esta información incluye varios elementos educativos: sector, eje, contenido

mínimo obligatorio, etc. Es de especial utilidad cuando se busca evaluar información respecto a un área educativa específica. El modelo de datos se expone en la Tabla 4-7: Modelo de datos - Información educativa.

Tabla 4-7: Modelo de datos - Información educativa

Entidad 6: empathy_info_sagde		
Nombre	Tipo	Descripción
idempathy	Integer	Referencia a una instancia de mensaje (campo id de empathy).
idsector	Integer	Referencia al sector educativo correspondiente a la actividad que originó el mensaje.
ideje	Integer	Referencia al eje educativo correspondiente.
idcmo	Integer	Referencia al contenido mínimo obligatorio correspondiente.
idobjcon	Integer	Referencia al objeto de conocimiento correspondiente.
idnivel	Integer	Referencia al nivel educativo correspondiente.

5 Tercera etapa: Implementación

Tras la tercera etapa de diseño se cuenta con un modelo de clases que se considera suficiente para ser llevado a producción, por lo que se dio paso a la etapa de implementación. Durante esta etapa se programan las clases de forma más completa que en los prototipos previos. La implementación se ajusta al diseño dado pero incluye elementos como enumeraciones y "azúcar sintáctico" para facilitar tareas recurrentes del trabajo. Esta etapa se divide en dos sub-etapas: separación del núcleo del framework, programación de clases con métodos pre-implementados. En esta sección se presenta además información más detallada del prototipo inicial para la empresa, desarrollado entre la segunda y tercera etapa de diseño (ver secciones 4.6, 4.5 y 4.7 respectivamente). Finalmente se muestra una visión general de las funcionalidades del framework al terminar la implementación.

5.1 Prototipos

Tan como se menciona en la sección 4.1, tras cada iteración del diseño de la estructura de clases se lleva a cabo un prototipo con el objetivo de verificar si se está cumpliendo con una característica mínima para seguir adelante en el proceso. Dicha característica es el esfuerzo en términos de implementación de implementar una regla utilizando el framework. Evaluar el costo de implementación de las 'interfaces empáticas' permite ver no sólo la calidad del diseño, sino que también observar la reducción de costos de implementación respecto a las soluciones existentes (ya existían otros desarrollos de 'interfaces empáticas al interior de la empresa). En las siguientes dos subsecciones se presentan los dos prototipos desarrollados.

5.1.1 Prototipo inicial

El prototipo desarrollado tras la primera etapa de diseño estaba basado en un programa muy sencillo. Esta aplicación de prueba permitía agregar una lista de amigos que podían o no comer golosinas. Se evaluaban dos reglas de prueba: cantidad de amigos inscritos mayor que dos, cantidad de amigos glotones mayor que dos. En esta etapa del trabajo se utilizaba un criterio aleatorio para definir cuál era el mensaje más adecuado para presentar retroalimentación.

En la Figura 5-1 se muestra la vista principal de la aplicación prototipo. Existen dos botones de interés: "Amigo" y "Probar Mensaje". El botón "Amigo" despliega una ventana que permite agregar un nuevo amigo a la lista de amigos. Como se puede ver en la Figura 5-3, en dicha ventana se puede marcar la opción de que el nuevo integrante come golosinas, lo que hace posible diferenciar entre amigos. El botón "Probar Mensaje" permite desplegar el mensaje que el framework presentaría en un contexto real de uso de la aplicación. En la Figura 5-2 se presenta un mensaje de prueba vacío. Este mensaje aparece cuando ninguna de las reglas especificadas es adecuada para mostrar un mensaje de acuerdo al framework. En la Figura 5-4, en cambio, se puede apreciar el mensaje originado por la regla que cuenta la cantidad de amigos que contiene la lista. Esta primera aproximación dejó de manifiesto que existían deficiencias en la forma en la cual se filtraban los elementos pues hacía muy engorroso el filtrar los amigos que comían golosinas. Dado esto se dio paso a la segunda etapa de diseño que originaría un nuevo prototipo (ver siguiente subsección).

Figura 5-1: Pantalla principal prototipo inicial

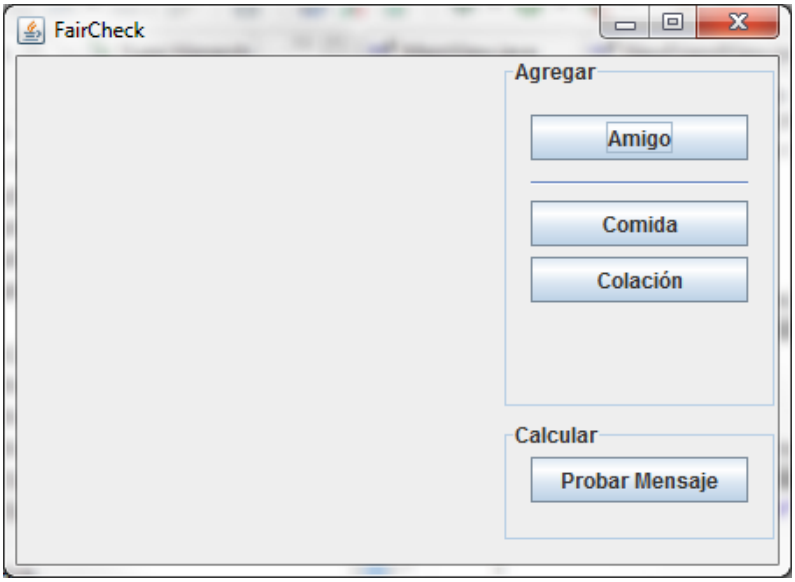


Figura 5-2: Mensaje de prueba de aplicación prototipo inicial

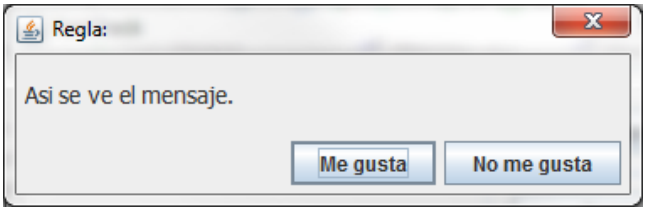


Figura 5-3: Agregar elementos en aplicación prototipo inicial

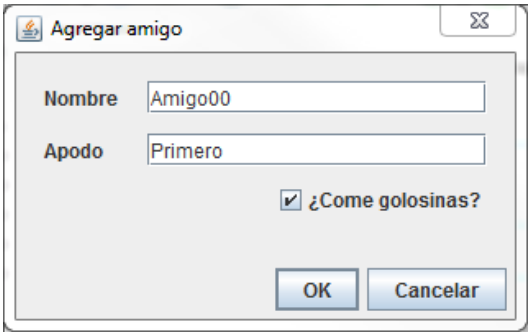
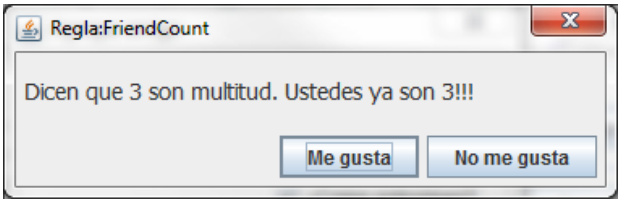


Figura 5-4: Mensaje de regla activa en aplicación prototipo inicial



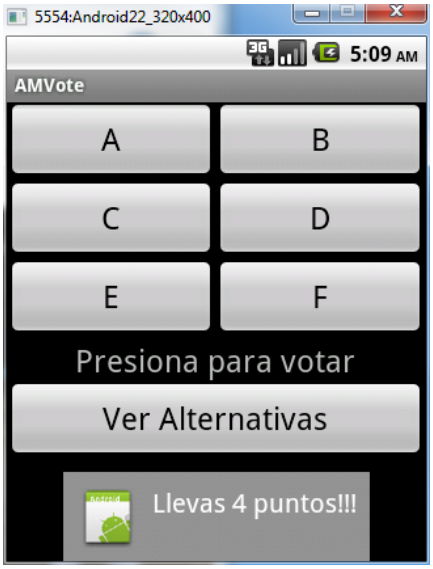
5.1.2 Segundo prototipo

Para el final de la segunda etapa de desarrollo se desarrolló un prototipo en un escenario más acorde a la realidad de la empresa que el del prototipo inicial. Para este desarrollo se ocupó un software ya existente llamado "Sistema de Votación". A grandes rasgos este sistema permite a usuarios tanto de equipos fijos como teléfonos móviles con sistema operativo Android participar de una votación electrónica dentro de una red local (para más detalles ver sección 7.2.1 - Sistema de Votación).

El segundo prototipo incluye todos los cambios señalados en la segunda etapa de diseño (ver sección 4.5) pero además incluye la implementación de elementos que no se probaron en el primer prototipo. En el primer prototipo la única fuente de datos que existía era local, no existiendo conexión a ningún tipo de fuente de datos externa. En el segundo prototipo se manejan fuentes de datos de ambos tipos (servicios web en el caso de las fuentes externas).

Se utiliza una única fuente de datos interna que almacena los votos enviados a través de la aplicación cliente y los marca como correctos o incorrectos según la respuesta del servidor. Se utiliza, además, una única fuente externa para obtener información histórica del rendimiento del estudiante. Con la información de estas dos fuentes de implementan tres reglas. La primera de las reglas cuenta la cantidad de puntos correctos que lleva el estudiante durante la presente sesión de trabajo. La segunda regla verifica si el estudiante está en una secuencia positiva y ha acertado al menos dos respuestas de forma consecutiva. La última de las reglas compara el rendimiento en la sesión (correctas/total) y lo compara con el rendimiento histórico, si el rendimiento de sesión es mayor se felicita al estudiante. En la Figura 5-5 se muestra el mensaje producido por la primera regla justo debajo de la información relevante de la aplicación, para ser consistente con los lineamientos de presentación de retroalimentación que se definieron anteriormente (ver sección 3.3.4).

Figura 5-5: Segundo prototipo. Mensaje de retroalimentación en Android.

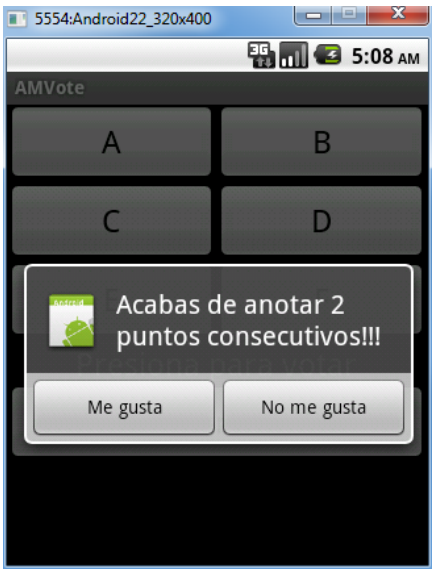


El segundo prototipo incluye mensajes de retroalimentación que no consultan al usuario si le agrada el mensaje. A diferencia de lo que se aprecia en la Figura 5-4 que siempre se presentan las opciones "Me gusta" y "No me gusta", en la Figura 5-5 el mensaje no pregunta al usuario

acerca de su opinión sobre la retroalimentación. Esta funcionalidad sigue presente, pero tal como se indica en la sección 3.3.4, no se debe consultar muy seguido al usuario de manera de no saturarlo ni quitarle el foco de las actividades pedagógicas. La Figura 5-6 presenta un mensaje de retroalimentación en el cual se recibe una respuesta del usuario.

Los resultados del segundo prototipo fueron satisfactorios pero se consideró que el diseño utilizado para su implementación aún era perfectible y que no sería el definitivo. Uno de los motivos por los que se tomó esta decisión fue la solicitud por parte de otros ingenieros de la empresa de contar con soporte para fuentes de datos SQL. Este tipo de fuentes de datos, aunque omitido en un principio (ver sección 4.3.3) se decidió implementar tomando en cuenta el interés de la empresa en ellos y que no afectaba mayormente los plazos para el desarrollo del presente trabajo.

Figura 5-6: Segundo prototipo. Mensaje de retroalimentación con consulta al usuario



5.1.3 Prototipo Final

El último prototipo se realizó en forma posterior a la tercera etapa de diseño e incluyó todos los cambios que se indican en ésta (ver sección 4.7). En términos generales el tercer prototipo es idéntico al segundo, salvo porque se ven reflejadas las mejoras respecto al diseño anterior. Se testeó el acceso a fuentes de datos SQL con un programa de prueba sencillo que no forma parte del prototipo desarrollado para Android. Una vez se terminó este prototipo, se consideró que el diseño estaba completo y se procedió a la implementación definitiva.

5.2 Separación del núcleo del framework

La primera parte de la implementación consiste en la separación del núcleo del framework. El objetivo de hacer esta separación es aislar los elementos que constituyen la estructura del framework de los elementos que corresponden a su implementación. La separación dejará las librerías *EmpathicCore* y *EmpathicFw*, siendo la primera la estructura del framework y la segunda el framework propiamente tal.

En el diseño realizado se distinguen clases e interfaces que conforman el núcleo del framework y que no pueden ser remplazadas por componentes distintas sin afectar la estructura de éste. Ejemplos de esto son: la clase `AbstractEmpathicKernel`, la interfaz `IRule`, etc. Por otra parte existen clases que hacen posible que el framework funcione, pero que podrían ser remplazadas por otras sin alterar la estructura de éste como por ejemplo la clase `LessUsedArbiterCriterion`.

De la implementación del prototipo final se tienen versiones funcionales de los elementos del framework que forman la estructura, por lo que es posible realizar una separación sin mayor esfuerzo. La primera de las librerías obtenidas en la separación es *EmpathicCore*, correspondiente al núcleo del framework. *EmpathicCore* está constituida casi en su totalidad por interfaces y clases abstractas, además de anotaciones y enumeraciones. Esta librería se obtuvo casi directamente en la separación, aunque algunas componentes debieron ser actualizadas para responder a los cambios realizados durante la etapa de diseño definitivo (ver sección 4.7). Aún cuando *EmpathicCore* es indispensable para el funcionamiento del framework no es funcional por sí misma pues cuenta únicamente con la implementación de los objetos y sus interacciones pero no con la implementación de sus funcionalidades completas.

Los elementos que no se consideran estructurales (y por ende no forman parte de *EmpathicCore*) son pasadas a una segunda librería llamada *EmpathicFw*. *EmpathicFw* corresponde a la implementación del framework como tal y está constituida por clases que en su mayoría extienden las clases pertenecientes a *EmpathicCore* y que implementan las funcionalidades que éstas no incluyen. A diferencia de lo sucedido con *EmpathicCore*, para *EmpathicFw* los elementos se encuentran parcialmente implementados y poco optimizados, por lo que se requiere trabajo adicional. Lo primero que se hizo fue replicar la estructura de paquetes de forma que reflejara la que posee *EmpathicCore*. Luego se tomaron los elementos existentes y se movieron a esta librería (*EmpathicFw*). Para la implementación y mejora de los elementos existentes se pasó a la etapa que se describe en la siguiente sección..

5.3 Programación de clases con métodos pre-implementados

Una vez que se separaron las librerías *EmpathicCore* y *EmpathicFw*, se pasó a completar la implementación de ésta última. Mientras *EmpathicCore* es la estructura del framework, *EmpathicFw* es el framework propiamente tal. *EmpathicFw* se compone de las clases abstractas que implementan parcialmente las interfaces y las clases concretas que articulan el funcionamiento del framework.

5.3.1 Clases abstractas

Existen clases que requieren un esfuerzo de implementación muy alto si es que se hace directamente desde las interfaces. Este es el caso de los administradores (de reglas, de fuentes de datos y de la interfaz de usuario) y de los tipos de fuentes de datos (interna, web, SQL). En Java, es común ver con interfaces del paquete `java.util` que se incluyen clases abstractas que implementan dichas interfaces con el objetivo de reducir el esfuerzo de desarrollo requerido. En esta línea es que se provee implementación para las siguientes interfaces:

1. `IWebDataSource<T>` con la clase `AbstractWebDataSource<T>`.
2. `ISqlDataSource<T>` con la clase `AbstractSqlDataSource<T>`.

3. `IDataManager` con la clase `AbstractDataManager`.
4. `IRuleManager` con la clase `AbstractRuleManager`.
5. `IUiManager` con la clase `AbstractUiManager`.

La implementación de estas interfaces en las correspondientes clases abstractas ayuda a reducir la cantidad de código que un desarrollador debe programar. Sin embargo, estas clases abstractas proveen implementaciones parciales de la interfaz. Dado esto es posible disminuir más todavía el trabajo de los usuarios (desarrolladores) por medio de la implementación de clases concretas que contengan todas las funcionalidades que requiere la interfaz raíz.

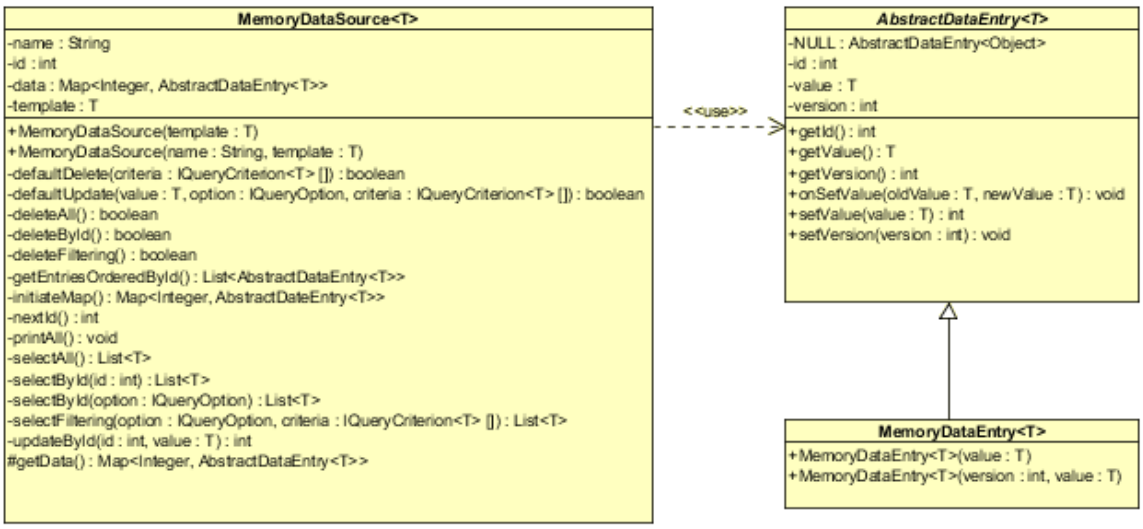
5.3.2 Clases concretas y ejecución del framework

Un framework debe incluir una línea de ejecución predeterminada, vale decir, debe poder funcionar con nula o mínima implementación de elementos extras por parte del desarrollador. Dado esto y la naturaleza de algunos elementos del framework, existen clases y funciones que se pueden dejar totalmente implementadas.

Para que el framework sea funcional se necesita un `AbstractEmpathicKernel`, un `AbstractEmpathicPlugin`, un `IArbiter`, un `IArbiterCriterion`, un `IDataManager`, un `IRuleManager`, un `IUiManager` y algún tipo de fuente de datos. Se implementaron las siguientes ocho clases:

1. DefaultEmpathicKernel: Una implementación funcional de la clase abstracta `AbstractEmpathicKernel`. Utiliza de forma predeterminada un `DefaultEmpathicPlugin` para obtener los elementos predeterminados.
2. DefaultEmpathicPlugin: Incluye referencias para el `IArbiter`, `IArbiterCriterion`, `IDataManager`, `IRuleManager` e `IUiManager` que serán utilizados por el framework.
3. DefaultArbiter: Una implementación funcional de la interfaz `IArbiter`. Esta clase debiese ser la menos susceptible a requerir modificación del usuario puesto que la mayor parte del trabajo de selección de una recta se da en las clases que implementan la interfaz `IArbiterCriterion`.
4. LessUserCriterion: Una implementación de `IArbiterCriterion`. Selecciona la regla menos utilizada entre las reglas seleccionables. En caso de haber más de una, escoge de forma aleatoria.
5. DefaultDataManager: Extiende `AbstractDataManager`.
6. DefaultRuleManager: Extiende `AbstractRuleManager`.
7. DefaultUiManager: Extiende `AbstractUiManager`. Presenta los mensajes en la consola.
8. MemoryDataSource<T>: Implementa `IDataSource<T>`. Representa una fuente de datos en la memoria interna de la aplicación.

Figura 5-7: El tipo predeterminado de fuentes de datos: MemoryDataSource<T>



5.4 Visión general de funcionalidades implementadas

Una vez concluida la implementación se cuenta con un framework que cumple con los requisitos expuestos en la sección 4.2.3, es decir: permite evaluar indicadores con la información disponible de los usuarios; provee un mecanismo para verificar y priorizar condiciones sobre los indicadores; y permite conectarse a fuentes de datos externas y es capaz de funcionar sin conexión. A continuación se presenta una visión general de las distintas funcionalidades presentes en el framework al momento de terminar su implementación.

5.4.1 Manejo de fuentes de datos

De acuerdo a lo expuesto en la sección 4.2.3, el framework debe permitir conectarse a fuentes de datos externas y debe ser capaz de funcionar sin conexión. Respecto a estas necesidades, la implementación final cuenta con soporte para fuentes de datos externas de tipo web y SQL, además de contar con fuentes de datos internas que hacen que el framework no requiera de una conexión a Internet para funcionar.

Las fuentes de datos de todo tipo interpretan las respuestas de las operaciones de inserción, actualización, eliminación y búsqueda de forma uniforme. En el caso de las inserciones, la respuesta a la solicitud se interpreta como un número entero. Para las actualizaciones, también se interpreta la respuesta como un número entero. En las eliminaciones se entrega un valor binario (`boolean`) para saber si es que la solicitud eliminó alguna entrada. Para las búsquedas el resultado se interpreta como una lista de objetos del tipo que se almacena en la fuente de datos, es decir, una fuente de datos de `String` retornará una lista de `String`s.

Las fuentes de datos externas web permiten hacer peticiones HTTP, pudiendo tener distintas URLs para hacer inserciones, actualizaciones, eliminaciones y búsquedas. Las fuentes de datos SQL, realizan consultas a bases de datos relacionales tradicionales. Finalmente, las fuentes de datos internas almacenan en memoria elementos de un tipo determinado.

5.4.2 Definición de reglas para presentar mensajes de retroalimentación

La funcionalidad de manejo de fuentes de datos, dota al framework de la capacidad de extraer información útil para obtener estadísticas de los usuarios de software educativo. Esto permite cumplir con la funcionalidad requerida en la sección 4.2.3, donde se indica que el framework debe permitir evaluar indicadores con la información disponible de los usuarios. Se requiere, además, que el framework provea un mecanismo para verificar y priorizar condiciones sobre los indicadores recién mencionados. Este objetivo se cumple parcialmente mediante el uso de reglas para presentar mensajes de retroalimentación.

Una regla constituye una serie de condiciones que deben cumplirse en la información que se dispone del usuario (y por ende en los indicadores que se calculen con ella) para que se presenten los mensajes de retroalimentación. El framework provee una interfaz para la definición de estas reglas (`IRule`) y una clase auxiliar que implementa la mayoría de los métodos de dicha interfaz (`AbstractRule`), a fin de reducir el esfuerzo requerido en el desarrollo de una regla.

5.4.3 Priorización de elementos y presentación de mensajes

En la sección 4.2.3, se requiere que el framework provea un mecanismo para verificar y priorizar condiciones sobre los indicadores recién mencionados. Parte de este objetivo (la verificación) se cumple mediante la posibilidad de especificar reglas para presentar mensajes de retroalimentación. La otra parte de este objetivo corresponde a la priorización de las condiciones que se cumplen.

En un escenario cotidiano, es muy posible que se cumplan dos o más condiciones de forma simultánea, como por ejemplo, que un estudiante tenga 100% de aciertos en una sesión y que haya acertado a tres preguntas de forma consecutiva. El framework provee una interfaz (`IArbiter`) para implementar la clase encargada de administrar las reglas y los criterios para priorizarlas. Se provee además una implementación completa de esta interfaz en la clase `DefaultArbiter`. Respecto a los criterios utilizados para priorizar, el framework tiene la interfaz `IArbiterCriterion` para definir que mecanismos se usaran para determinar cuál es la regla más indicada para presentar retroalimentación. Se dispone de una implementación de esta interfaz en la clase `LessUsedCriterion`.

5.4.4 Registro de mensajes de retroalimentación

En la sección 3.3.4 se reconoce la necesidad de consultar al usuario sobre la retroalimentación que recibe y como la percibe. Esta información es de gran importancia pues permite que se conozca la efectividad de la retroalimentación que se está entregando. Si bien no se especificó como un requisito para el presente trabajo, se consideró demasiado relevante como para dejarlo fuera.

Se provee de un modelo de datos para el almacenamiento de información sobre la retroalimentación entregada. Este modelo de datos se separa en dos grupos principales de entidades: uno global que se puede utilizar en cualquier tipo de aplicación y otro especifico para las necesidades de AutoMind.

6 Cuarta etapa: Evaluación

La cuarta etapa del trabajo consiste en la evaluación del framework, esta evaluación se divide en dos ramas principales: implementación y usabilidad.

En la evaluación de implementación se calculan métricas sobre el código implementado. En la de usabilidad, se hace una prueba de las funcionalidades del framework con usuarios finales, que, en este caso son desarrolladores.

6.1 Implementación

Para calcular las métricas se hizo uso de JDepend, específicamente del plugin para Eclipse. JDepend calcula específicamente las métricas de acoplamiento aferente, acoplamiento eferente, inestabilidad, abstracción, distancia. Muestra además el número de clases concretas y abstractas (incluyendo interfaces).

Se hizo una toma de resultados para *EmpathicCore*, *EmpathicFw* y para ambas librerías juntas de forma de capturar de mejor forma todas las dependencias y acoplamientos.

Dentro de los valores obtenidos, el más importante es la distancia a la recta ideal. Esta recta está dada por la formula A + I = 1 (abstracción + inestabilidad = 1). La distancia mide la diferencia entre la suma con el valor ideal: D = |A + I – 1|. Se considera un valor de 0,1 como satisfactorio.

6.1.1 Métricas de librería EmpathicCore

Tabla 6-1: Métricas usando JDepend para EmpathicCore

Paquete	CC	AC	CA	CE	A	I	D
cl.automind.empathy	2	5	0	4	0.71	1.00	0.71
cl.automind.empathy.data	16	12	4	4	0.42	0.50	0.08
cl.automind.empathy.data.sql	12	10	0	2	0.45	1.00	0.45
cl.automind.empathy.data.web	0	1	0	2	1.00	1.00	1.00
cl.automind.empathy.feedback	2	4	3	2	0.66	0.40	0.06
cl.automind.empathy.rule	7	5	1	4	0.41	0.80	0.21
cl.automind.empathy.ui	0	1	1	1	1.00	0.50	0.50
Promedio							0.43

Observaciones: Una distancia de 0.43 es un pésimo valor para el diseño. Este valor, sin embargo, no es extraño considerando la composición de esta librería: *EmpathicCore* está compuesta por las clases abstractas e interfaces que construyen el framework. Por otra parte al estar toda la estructura del framework, las clases están fuertemente asociadas, lo que implica una mayor inestabilidad.

6.1.2 Métricas de librería EmpathicFw

Tabla 6-2: Métricas usando JDepend para EmpathicFw

Paquete	CC	AC	CE	CA	A	I	D
cl.automind.empathy.fw	2	0	0	8	0.00	1	0.00
cl.automind.empathy.fw.arbiter	3	0	1	2	0.00	0.66	0.33

cl.automind.empathy.fw.data	5	1	1	5	0.16	0.83	0.01
cl.automind.empathy.fw.data.sql	4	2	0	6	0.33	1.00	0.33
cl.automind.empathy.fw.data.web	1	2	0	5	0.66	1.00	0.66
cl.automind.empathy.rule	3	0	1	2	0.00	0.66	0.34
cl.automind.empathy.ui	3	1	1	3	0.25	0.75	0.00
Promedio							0.24

Observaciones: Esta librería presenta una distancia de 0.24, muy menor a la de 0.43 de *EmpathicCore*. Esta librería incluye más clases concretas, por lo que la abstracción es más baja que la de la primera librería. La inestabilidad es similar en ambos casos.

6.1.3 Métricas de librería EmpathicCore y EmpathicFw

Tabla 6-3: Métricas usando JDepend para ambas librerías

Paquete	CC	AC	CE	CA	A	I	D
cl.automind.empathy	2	5	2	4	0.71	0.66	0.38
cl.automind.empathy.data	16	12	8	4	0.42	0.33	0.23
cl.automind.empathy.data.sql	12	10	1	2	0.45	0.66	0.12
cl.automind.empathy.data.web	0	1	1	2	1.00	0.66	0.66
cl.automind.empathy.feedback	2	4	4	2	0.66	0.33	0.00
cl.automind.empathy.rule	7	5	4	4	0.41	0.50	0.09
cl.automind.empathy.ui	0	1	3	1	1.00	0.25	0.25
cl.automind.empathy.fw	2	0	0	8	0.00	1	0.00
cl.automind.empathy.fw.arbiter	3	0	1	2	0.00	0.66	0.34
cl.automind.empathy.fw.data	5	1	1	5	0.16	0.83	0.01
cl.automind.empathy.fw.data.sql	4	2	0	6	0.33	1.00	0.33
cl.automind.empathy.fw.data.web	1	2	0	5	0.66	1.00	0.66
cl.automind.empathy.rule	3	0	1	2	0.00	0.66	0.33
cl.automind.empathy.ui	3	1	1	3	0.25	0.75	0.00
Promedio							0.24

Observaciones: al unir ambas librerías la distancia obtenida es similar a la que se obtiene sólo por el uso de *EmpathyFw* y es menor a la de *EmpathicCore*. Un valor de 0.25 no es catalogable como muy bueno, pero tampoco es deficiente. Una forma de atacar el problema es agregando una mayor cantidad de clases concretas de modo de disminuir la abstracción y por consiguiente la distancia. Sin embargo, los números menos favorables se encuentran en la columna Inestabilidad. Un diseño futuro debiese considerar la opción de hacer clases menos acopladas.

6.2 Usabilidad

Una vez realizada la evaluación de las métricas se efectuó la evaluación con usuarios finales. A diferencia de lo que sucede en una prueba de usabilidad tradicional, en este trabajo los usuarios finales son desarrolladores de software. En este caso no se busca evaluar una vista o una presentación sino que la extensibilidad del software y la facilidad de uso de la API (siglas en inglés para interfaz de programación de aplicación) que se entrega.

Para llevar a cabo la evaluación se diseñó una encuesta de usabilidad. Esta encuesta está ampliamente basada en las desarrolladas por el profesor del Departamento de Ciencias de la Computación de la Universidad de Chile, Jaime Sánchez.

La encuesta tiene relación con la experiencia en un desarrollo en particular utilizando el framework y se divide en dos partes:

1. Una sección de preguntas que se califican de acuerdo a una escala numérica.
2. Una sección de preguntas abiertas.

Las preguntas apuntan a medir las siguientes dimensiones en relación a la experiencia del usuario:

- Facilidad de uso: Qué tan fácil es para el usuario utilizar el framework.
- Facilidad de extensión: Qué tan fácil es para el usuario incorporar nuevos elementos que extiendan los ya existentes en el framework.
- Satisfacción en general: Qué tan conforme está el usuario con el framework.

Se realizó esta encuesta a tres personas que se dedican al desarrollo de software. Cada uno de ellos cuenta con experiencia en el desarrollo con Java y el manejo de clases con tipos genéricos y anotaciones. Los encuestados son:

1. Josefina Hernández: Estudiante de Ingeniería Civil en Computación de la Pontificia Universidad Católica.
2. Manuel Bahamondez: Ingeniero Civil en Computación de la Universidad de Chile.
3. Carlos Aguirre. Ingeniero Civil Electricista de la Universidad de Chile.

6.2.1 Resultados primera parte de la encuesta

A continuación se presentan los resultados de la primera parte de la encuesta, correspondiente a la evaluación cualitativa. Se separan las distintas sub-secciones de esta parte y se agregan comentarios respecto a los números obtenidos.

6.2.1.1 Datos

Tabla 6-4: Resultados test usabilidad - Uso general del framework

PREGUNTA	RESPUESTA (1 al 7)			
USO GENERAL DEL FRAMEWORK	JH	MB	CA	Media
Me gustó el framework	7	7	6	6.66
Usaría el framework en mi aplicación	6	7	6	6.33
El framework es fácil de usar	7	6	5	6.00
El framework es fácil de extender	6	7	6	6.33

Observaciones: Como se puede apreciar en la primera tabla, hubo una buena recepción para el framework. El valor más bajo está ubicado en el ítem “El framework es fácil de usar”. Este resultado sugiere que es necesario mejorar algún elemento que preste ayuda al usuario.

Tabla 6-5: Resultados test usabilidad - Sistema de plugins del framework

PREGUNTA	RESPUESTA (1 al 7)			
SISTEMA DE PLUGINS DEL FRAMEWORK	JH	MB	CA	Media
El sistema de plugins del framework es útil	7	7	7	7.00
Los plugins son fáciles de usar	7	6	6	6.33

Observaciones: En general, hay una buena evaluación para el sistema de plugins del framework. Este sistema permite cargar reglas y fuentes de datos en lotes distintos, lo que da la posibilidad de agruparlos y activarlos/desactivarlos modificando poco código.

Tabla 6-6: Resultados test usabilidad - Programación con el framework

PREGUNTA	RESPUESTA (1 al 7)			
PROGRAMACIÓN/IMPLEMENTACIÓN CON EL FRAMEWORK	JH	MB	CA	Media
Programar/Implementar un criterio es sencillo	6	6	7	6.33
Programar/Implementar una regla es sencillo	5	7	7	6.33
Programar/Implementar un mensaje es sencillo	7	7	6	6.66

Observaciones: Durante la prueba se sugirió utilizar las clases abstractas que provee el framework. Estas clases están enfocadas en disminuir los métodos que es necesario implementar para desarrollar cualquiera de los distintos elementos.

Tabla 6-7: Resultados test usabilidad - Documentación del framework

PREGUNTA	RESPUESTA (1 al 7)			
DOCUMENTACIÓN DEL FRAMEWORK	JH	MB	CA	Media
La documentación de las clases es clara	7	5	6	6.00
La documentación es completa	5	5	5	5.00
La documentación es útil	7	5	6	6.00

Observaciones: El punto más bajo en la evaluación general es la documentación. Al momento de realizar el test, la documentación no se encontraba completa, por lo que algunas clases carecían de información detallada para su uso.

Tabla 6-8: Resultados test usabilidad - Nombres de los elementos del framework

PREGUNTA	RESPUESTA (1 al 7)			
NOMBRES DE LOS ELEMENTOS DEL FRAMEWORK	JH	MB	CA	Promedio
Los nombres de las clases son representativos (auto-	6	7	7	6.66

explicativos)				
Los nombres de los métodos son representativos (auto-explicativos)	6	7	7	6.66
Los nombres de las propiedades son representativos (auto-explicativos)	6	7	6	6.33

Observaciones: Een el punto de documentación la nota fue bajo 6. La buena evaluación a los nombres de clases y métodos puede explicar el por qué, en general, el framework no fue considerado difícil de usar.

6.2.1.2 Observaciones

En términos generales el framework recibió una buena evaluación cualitativa. Se destacan positivamente los aspectos de nombramiento (clases, métodos, propiedades) y negativamente la documentación.

Se desprende de los resultados que es necesario hacer más clara la documentación. Además se aprecia que el desarrollador valora una documentación completa.

6.2.2 Resultados segunda etapa

A continuación se presentan los resultados de la segunda parte de la encuesta, correspondiente a la evaluación cuantitativa. Se separaran las distintas sub-secciones de esta esta parte y se agregan comentarios respecto a los comentarios recibidos.

6.2.2.1 Datos

Tabla 6-9: Resultados test usabilidad - Sintaxis

SINTAXIS	
¿Cómo considera la sintaxis (engorrosa, sencilla, etc.)? ¿Por qué?	
JH	Sencilla, los nombres se entienden sin problemas. Sin embargo, quizás, hay muchos nombres parecidos que hacen lenta la búsqueda de objetos.
MB	A primera vista parecía engorrosa, pero luego de hacer el test de usabilidad se logra comprender el funcionamiento y la sintaxis del sistema.
CA	Sencilla, ya que el esquema está bien modularizado.
La ayuda otorgada por la documentación y los nombres de los distintos elementos (clases, métodos, etc.). ¿Es útil? ¿Facilita la elaboración del código?	
JH	Sí, es útil y facilita la elaboración, pero falta un poco.
MB	Sí, los nombres de los métodos y clases son bastante descriptivos.
CA	Está bien ya que cumple con el estándar típico de las aplicaciones Java.
¿Considera que algún elemento es redundante? ¿Considera que se necesita algún elemento adicional?	
JH	Para el ejercicio propuesto están todos los elementos necesarios, sin encontrarse problemas al respecto.

MB	No.
CA	No.

Observaciones: No se consideró que se necesitara algún elemento adicional en el diseño, esto puede deberse como menciona Josefina al caso de uso específico. Sin embargo, tanto la sintaxis como los nombres de los elementos son compartidos por los elementos que no se cubrieron. Esto hace pensar que no debiese haber mayor diferencia en otros casos de uso.

Tabla 6-10: Resultados test usabilidad - Diseño

DISEÑO	
¿Considera que el framework ayuda a realizar una separación entre la lógica de las reglas y la lógica propia de la aplicación?	
JH	Sí, no es problema.
MB	Sí, el sistema entrega una buena separación permitiendo que las reglas puedan ser reutilizadas por diferentes aplicaciones.
CA	Sí.
¿Qué le parece el diseño del framework? ¿Es muy complejo? ¿Tiene muchas clases, métodos y propiedades? ¿Facilita su extensión?	
JH	Está bien separado en carpetas (paquetes) por lo que no es complejo, por lo menos para el ejercicio propuesto.
MB	Las clases internas son bastante complejas, pero las clases abstractas que se entregan como base simplifican bastante el desarrollo.
CA	Excelente, sencillo pero poderoso.

Observaciones: Los usuarios reconocieron que el framework facilita la tarea de separar las lógicas. Consideraron además que el diseño, estaba ordenado y que las clases abstractas simplificaban el desarrollo.

Tabla 6-11: Resultados test usabilidad - Selección de elementos

SELECCIÓN DE ELEMENTOS	
¿Qué destaca del framework?	
JH	La facilidad con que se crean las reglas y los mensajes.
MB	Su extensibilidad.
CA	Simplicidad y potencia.
¿Qué mejoraría del framework?	
JH	Podría mejorar el tener que editar y/o cambiar la menor cantidad de clases posibles.
MB	La documentación.
CA	Documentación.

¿Qué agregaría al framework?	
JH	Clases protegidas que no se puedan evitar para evitar problemas.
MB	Nada.
CA	Ejemplos y tutoriales.

Observaciones: Se destacan distintos elementos del framework, existe consenso en que el punto más débil es la documentación.

Tabla 6-12: Resultados test usabilidad - Preguntas generales

PREGUNTAS GENERALES	
¿Qué fue lo que más le gustó del framework?	
JH	Fácil de entender como se usa.
MB	La extensibilidad, lo que permite implementar reglas y fuentes de datos no contempladas al momento de la creación de este framework.
CA	Simplicidad.
¿Qué fue lo que menos le gustó del framework?	
JH	Poco protegido contra escritura por error.
MB	Sin comentarios.
CA	Sin comentarios.
¿Qué otro uso le daría al framework?	
JH	Existen demasiadas aplicaciones que pueden usarlo.
MB	Sin comentarios.
CA	Es genérico, por lo que su uso es variado y aplicable a varios campos.
Algún otro comentario que agregar	
JH	Bien.
MB	Excelente trabajo, muy completo y a la vez extensible.
CA	Sin comentarios.

6.2.2.2 Observaciones

El framework resultó bien evaluado en términos generales. Se recopiló información importante de elementos que se deben potenciar y elementos que se deben mejorar. Dentro de los elementos a potenciar están: la extensibilidad, el buen uso de las convenciones de nombres y la

posibilidad de ser utilizado para otro tipo de aplicaciones. Dentro de los elementos a mejorar están: la protección de algunos elementos (prohibir herencia en algunas partes), requerir la edición de menos clases y, por sobretodo, la documentación.

7 Quinta etapa: Mejoras a un software existente de la empresa utilizando el framework

7.1 Descripción

En esta etapa se hace uso del framework para realizar mejoras a un programa ya existente de la empresa. Se escogió trabajar en el Sistema de Votación, sistema en que el que se contaba con experiencia previa tanto en el diseño como en el desarrollo. Esta etapa del trabajo se dividió en dos sub-etapas. La primera de ellas consistió en integrar los datos del programa con los datos de la empresa. La segunda en implementar las mejoras al sistema.

7.2 Integración del Sistema de Votación

El Sistema de Votación era el único de los servicios que se encontraban operando fuera de los datos que utilizan los distintos servicios de la empresa. A continuación se presenta con más detalle una descripción del programa y cuál era el estado en que se encontraba al momento de comenzar el trabajo.

7.2.1 Sistema de Votación

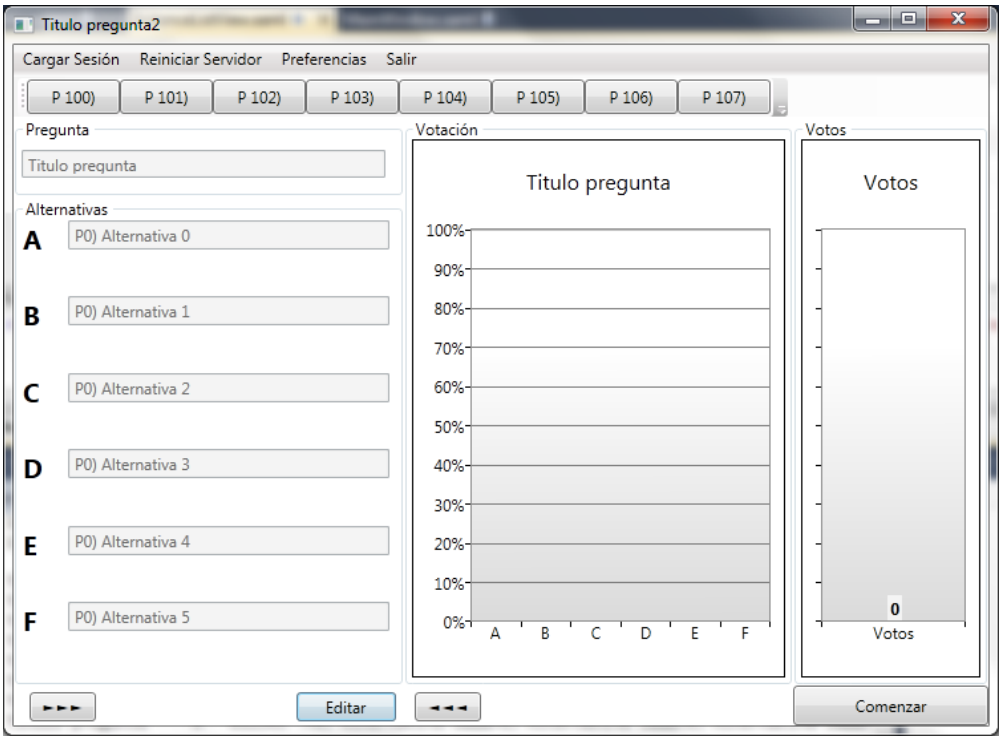
El Sistema de Votación consiste en una herramienta que permite proyectar preguntas y recabar las respuestas de los participantes mediante una "botonera". La primera versión de esta aplicación fue desarrollada para ser usada en el evento Chile VA! 2011. Consistía de dos partes principales, el servidor de votación y el votador móvil que funcionaba en celulares con Android 1.6 que contaran con Wi-Fi. Además presenta una herramienta de testeo que no es de mayor interés para este desarrollo.

La segunda versión del Sistema de Votación cuenta con las mismas dos partes que la primera versión y agrega además un editor independiente de preguntas. El desarrollo de esta segunda versión está motivado en la necesidad de tener una interfaz de mayor calidad y dar mejores opciones de presentación y usabilidad que la versión inicial.

7.2.1.1 Servidor Votación

La versión inicial del servidor de votación consistía en un editor de preguntas que permitía recoger las preferencias de los usuarios conectados. El editor era limitado y con una interfaz de usuario que a pesar de ser usable no era la más amigable.

Figura 7-1: Versión inicial del Sistema de Votación



Durante el primer semestre del 2011 se comenzó con el desarrollo de una segunda versión del programa. Esta nueva versión permitía realizar votaciones de la misma forma que la versión anterior, pero las preguntas se presentan en forma de diapositivas y el trabajo de servidor se realiza de manera más transparente para el usuario. La interfaz de usuario fue desarrollada usando *Windows Presentation Foundation* (WPF), dando énfasis al control *Ribbon* que fue introducido en la versión 2007 de Microsoft Office.

Entre las principales diferencias con la versión inicial, está el hecho de que se separó la labor de edición y creación de contenido a un programa externo (Editor de Preguntas, que se detalla en la siguiente parte). Este cambio dejó al Servidor de Votación como un visor de preguntas y servidor.

7.2.1.2 Votador Móvil

La versión original del votador móvil consistía en una botonera que permitía a los usuarios autentificados con su RUT votar. La segunda versión, desarrollada en paralelo con la del servidor de votación, incluye un código de autentificación y la opción de ver las preguntas de la votación activa.

7.2.1.3 Editor de Preguntas

El editor de preguntas fue diseñado para eliminar la labor de creación y edición de contenidos del programa servidor. El objetivo de esto era poder hacer uso de este programa sin necesidad de estar conectado a los servidores y de poder distribuirlo a personas sin necesidad de darles acceso a las bases de datos.

El diseño de la interfaz, al igual que para el Servidor de Votación, está basado en los controles de Microsoft Office 2007 y 2010 haciendo uso del control gráfico "Ribbon". Por medio de esta implementación se buscaba que la interfaz fuera más sencilla de usar.

Figura 7-2: Segunda versión del Sistema de Votación

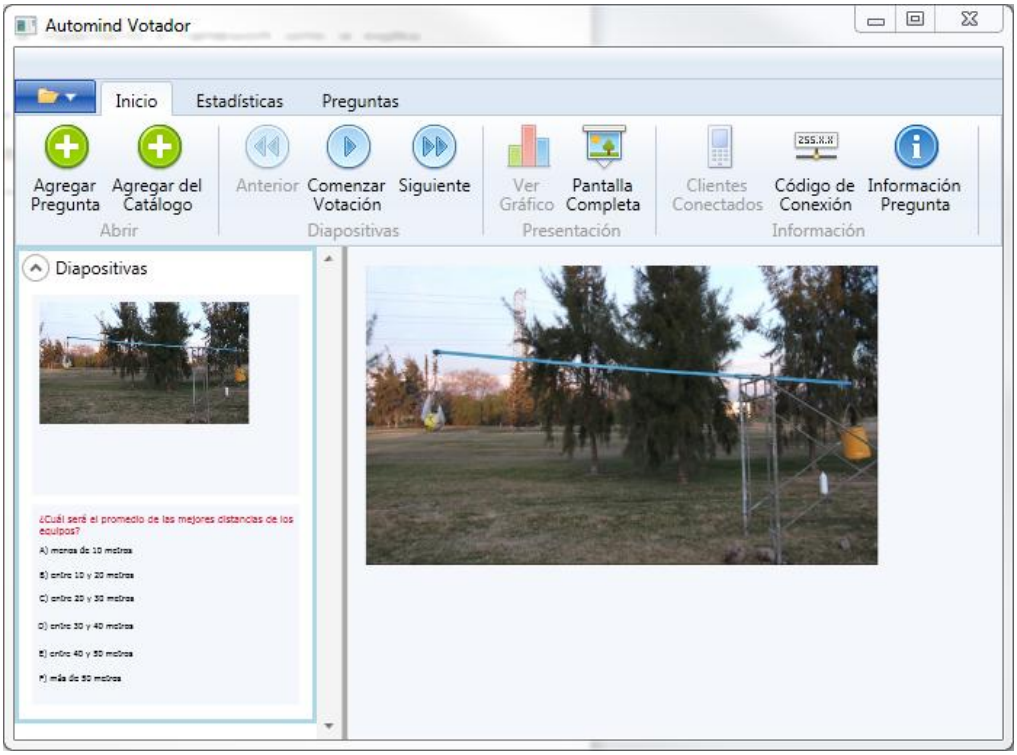
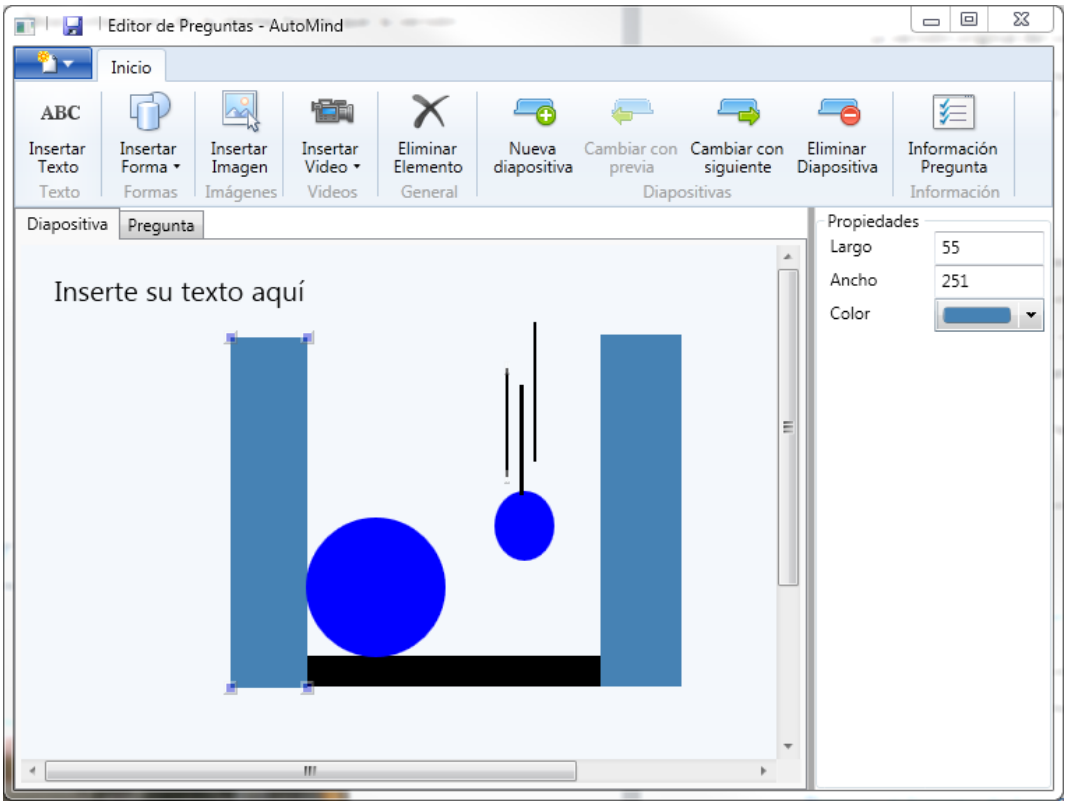


Figura 7-3: Segunda versión del Editor de preguntas



7.2.2 Estado del Sistema de Votación al comenzar el proyecto

Al comenzar el presente trabajo el Sistema de Votación se encontraba incompleto y la información que utilizaba y almacenaba se encontraba ajena a las que utilizan los demás sistemas de la empresa.

7.2.3 Mejoras al Sistema de Votación ajenas al framework

7.2.3.1 Integración de Datos

La primera y más importante de las mejoras a llevar a cabo en el Sistema de Votación era la integración con los datos centrales. Cada votación debe tener asignado un profesor responsable, además de un curso al que se realiza y por lo tanto un nivel educativo y un establecimiento específico. Previo a este trabajo no se contaba con ninguno de estos elementos.

Para llevar a cabo las mejoras se desarrolló una librería con el objetivo de aislar la comunicación y consultas específicas a la base de datos. Con este aislamiento se consigue dar flexibilidad para que si en el futuro se cambia a una nueva base de datos o el modelo de datos difiere, el cambio se haga con relativa facilidad.

Los distintos servidores de bases de datos de la empresa utilizan Postgresql, por lo que para el desarrollo concreto de las clases que realizan las consultas se utilizó Npgsql. Se distinguen las siguientes funcionalidades agregadas desde que se integran los datos:

1. Se asigna institución en la que se realiza la votación.

2. Se asigna nivel específico que participa de la votación.
3. Se asigna la votación a un curso determinado.
4. Se asigna a un profesor responsable de llevarla a cabo.
5. Se filtra las personas que pueden participar en la votación por curso específico dentro de la institución.
6. Se registra la creación de una determinada sesión de votación.
7. Se registran las preguntas que serán votadas en una sesión.
8. Se registran los votos de los participantes indicando pregunta específica, voto, emisor y tiempo demorado en responder.
9. Se agrega la opción de modificar la alternativa correcta de una pregunta durante las mismas votaciones.
10. Se agregan tablas indicando el rendimiento de los votantes durante una determinada votación.

7.3 Mejoras al sistema de votación utilizando el framework

Luego de realizar mejoras generales al Sistema de Votación, se continuó haciendo mejoras que involucren el uso del framework desarrollado.

Tal como se indicó en la primera parte de este trabajo (ver sección 3) existen necesidades de motivación de los estudiantes. Estas necesidades se han decidido abordar haciendo uso de una mejor retroalimentación, basada en la información existente del alumno. Se identificó información de dos tipos: histórica y de sesión.

7.3.1 Contexto

El Sistema de Votación permite a los estudiantes contestar determinadas preguntas desde un equipo móvil (con Android) o desde un computador de escritorio. Cada una de estas preguntas tiene asociada información específica de su contenido educativo. Esta información incluye: sector educativo, eje educativo, contenido mínimo obligatorio y nivel correspondiente.

Cuando un estudiante responde a una pregunta, se almacena su voto, incluyendo su información personal, y otra miscelánea como el tiempo que tardó en contestar. Dado esto se cuenta con el historial de los votos de los estudiantes, además de los elementos propios de cada sesión de votación. Con estos datos es posible recopilar la información de interés para dar una buena retroalimentación.

Antes de mostrar las mejoras es necesario conocer algo más de la aplicación cliente. Esta aplicación lleva un registro de las preguntas respondidas y los puntajes obtenidos en un objeto de tipo `VoteResult` como se ve en el Código 7-1. Una de las variables de instancia de dicha clase tiene tipo SagdeMetadata (ver Código 7-2). Este tipo almacena información relativa al objeto educativo que emanó este mensaje de retroalimentación.

Código 7-1: La clase VoteResult

```
public class VoteResult {
	private final int turn;
	private final int value;
	private int idInstance;
	private int idQuestion;
	private SagdeMetadata sagdeMetadata;
```

```
        public VoteResult(int turn, int value){
                this.turn = turn;
                this.value = value;
                this.sagdeMetadata = new SagdeMetadata();
        }
        /*  Getters y Setters con nombre por estándar. */
}
```

Código 7-2: La clase SagdeMetadata

```
public class SagdeMetadata{
        private int idSector;
        private int idAxis;
        private int idCmo;
        private int idKo;
        private int idLevel;

        public SagdeMetadata(){
                this.idSector = 1;
                this.idAxis = 1;
                this.idCmo = 1;
                this.idKo = 1;
                this.idLevel = 1;
}
/*  Getters y Setters con nombre por estándar. */
}
```

7.3.2 Mejoras utilizando información histórica

Los datos almacenados son suficientemente completos como para poder separar distintos elementos. Por ejemplo, es posible identificar el rendimiento del estudiante por sector educativo, eje educativo, mes específico de trabajo, etc.

Para los cambios utilizando información histórica, se optó por premiar las mejoras que ha hecho el estudiante en su desempeño. Durante el resto de esta sección (7.3.2) se explica detalladamente en que consiste cada elemento y los pasos a seguir involucrados en la implementación de una regla real en un contexto de producción. Se incluye adicionalmente el código utilizado para ejemplificar de mejor manera elementos como la extensión de las clases y métodos importantes. La regla a implementar mide si es que el desempeño durante una sesión de trabajo del estudiante es al menos un 10% superior a su desempeño histórico, para un CMO específico. Si es así, entonces se le felicita. Se pide como requisito que existan al menos tres preguntas con ese CMO. Hay que notar que esta regla incluye información de ambos tipos, histórica y de sesión.

Lo primero que es necesario saber es si esta regla es factible de evaluar, vale decir, si existen a lo menos tres ejercicios que tengan un CMO en común. Para obtener esta información es necesario filtrar los resultados de votaciones que pertenezcan a un CMO determinado. Para esto se extiende la clase `QueryCriterion<T>` como se muestra en Código 7-3.

Código 7-3: Extensión de un criterio para filtrar datos

```
public class ScoreByIdCmoCriterion extends QueryCriterion<VoteResult> {
    public ScoreByIdCmoCriterion(){
        super();
      }
    public ScoreByIdCmoCriterion
        (VoteResult target, NamedValuePair<?>[] params) {
```

```
            super(target, params);
        }
        @Override
        public boolean apply
            (VoteResult target, NamedValuePair<?>... params) {
            for (NamedValuePair<?> pair : params){
                if (pair.getKey().equals("idcmo")){
                    try{
                        int idcmo = (Integer) pair.getValue();
                        return target.getSagdeMetadata().getIdCmo()== idcmo;
                    } catch (Exception e){
                        return false;
                    }
                }
            }
            return true;
        }
    }
```

Una vez que se ha escrito la clase responsable de filtrar los datos, se crea una regla que evalúe si el rendimiento del usuario en esta sesión de trabajo supera al menos en un 10% a su rendimiento histórico. A esta regla se le da el nombre de `CmoPerformanceIncreaseRule` y extiende a la clase `AbstractRule`. Es necesario sobrescribir al menos dos métodos de esta clase (`AbstractRule`) para poder hacer operativa esta regla. En primer lugar se sobrescribe el método `canEvaluate` que identifica si se cumplen las condiciones mínimas para que esta regla pueda ocurrir (pueda ser evaluada), que en este caso es que al menos se haya votado en tres preguntas con un mismo CMO. La implementación de esto se puede ver en el Código 7-4.

Código 7-4: Sobrescribir el método canEvaluate de la interfaz IRule

```
    @Override public boolean canEvaluate(Object... params) {
        filter.setParams((NamedValuePair<String>) params[0]);
        return getInSource(
            EmpathicConstants.DS_VOTE_RESULTS,
            QueryOptions.FILTER, filter).size() > 2;
    }
```

Una vez que ya se verificó que la regla es evaluable, se evalúa. Para esto se sobrescribe el método `evaluateImpl`, también de la clase `AbstractRule`. Este método es el responsable de determinar el valor definitivo de una regla. En Código 7-5 se presenta este método, a continuación se indica paso a paso lo que se hace. En las líneas 5, 6 y 7 del método sobrescrito se busca en la fuente de datos local de nombre `EmpathicConstants.DS_VOTE_RESULTS` las respuestas a preguntas que cumplen con un filtro, que en este caso es aquellas que apuntan a un mismo CMO. En la línea 9 se cuenta el número de elementos que se obtuvieron mediante la consulta. En el bucle entre las líneas 12 y 15 se toman las respuestas y se suma el valor (puntaje) de cada respuesta, se asigna 1 para las correctas y 0 para las incorrectas, por lo que la suma de los puntajes da la cantidad total de respuestas acertadas. Entre las líneas 18 y 24 se cargan los parámetros para hacer la consulta que se realiza entre las líneas 26 y 28. Aquí es importante notar la similitud respecto a las líneas 5, 6 y 7. La consulta realizada en estas tres líneas (5, 6 y 7) se hace a una fuente de datos local, mientras que la que se hace en las líneas 26, 27 y 28 es a una fuente de datos externa. La interfaz utilizada es la misma, lo que hace más transparente el proceso de filtrado de la información. En las líneas restantes del método se calculan las estadísticas de rendimiento. El único punto de interés restante está en las líneas 38 y 42 donde se asignan valores que se mostrarán en el mensaje de retroalimentación.

Código 7-5: Sobrescribir método evaluateImpl de la clase AbstractRule

```
01.@Override public double evaluateImpl(Object... params) {
02.     int total = 0;
03.     int ok = 0;
04.     filter.setParams((NamedValuePair<String>) params[0]);
05.     List<VoteResult> scores =
06.         getInSource(EmpathicConstants.DS_VOTE_RESULTS,
07.     QueryOptions.FILTER, filter);
08.     // se cuentan los elementos de esta sesión que pasan el filtro
09.     total = scores.size();
10.     SagdeMetadata metadata = new SagdeMetadata();
11.     // se suma el puntaje de todos, 1 si acertó, 0 si no
12.     for (VoteResult score: scores){
13.         ok += score.getValue();
14.         metadata = score.getSagdeMetadata();
15.     }
16.     /* se listan los parámetros
17.      * para consultar a la fuente de datos externa */
18.     IQueryCriterion<Integer> filterUser =
19.         new QueryCriterion<Integer>
20.             (0,
21.             (NamedValuePair<String>) params[0],
22.             (NamedValuePair<String>) params[1],
23.             (NamedValuePair<String>) params[2]
24.         );
25.     // se consulta a la fuente de datos externa
26.     List<Integer> historic =
27.         getInSource(EmpathicConstants.DS_CMO_PERFORMANCE_INCREASE,
28.         QueryOptions.FILTER, filterUser);
29.     int historicPerformance = 0.5;
30.     // si no hay información previa se asume 50% rendimiento
31.     if (historic.size() > 0) historicPerformance = historic.get(0);
32.     // se calcula el rendimiento de la sesión
33.     int performance = (int)((ok*100.0)/total);
34.     double increase =
35.         historicPerformance > 0 ? (performance - historicPerformance + 0.0) /
36.     historicPerformance : 1.0;
37.     // se asigna un valor para el mensaje
38.     putValue(INCREASE, ((int) (increase*100)));
39.     try{
40.         NamedValuePair<String> namecmo =
41.             (NamedValuePair<String>) params[3];
42.         putValue(CMO, namecmo.getValue());
43.     } catch (Exception e){
44.         putValue(CMO, "un cmo");
45.     }
46.     getMessage().getContext().getData().clear();
47.     getMessage().getContext().getData().add(metadata);
48.     return total > 0 ? increase : 0;
49.}
```

Hasta ahora no se ha especificado la declaración del mensaje a mostrar. Para hacer esto se extiende la clase `AbstractMessage`, como se muestra en el Código 7-6.

Código 7-6: Extensión de la clase AbstractMessage

```
public class CmoPerformanceRuleMessage
extends AbstractMessage{
@Override
public String getUnfilteredText() {
       return "Has mejorado un "+key(INCREASE)+
" % tu rendimiento en "+key(CMO)+"!!!";
}
```

```
@Override
public String getName() {
	return "mCmoPerformance00";
}
@Override
public String getEmotionName() {
	return "happiness";
}
}
```

Finalmente para cargar esta nueva regla, debe agregarse en el método `getRules` del `AbstractEmpathicPlugin` que se carga al comienzo de la aplicación como se muestra en elCódigo 7-7.

Código 7-7: Agregar regla al AbstractEmpathicPlugin

```
@Override
public Collection<IRule> getRules() {
List<IRule> rules = new ArrayList<IRule>();
rules.add(new CmoPerformanceIncreaseRule());
return rules;
}
```

Con eso la regla está cargada y operativa, el árbitro la carga de forma automática y la lee, además de tener acceso a las distintas fuentes de datos.

7.3.3 Mejoras usando información de sesión

Las reglas que utilizan información de sesión se agregan de la misma forma que las que utilizan información histórica. Para estas reglas se decidió dar un enfoque que premie el desempeño del estudiante.

Se agregaron tres reglas basadas únicamente en el rendimiento propio:

1. La primera de estas reglas verifica si hay "secuencias" de buenas respuestas. Si hay tres o más preguntas consecutivas contestadas de forma correcta entonces esta regla se activa.
2. La segunda regla se activa cuando el porcentaje de respuestas correctas supera al 80%.
3. Una tercera regla que se activa cuando la primera respuesta es correcta.

Adicionalmente se probaron tres reglas que comparan el rendimiento del estudiante con el de sus pares en la sesión. Una de ellas respecto a toda la sesión, otra respecto a un CMO específico durante la sesión y finalmente una para la última pregunta respondida.

8 Sexta etapa: Transcripción del trabajo a Javascript

8.1 Descripción

Tras la implementación del framework y las mejoras al Sistema de Votación, surgió la opción de traspasar el framework a Javascript. La motivación de este cambio está en los actuales desarrollos que se están haciendo en la empresa con HTML5. Esta tarea no estaba considerada inicialmente en el plan de trabajo, pero se realizó porque se consideró una buena forma de ver cuánto del diseño podía preservarse durante una migración. Se reconoce, además, el potencial de contar con este framework implementado en múltiples lenguajes y como esto favorece la posibilidad de desarrollar nuevas aplicaciones para varios ambientes.

8.2 Rediseño e implementación

8.2.1 Consideraciones generales

Para poder llevar a cabo una migración de un lenguaje a otro, siempre es necesario tomar en cuenta las diferencias entre ambos lenguajes. Para el caso de Java y Javascript las diferencias son muchas. La siguiente tabla presenta algunas de estas diferencias.

Tabla 8-1: Comparación Java - Javascript

	Java	Javascript
Tipos	Estáticos	Dinámicos
Compilado	Compilado	Interpretado
Cantidad de threads	Multithread	SingleThread, paralelismo simulado con Workers.
Paradigma	Orientado a objetos	Múltiples
Métodos son objetos	No	Si

8.2.2 Programación orientada a objetos en Javascript

A pesar de que Javascript no es estrictamente un lenguaje orientado a objetos es posible simular esto. No es intención de este trabajo abarcar de forma amplia el cómo abordar esta temática y por lo tanto se presentará solamente una posible aproximación.

Para empezar, los objetos en Javascript se definen como funciones. Para declarar un nuevo "tipo" A, se debe usar un código como el que se muestra en Código 8-1 mientras que para definir un método para el tipo A, se debe usar el Código 8-2.

Código 8-1: Definición de un constructor en Javascript

```
function A(){
	// esto es un constructor
}
```

Código 8-2: Definición de un método para un tipo en Javascript

```
A.prototype.myFunction = function () {
	// hacer algo
}
```

En el código anterior se aprecia el uso de una propiedad `prototype`. En Javascript cada objeto está asociado a un `prototype` que se origina a partir de su constructor. Cada vez que Javascript necesita evaluar una propiedad, comienza por buscarla en la instancia del objeto, si no la encuentra, pasa a buscarla a su `prototype`.

Para poder hablar de programación orientada a objetos se necesita herencia. Un subtipo `B` de un tipo `A` debiese heredar las propiedades que este tipo tiene. La forma más directa es asociar el `prototype` del tipo `B` al del tipo `A` (ver Código 8-3). Un tipo `B` puede agregar sus propios métodos y/o sobrescribir los existentes. Para llevar a cabo esto de forma más sencilla se utilizó la implementación que se muestra en el Código 8-4.

Código 8-3: Simulación de programación orientada a objetos en Javascript

```
B.prototype. = new A;
B.prototype.constructor = B;
function B(){
    // esto es un constructor
}
// nuevo método
B.prototype.myFunctionB = function () {
    // hacer algo
}
// sobreescribir
B.prototype.myFunction = function () {
    // hacer algo
}
```

Código 8-4: Simulación de herencia en Javascript

```
Function.prototype.subclassOf = function( parentClassOrObject ){
    if ( parentClassOrObject.constructor == Function )  {
        this.prototype = new parentClassOrObject;
        this.prototype.constructor = this;
        this.prototype.parent = parentClassOrObject.prototype;
    }  else  {
        this.prototype = parentClassOrObject;
        this.prototype.constructor = this;
        this.prototype.parent = parentClassOrObject;
    }
    return this;

}
```

8.2.3 De multiples threads a observer

Parte importante de la estructura del framework es precisamente la capacidad de realizar diversas tareas en paralelo. Una de las principales aplicaciones de esto, es la evaluación de las distintas reglas y los accesos a fuentes de datos por parte de las mismas. En Javascript, hasta la fecha, no es posible hacer una aproximación similar.

Javascript es un lenguaje que se ejecuta en un único *thread* y por lo tanto el que una tarea se realice de manera asíncrona es puramente simulado. Esto tiene implicancias importantes pues no permite dejar a un *thread* en segundo plano esperando por un resultado. A priori, es imposible saber cuánto demorará en ejecutarse una tarea como el evaluar una regla; bloquear el control de la página para esperar un resultado no es una opción.

Para enfrentar este problema se decidió hacer una aproximación por medio del patrón de diseño *Observer*. En lugar de esperar de forma activa por un resultado (que implicaría hacer un

bloqueo) se espera por éste de manera pasiva. El que Javascript permita tratar a las funciones como objetos hace más fácil el poder asignar métodos que manejen los eventos (*handlers*) para cuando una tarea termina de ejecutarse.

Escoger este enfoque conlleva por un lado el cambiar de forma brusca el paradigma utilizado en la implementación realizada en Java. Sin embargo, esta forma de trabajar no es extraña para los desarrolladores de aplicaciones web. Un manejo similar de eventos es el que se hace cuando se realiza una petición asíncrona.

Existen diversos puntos en los que es deseable esperar el resultado de un método (en los que no se sabe cuánto tardará en tenerse este resultado):

1. En un `Arbiter` al solicitarle escoger una regla.
2. En un `ArbiterCriterion` al procesar las distintas reglas.
3. En un `EmpathicRule` al ver si es evaluable, al evaluarla y al verificar si es seleccionable.
4. En un `DataSource` al realizar una operación de búsqueda, actualización, inserción y eliminación.

Para todos estos casos se agregó la opción de suscribirse (observar) el resultado. Los demás cambios en el código son muy menores por lo que no requieren una cobertura especial en este trabajo.

9 Conclusiones

9.1 Resultados obtenidos

En el presente trabajo se diseñó y desarrolló un framework para la entrega de retroalimentación a usuarios de software educativo. Dicha retroalimentación está basada en información de rendimiento y uso de software educativo por parte del usuario.

La primera etapa en el desarrollo de este trabajo correspondió a la identificación de la información de interés que será utilizada para la generación de mensajes de retroalimentación para los estudiantes. Esta etapa cumplió con un doble rol pues, además, se definió lineamientos para el diseño e implementación de un framework con las características que se solicitan. Respecto a la información a utilizar se decidió que era importante usar aquella que está orientada al desempeño del estudiante. Se puede considerar tanto información histórica del estudiante, como información recabada durante la misma sesión de trabajo en que se presenta la retroalimentación.

La segunda etapa en el proceso fue la del diseño propiamente tal. Para el diseño se utilizó el paradigma de programación orientada a objetos con el objetivo de que el framework fuera extensible y permitiera a agregar contenido nuevo sin mayores complicaciones. Se identificaron cuatro elementos fundamentales para desarrollar este framework: los mensajes de retroalimentación a presentar; cómo elegir el mensaje más adecuado para presentar; en qué contexto debe presentarse un mensaje; y de dónde se obtiene la información para evaluar dicho contexto. A cada uno de estos elementos se le asignó un módulo en particular del framework: mensajes, selección de mensajes, eventos o reglas y fuentes de datos.

La tercera etapa realizada fue la de implementación del diseño. Una vez finalizada esta implementación se dio paso a una etapa de evaluación del framework. Esta evaluación se llevó a cabo por medio del cálculo de métricas de software y de un test de usabilidad del framework.

Para el caso de las métricas se utilizó la herramienta JDepend. JDepend permite medir la calidad de un diseño y su extensibilidad, reusabilidad y mantenibilidad, observando los distintos paquetes del mismo. Las principales métricas a evaluar son la de abstracción y de inestabilidad. El valor de ambas métricas va desde cero a uno. Se calcula finalmente una métrica que corresponde a la distancia (perpendicular) a la recta ideal. En un diseño ideal, se tiene que Abstracción + Inestabilidad = 1. Dados los valores indicados por la herramienta, un buen diseño tiene una distancia menor o igual a 0,1. En el caso del framework desarrollado, dicho valor asciende a 0,24. Este valor puede deberse a factores tales como que se implementaron pocas clases concretas, lo que elevaría el nivel de abstracción del framework. Por otro lado, se agregó clases que facilitaran el desarrollo. Dichas clases aumentan el acople que hay entre las clases y, por lo tanto, aumenta la inestabilidad.

La segunda parte de la evaluación consistió en un test de usabilidad. Este test se llevó a cabo con una actividad en la que dos profesionales y una estudiante del área de computación realizaron un grupo de tareas utilizando el framework. Los resultados obtenidos en este caso se consideran favorables. Salvo por la documentación que fue evaluada como el punto más bajo del framework, los encuestados consideraron que el framework no era de difícil uso, evaluando con nota igual a superior a 6 (en escala de 1 a 7, con 1 muy en desacuerdo y 7 muy de acuerdo) a los

ítems: Usaría el framework en mi aplicación, el framework es fácil de usar y el framework es fácil de extender. Tras la evaluación se utilizó el framework para hacer mejoras a un sistema ya existente de la empresa, no presentándose inconvenientes en el desarrollo.

Finalmente, se transcribió el framework a Javascript con el fin de poder utilizarlo en aplicaciones web. Esta migración implicó hacer diversos cambios en el framework dada la distinta naturaleza de Java y Javascript.

Resumiendo lo expuesto anteriormente, el trabajo originó un framework que permite presentar a los estudiantes retroalimentación basada en su información de rendimiento y del uso que ha tenido de software educativo. El framework se encuentra implementado en Java y Javascript y fue evaluado tanto por métricas de diseño como por usabilidad. Los resultados de la primera evaluación son satisfactorios, pero a la vez manifiestan que el diseño puede ser mejorado ya sea proveyendo más clases concretas o disminuyendo el acople de las clases existentes. Por otra parte, se obtuvo una buena recepción en personas con experiencia en desarrollo. Los entrevistados evaluaron positivamente elementos distintos elementos del diseño, además de la buena utilización de nombres para métodos y clases. Se rescata principalmente la buena evaluación que recibieron los puntos de la facilidad de uso del framework y su extensibilidad, siendo este último, uno de los principales objetivos de este trabajo.

Este trabajo abre la posibilidad de implementar de forma estructurada herramientas que presenten retroalimentación a usuarios de software educativo y de registrar está retroalimentación junto a la recepción que ha tenido ésta en los usuarios. De esta forma es posible implementar mejoras para software educativo de manera más eficiente y además permite estudiar a futuro que tan efectiva es la retroalimentación en términos motivacionales y pedagógicos, abriendo la posibilidad de investigar con más detalle estos aspectos de la informática educativa.

9.2 Trabajo futuro

Existen múltiples elementos del presente trabajo que pueden extenderse y/o mejorarse, entre ellos se encuentran:

- Implementar soporte '*out-of-the-box*' para bases de datos de amplio uso como por ejemplo PostgreSQL y MySQL. Actualmente hay que incluir y configurar manualmente el driver de la base de datos.
- Implementar otros tipos de fuentes de datos, como por ejemplo leyendo de archivos. Este tipo de extensión podría incluir soporte para múltiples formatos de archivos.
- Implementar un gestor de interfaz de usuario usando Swing. El framework incluye uno predeterminado que escribe en la consola, pero no despliega mayor retroalimentación gráfica.
- Implementar configuración de las reglas usando archivos .properties o cargando información desde algún servidor. Actualmente el framework provee los mecanismos para configurar las reglas a través de anotaciones.
- Estudiar e incluir más criterios para la selección del mensaje de retroalimentación que se presentará. El framework actualmente incluye uno aleatorio y uno que utiliza el mensaje menos usado. Hay muchas opciones para extender esto, pudiéndose priorizar distintas necesidades dependiendo del contexto.

- Complementando el punto anterior, dado que el framework almacena información de los mensajes generados, es posible estudiar cuáles son los mensajes que dan mejores resultados para los estudiantes.
- En un aspecto relacionado a lo anterior, pero un poco más ajeno a este trabajo, la misma información que se menciona permitiría ver en que elementos los estudiantes tienen un mejor rendimiento y reciben más retroalimentación.

10 Referencias

[1] M. Martyn, «The Hybrid Online Model: Good Practice,» *EDUCAUSE Quarterly 1,* 2003.

[2] M. Ben Ammar, A. M. Alimi, M. Neji y G. Gouardères, «Agent-based collaborative affective e-learning system,» de *First International Conference on Immersive Telecommunications*, Brussels, Belgium, 2007.

[3] H. Wang, M. Chignell y M. Ichizuka, «Empathic Tutoring Software Agents Using Real-time Eye Tracking,» de *Symposium on Eye tracking research & applications*, New York, NY, USA, 2006.

[4] M. Ochs, C. Pelachaud y D. Sadek, «An Empathic Virtual Dialog Agent to Improve Human-Machine Interaction,» de *International joint conference on Autonomous agents and multiagent systems*, Richland, SC, 2008.

[5] S. W. McQuiggan y J. C. Lester, «Learning Empathy: A Data-Driven Framework for Modeling Empathetic Companion Agents,» de *International joint conference on Autonomous agents and multiagent systems*, New York, NY, USA, 2005.

[6] R. Araya, Las emociones, motivaciones y fuentes cognitivas de placer en el aprendizaje de la matemática. Curso "Estrategias para la Enseñanza de la Matemática", OEA, 200-.

[7] R. Araya, Fuentes tecnológicas e interpersonales de placer, motivación y atracción. Curso "Estrategias para la Enseñanza de la Matemática", OEA, 200-.

[8] Blackboard Inc., «Blackboard Inc.,» Noviembre 2009. [En línea]. Available: http://www.blackboard.com/CMSPages/GetFile.aspx?guid=be3a6c2a-ea0f-47c3-8148-8cf4caedd40b. [Último acceso: 31 Julio 2011].

[9] P. Maes, «Artificial Life Meets Entertainment: Life like Autonomous Agents,» *Communications of the ACM,* vol. 38, nº 11, pp. 108-114, Noviembre 1995.

[10] S. Franklin y A. Graesser, «Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents,» de *Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, Londres, 1996.

[11] T. Alexander, «An Optimized Fuzzy Logic Architecture for Decision-Making,» de *AI Game Programming Wisdom*, Hingham, Massachusetts, Charles River Media Inc., 2002, pp. 367-374.

[12] D. Isla y B. Blumberg, «Blackboard Architectures,» de *AI Game Programming Wisdom*, Hingham, Massachusetts, Charls River Media Inc., 2002, pp. 333-344.

[13] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design Patterns: Element of Reusable Object Oriented Software, Adisson-Wesley, 1995.

11 Anexos

11.1 Evaluación: Encuesta de Usabilidad

La siguiente encuesta tiene por objetivo evaluar el nivel de satisfacción de los usuarios finales (desarrolladores en este caso) en el uso del framework. Esta encuesta se divide en dos partes que tienen relación con su experiencia con el framework: 1) una sección de preguntas que se califican de acuerdo a una escala numérica, y 2) una sección de preguntas abiertas.

Las preguntas apuntan a medir las siguientes dimensiones en relación a la experiencia del usuario:

- Satisfacción en general: Qué tan conforme está el usuario con el framework.
- Facilidad de uso: Qué tan fácil es para el usuario utilizar el framework con su ejecución predeterminada.
- Facilidad de extensión: Qué tan fácil es para el usuario incorporar nuevos elementos que extiendan los ya existentes en el framework.

En esta encuesta se usan algunos términos para identificar algunos elementos presentes en el framework, los cuales se definen a continuación:

- Mensaje: Es la información que se usa para presentar la retroalimentación al usuario final.
- Regla: Evalúan un indicador basado en la información existente en las fuentes de datos.
- Criterio: Mecanismo para seleccionar cual es la Regla para a que se presentará un Mensaje.

11.1.1 Primera parte

En esta sección se le solicita asignar una calificación a cada una de las sentencias (preguntas). La escala de calificación es de 1 a 7, donde cada valor corresponde a:

1. Muy en desacuerdo
2. En desacuerdo
3. Levemente en desacuerdo
4. Indiferente
5. Levemente de acuerdo
6. De acuerdo
7. Muy de acuerdo

PREGUNTA	RESPUESTA (1 al 7)
USO GENERAL DEL FRAMEWORK	
Me gustó el framework	
Usaría el framework en mi aplicación	
El framework es fácil de usar	
El framework es fácil de extender	

SISTEMA DE PLUGINS DEL FRAMEWORK
El sistema de plugins del framework es útil
Los plugins son fáciles de usar
PROGRAMACIÓN/IMPLEMENTACIÓN CON EL FRAMEWORK
Programar/Implementar un criterio es sencillo
Programar/Implementar una regla es sencillo
Programar/Implementar un mensaje es sencillo
DOCUMENTACIÓN DEL FRAMEWORK
La documentación de las clases es clara
La documentación es completa
La documentación es útil
NOMBRES DE LOS ELEMENTOS DEL FRAMEWORK
Los nombres de las clases son representativos (auto-explicativos)
Los nombres de los métodos son representativos (auto-explicativos)
Los nombres de las propiedades son representativos (auto-explicativos)

11.1.2 Segunda parte

En esta sección se le solicita responder 12 preguntas abiertas, donde debe responder abiertamente de acuerdo a su opinión personal en los diferentes aspectos que son consultados en relación a su experiencia con el framework.

11.1.2.1 Sintaxis

1. ¿Cómo considera la sintaxis (engorrosa, sencilla, etc.)? ¿Por qué?
2. La ayuda otorgada por la documentación y los nombres de los distintos elementos (clases, métodos, etc.). ¿Es útil? ¿Facilita la elaboración del código?
3. ¿Considera que algún elemento es redundante? ¿Considera que se necesita algún elemento adicional?

11.1.2.2 Diseño

4. ¿Considera que el framework ayuda a realizar una separación entre la lógica de las reglas y la lógica propia de la aplicación?
5. ¿Qué le parece el diseño del framework? ¿Es muy complejo? ¿Tiene muchas clases, métodos y propiedades? ¿facilita su extensión?

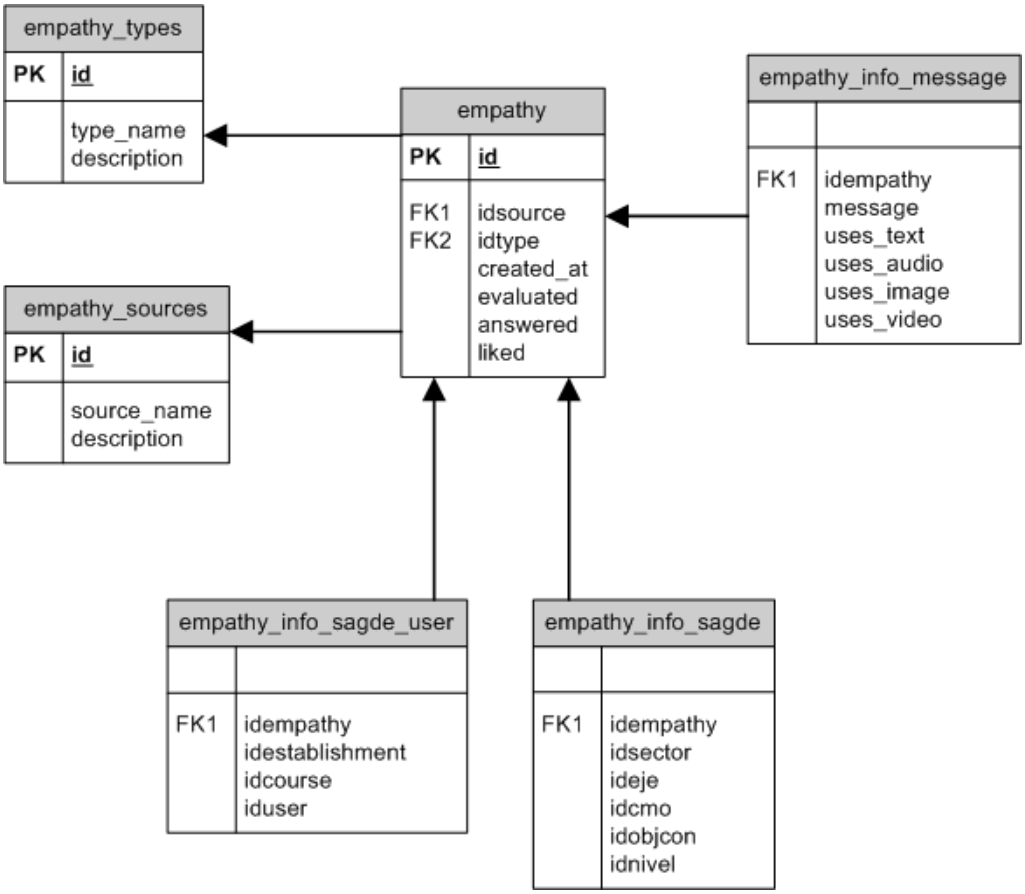
11.1.2.3 Selección de elementos

6. ¿Qué destaca del framework?
7. ¿Qué mejoraría del framework?
8. ¿Qué agregaría al framework?

11.1.2.4 Preguntas abiertas

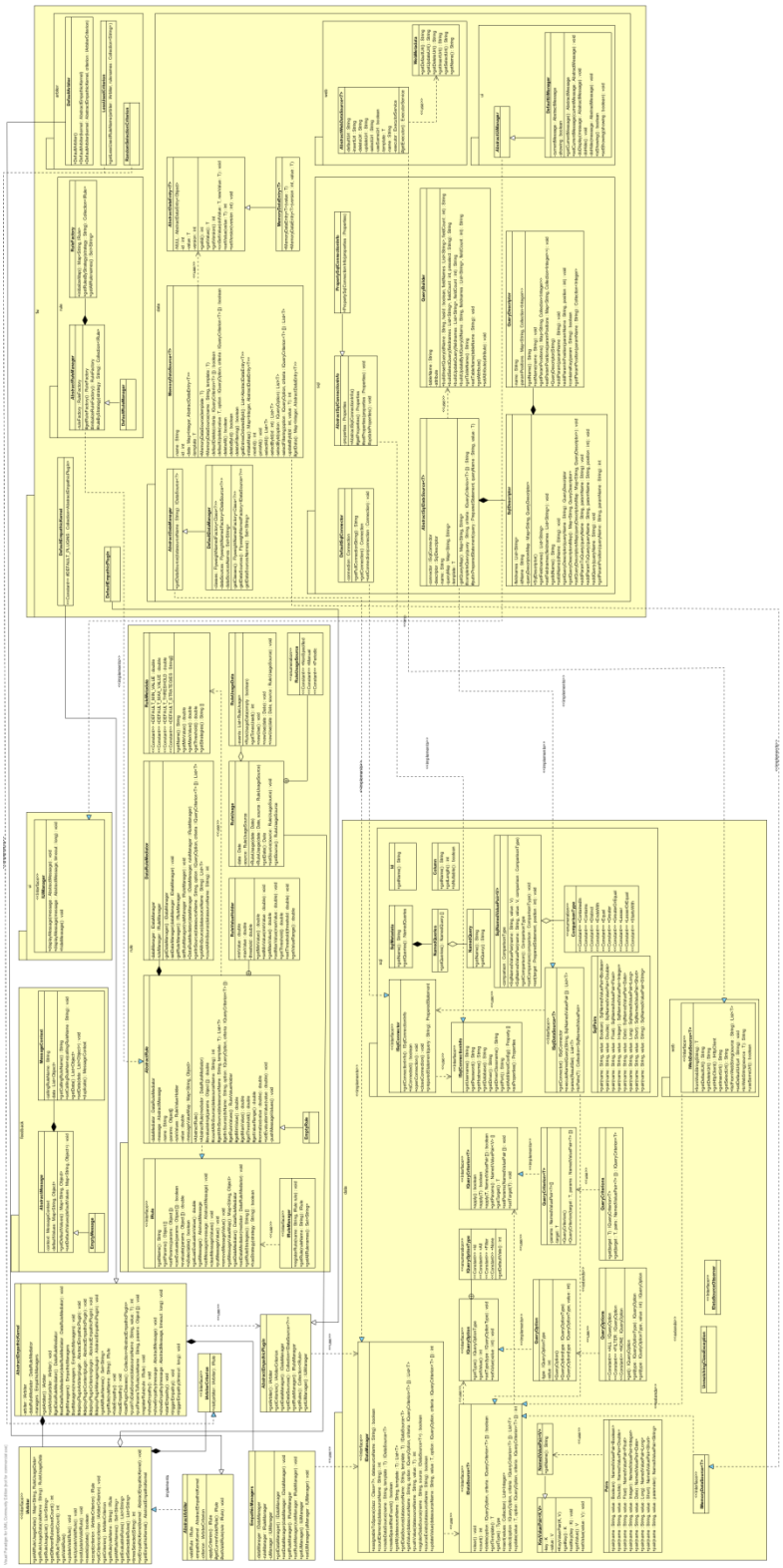
9. ¿Qué fue lo que más le gustó del framework?
10. ¿Qué fue lo que menos le gustó del framework?
11. ¿Qué otro uso le daría al framework?
12. Algún otro comentario que agregar

11.2 Diseño: Modelo de datos

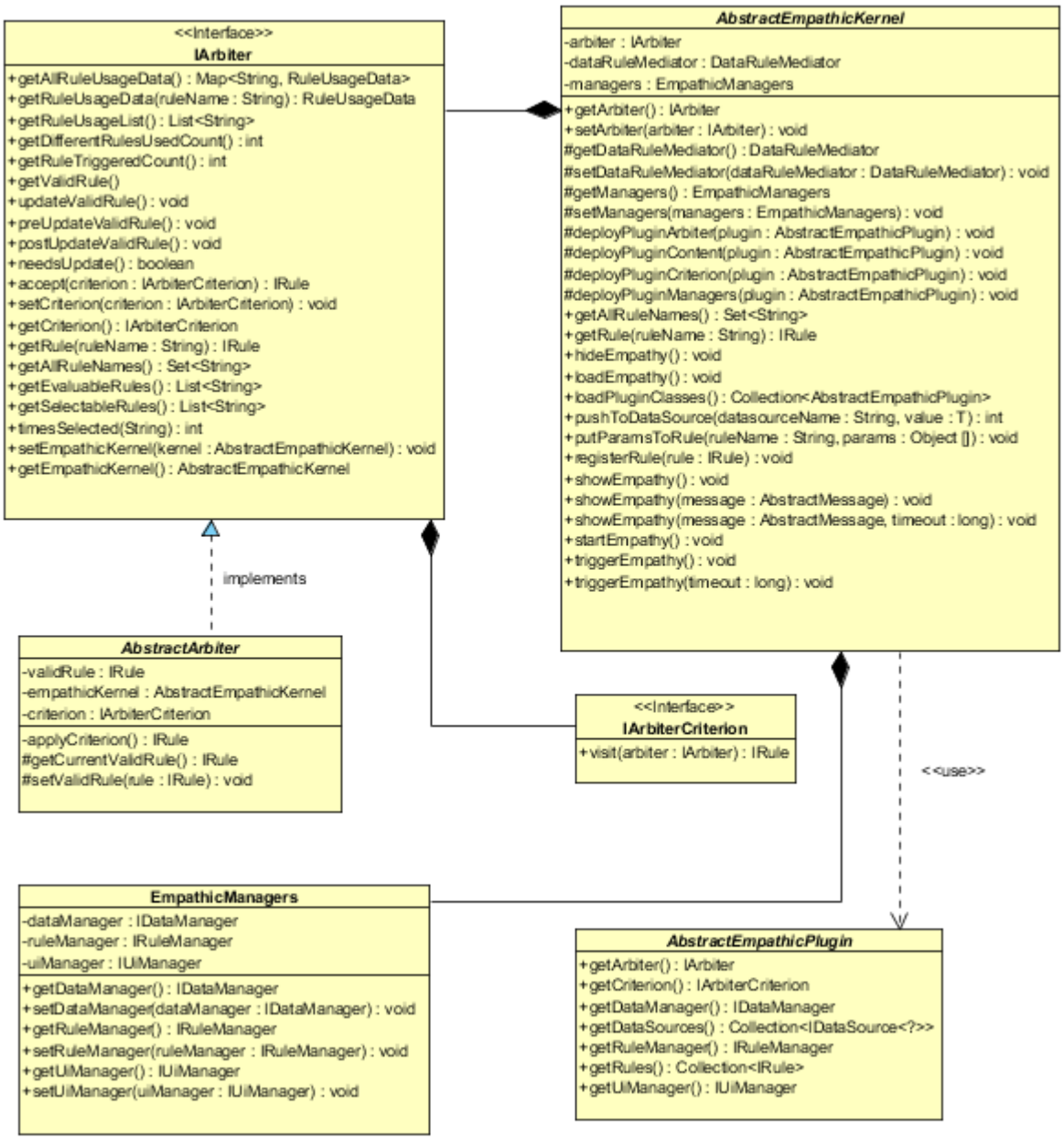


11.3 Diseño: Diagramas de clases

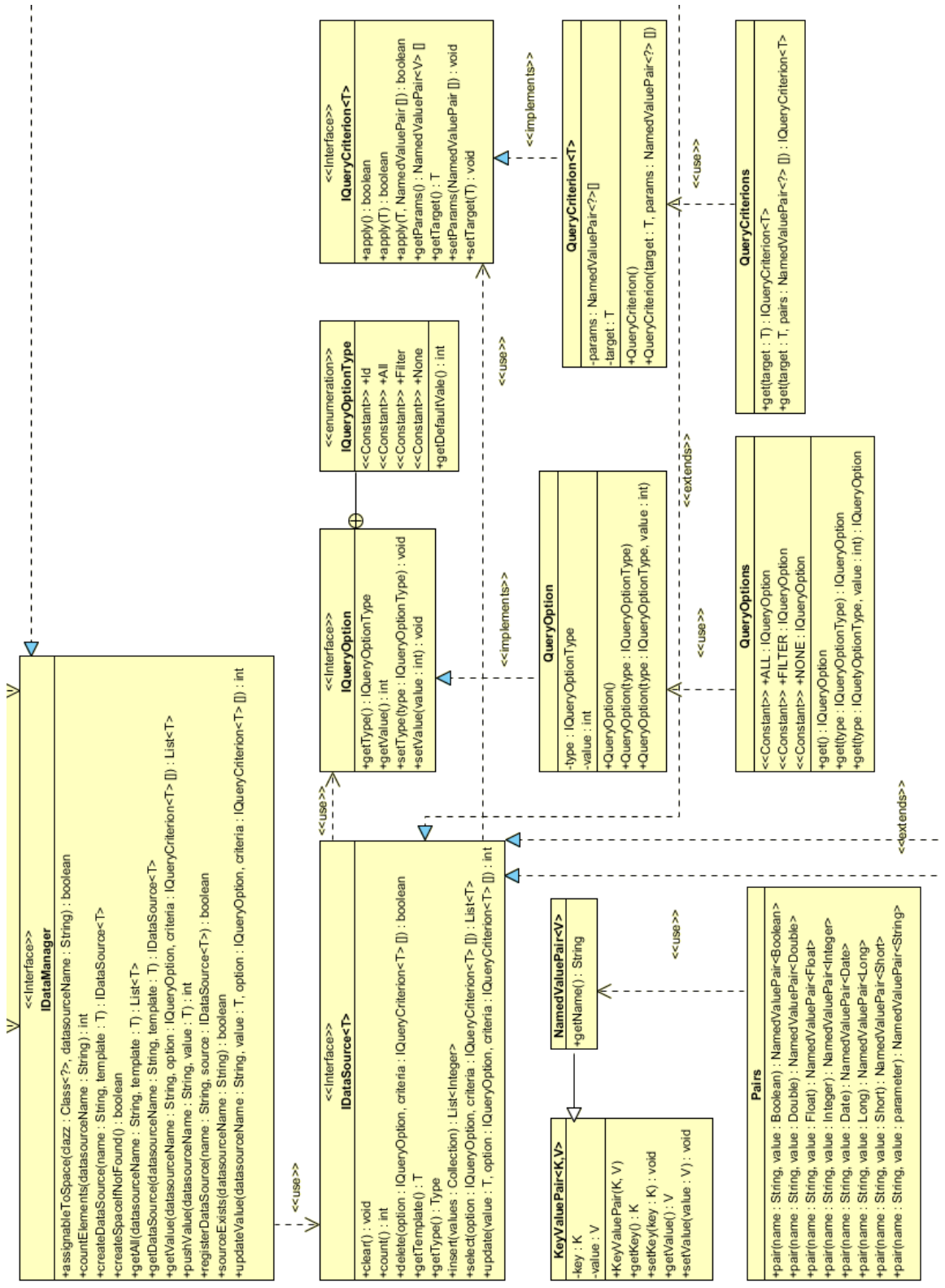
11.3.1 Diagrama completo



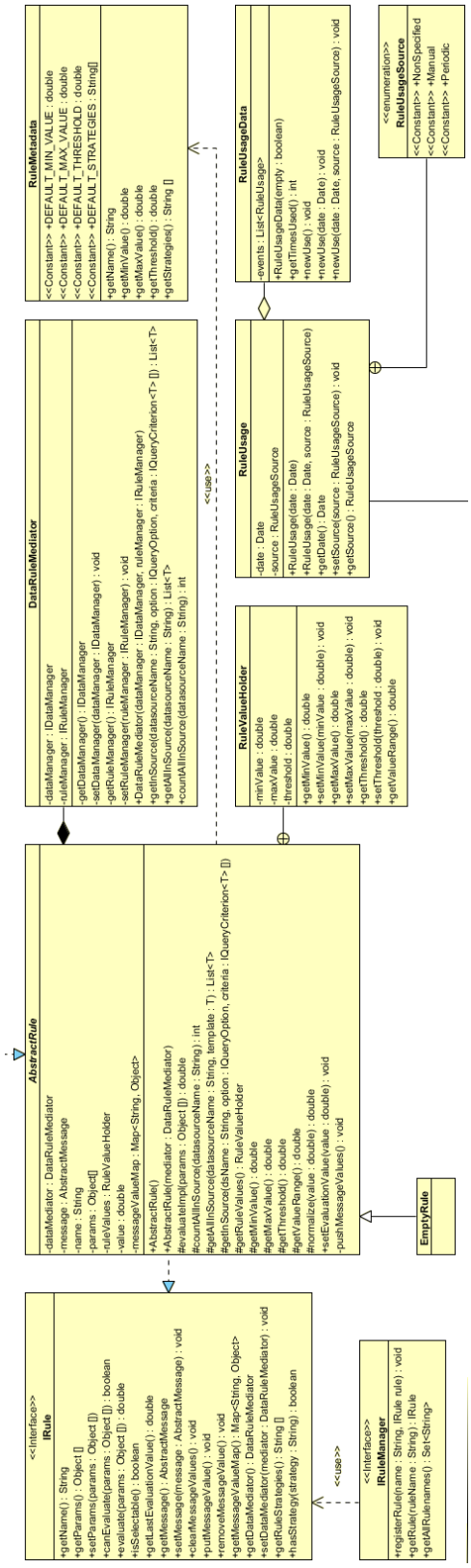
11.3.2 Clases principales: AbstractEmpathicKernel y IArbiter



11.3.3 Paquete data en EmpathicCore



11.3.4 Paquete rule en EmpathicCore



11.3.5 EmpathicFw