AIBench: framework de programación para la enseñanza de técnicas de IA en tercer ciclo

R. Domínguez, D. Glez-Peña, J. R. Méndez, F. Fdez-Riverola
ESEI: Escuela Superior de Ingeniería Informática
Universidad de Vigo
Campus Universitario As Lagoas s/n, 32004, Ourense
rubendc@sing.ei.uvigo.es, {dgpena, moncho.mendez, riverola}@uvigo.es
F. Díaz
Escuela Universitaria de Informática
Universidad de Valladolid
Plaza Santa Eulalia, 9-11, 40005, Segovia
fdiaz@infor.uva.es

Resumen

En este trabajo se presenta AIBench, un framework que agiliza la codificación, ejecución, prueba y optimización de técnicas de Inteligencia Artificial (IA). AIBench está pensado para su utilización por parte de alumnos de tercer ciclo, involucrados en programas de doctorado donde se imparten materias que implican la comprensión e implementación de distintas técnicas de IA. La finalidad del framework desarrollado es liberar al alumno de las labores de programación repetitivas y propensas a errores que están relacionadas, entre otras, con la entrada de datos, diálogos de configuración de parámetros o presentación de resultados, permitiéndole centrarse únicamente en los algoritmos objeto de estudio.

1. Introducción y Motivación

El objetivo principal de un programa de doctorado es la formación del alumno en distintas áreas de investigación y su capacitación final para llevar a cabo el desarrollo posterior de una tesis doctoral. En este contexto, la finalidad de las distintas materias (primer curso) y trabajos (segundo curso) que componen el itinerario de tercer ciclo, es la de proporcionar una rigurosa base científica y una visión global sobre un amplio espectro de técnicas y algoritmos de investigación aplicados a distintos campos.

En este sentido, el programa de doctorado TADSI (*Tecnologías Avanzadas para el Desarrollo de Software Inteligente*) impartido en el Departamento de Informática de la Universidad de Vigo, tiene como finalidad la especialización del alumno en técnicas de investigación relativas al desarrollo de software usando técnicas basadas en IA e ingeniería de software avanzada [1].

Formando parte del mencionado programa de doctorado, las líneas prioritarias que ofertan los miembros del grupo de investigación SING (*Sistemas Informáticos de Nueva Generación*) [2] engloban a diversos campos de la IA aplicando distintas técnicas para la resolución de problemas de ámbito diferente. En concreto, se puede mencionar la utilización de redes neuronales, lógica difusa, agentes inteligentes o sistemas de razonamiento basado en casos, aplicados a problemas tan diversos como la clasificación de correo spam, el análisis del genoma humano o el desarrollo de sistemas de detección de intrusos (IDS, *Intrusion Detection System*), entre otros.

En el caso concreto de las distintas materias existentes en el primer curso del bienio de doctorado, es habitual la asignación al alumno de tareas de implementación de técnicas concretas de IA que complementen el contenido más teórico de los cursos. Por lo tanto, la situación actual existente en el curso de doctorado y en el seno del grupo SING, es que como resultado de cada trabajo de investigación realizado por parte de los alumnos y/o investigadores, se han venido desarrollando un conjunto de aplicaciones software que han sido implementadas de forma independiente e inconexa.

Siguiendo este método de trabajo, además del esfuerzo requerido en cada caso para el estudio de las distintas técnicas, es necesario dedicar una cantidad considerable de tiempo extra para el

desarrollo "desde cero" de la herramienta software final de soporte y prueba. Considerando a mayores el hecho de que los alumnos del curso de doctorado posteriormente se integran en los distintos grupos de investigación existentes en el departamento, los beneficios esperados de la utilización de un entorno común de programación de carácter integrador para tercer ciclo y grupos de investigación, se vislumbran a todas luces enriquecedores.

Teniendo en cuenta lo comentado anteriormente, se ha realizado un análisis de las distintas herramientas software desarrolladas por el grupo SING, así como otras alternativas disponibles en Internet (p.ej. Weka [6]), con el objetivo de identificar un patrón común acerca de las funcionalidades que las distintas técnicas de IA demandan de forma repetitiva. Resultado de este proceso de análisis, se ha observado que en cualquiera de las herramientas existentes el flujo de actividades que se ejecutan reiteradamente es el siguiente: carga del conjunto de datos de prueba, preprocesamiento, configuración de parámetros, ejecución de técnicas y visualización de resultados.

En este contexto y como solución a la problemática existente nace AIBench [3] con un doble objetivo: (*i*) eliminar el trabajo repetitivo que supone el desarrollo de toda aplicación de soporte a la investigación y (*ii*) ofrecer al alumno y/o investigador una serie de facilidades o servicios de los que puede hacer uso en la codificación de su técnica, facilitando con ello la reutilización de componentes software.

En este sentido, AIBench se puede definir como un framework de programación abierto que proporciona los servicios comunes anteriormente citados, liberando al investigador de las tareas comunes de programación relacionadas con la interfaz de usuario, entrada y salida de datos, etc. El programador puede así centrarse únicamente en la codificación de la técnica o algoritmo objeto de estudio. Para las restantes tareas se hará uso de los servicios disponibles en el framework proporcionado. Entre dichos servicios se encuentran ya disponibles distintas utilidades para la importación y exportación de datos en diferentes formatos, paneles genéricos para llevar a cabo la visualización de resultados, formularios parametrizables para captura de datos de usuario, etc.

Una vez adquirido el domino necesario de la plataforma, las principales ventajas para los alumnos de tercer ciclo y los docentes del programa de doctorado se pueden resumir en los siguientes grandes apartados:

- El alumno solamente debe centrarse en los aspectos relativos a la programación de las técnicas objeto de estudio.
- Se genera la posibilidad natural de integración de distintas técnicas entre sí, lo que permite el tratamiento de problemas más complejos o el análisis del mismo problema desde una perspectiva más amplia.
- De forma paralela se lleva a cabo la creación de un repositorio de algoritmos especializados y documentados, que favorece la reutilización de componentes software específicos.
- Permite la existencia de un único punto de integración, ampliación y utilización para la interconexión con otras plataformas de desarrollo estándar.

Mientras que la primera sección ha presentado una breve introducción al trabajo realizado motivando su originalidad e interés, el resto del artículo se estructura como sigue. En primer lugar se presentan las abstracciones básicas empleadas en el diseño de AIBench, necesarias para poder extender y hacer uso de las funcionalidades que ofrece. Posteriormente se describe brevemente la forma en la que el programador de AIBench puede implementar nuevas operaciones o técnicas haciendo uso de los servicios básicos actualmente disponibles. A continuación se detalla la arquitectura interna del framework, haciendo especial hincapié en su comportamiento incremental basado en plugins. Finalmente, se exponen las conclusiones así como el trabajo presente y futuro, citando alguno de los proyectos asociados en los que se está trabajando actualmente.

2. Conceptos Básicos en AIBench

AIBench implementa un framework de programación ligero, no intrusivo y basado en MVC (Modelo Vista/Controlador) que posibilita el desarrollo de aplicaciones JAVA permitiendo:

- La conexión, ejecución e integración de operaciones con una entrada/salida bien definida.
- Diseño independiente del problema, es decir, AIBench no contiene clases relativas a conceptos de Inteligencia Artificial, como

pueden ser: *classifier, clusterer, feature selection, model, genetic search, cross validation,* etc.

- Diseño *default-driven*, es decir, AIBench toma una decisión por defecto para los posibles valores de configuración, buscando que el programador obtenga con el mínimo código posible una versión inicial, pero funcional, de su trabajo.

El framework desarrollado constituye una aplicación modular y fácilmente extensible compuesta por diferentes elementos, denominados plugins. Tanto los plugins que componen el núcleo de AIBench, como aquellos que pueden ser desarrollados por los programadores, se ejecutan bajo el motor de plugins Platonos [4]. A través de esta filosofía de desarrollo se pueden identificar dos roles de usuario relacionados con AIBench: (*i*) los desarrolladores del núcleo o *core* y (*ii*) los programadores de nuevas extensiones (plugins) del framework (habitualmente alumnos de tercer ciclo e investigadores).

En el ámbito de AIBench, bajo el concepto de plugin se engloban los siguientes componentes software:

- **Operaciones** que representan e implementan técnicas o algoritmos concretos.
- **Tipos de datos** que representan instancias del dominio del problema.
- **Vistas** asociadas a los distintos tipos de datos con los que trabaja el framework.

La Figura 1 muestra el aspecto visual de la herramienta donde se guarda un registro de las *operaciones* llevadas a cabo (*Session Tree*), un registro de las instancias de los tipos de datos generados (*Clipboard Tree,* ver punto 2.4), el diálogo generado automáticamente para la entrada de datos de una operación específica (*Generated Dialog*) y una *vista* personalizada para un *tipo de datos* concreto (*Custom View*) (conjunto de microarrays de ADN).

A continuación se exponen en detalle los distintos tipos de plugins existentes en AIBench.

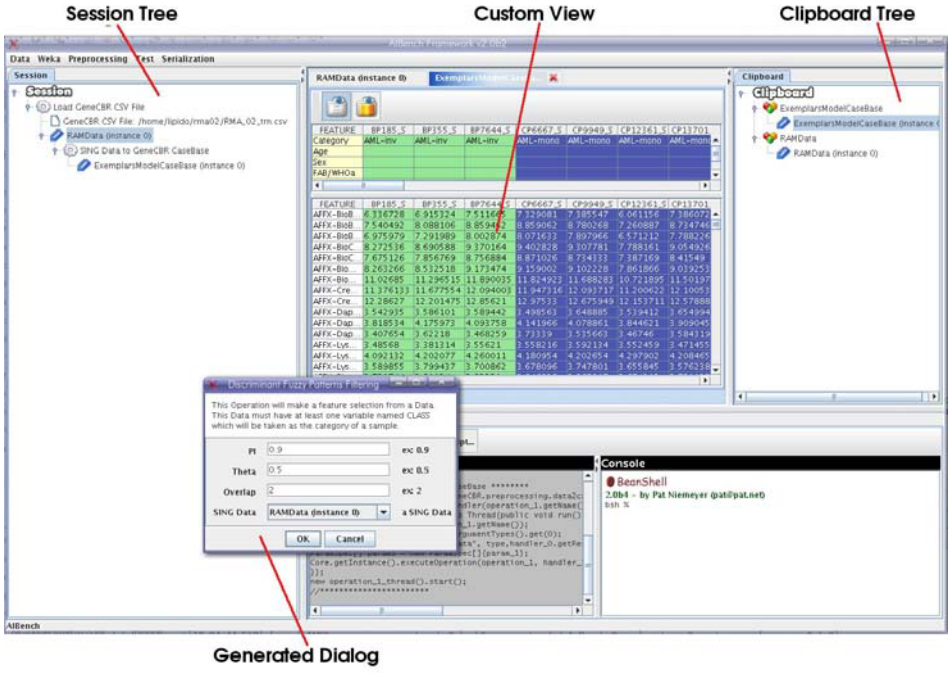


Figura 1. Interfaz de usuario de AIBench durante la ejecución de un experimento

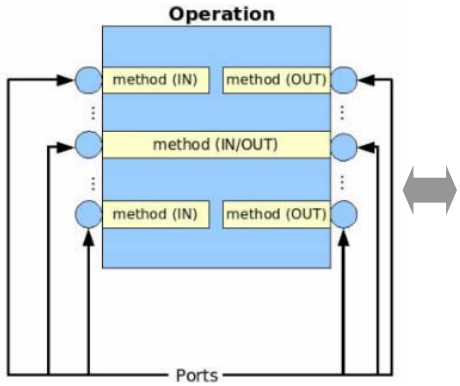
2.1. Operaciones

La extensión natural del framework de AIBench se realiza a través de la codificación de nuevos bloques de código fuente, denominados operaciones. Una operación corresponde con la implementación concreta de una técnica o algoritmo particular.

El modelo subyacente de programación de operaciones en AIBench presupone que:

- La lógica puede ser desacoplada totalmente de la interfaz de usuario.
- La interconexión de operaciones puede ser modelada utilizando el concepto de *experimentos*, que pueden ser ejecutados de forma repetitiva sobre distintos datos.
- El programador está obligado a *pensar antes de programar*, a través de la definición de distintas operaciones.

Para el programador de AIBench una operación es simplemente una clase Java con anotaciones en su código, que definen claramente sus entradas y salidas a través de los correspondientes *puertos* (puertos de entrada, de salida o de entrada/salida). El ejemplo mostrado en la Figura 2 define una sencilla operación que implementa la suma de dos números enteros.




```
@Operation(description="this operation adds two numbers")
public class Sum{
 private int x,y;

@Port(direction=Direction.INPUT, name="x param")
public void setX(int x){
 this.x = x;
}

@Port(direction=Direction.INPUT, name="y param")
public void setY(int y){
 this.y = y;
}

@Port(direction=Direction.OUTPUT)
public int sum(){
 return this.x + this.y;
}
}
```

Figura 2. Modelo abstracto de operaciones de AIBench basado en puertos de entrada, salida y entrada/salida

A través de la utilización de anotaciones, junto con un descriptor de la operación o plugin en formato XML, AIBench dispone de toda la información necesaria para:

- Colocar las operaciones en sus correspondientes menús.
- Generar los cuadros de diálogo necesarios para la introducción de datos.
- Almacenar los resultados obtenidos para presentarlos como posible entrada en posteriores ejecuciones de otras operaciones compatibles con el tipo de dato generado.

2.2. Tipos de datos

Con el objetivo de facilitar la extensión del framework, AIBench ofrece al programador la posibilidad de definir sus propios tipos de datos, que serán utilizados como entrada o salida en las operaciones disponibles. Dichos tipos de datos no heredan ni implementan ninguna clase o interfaz para evitar cualquier tipo de acoplamiento, únicamente serán representaciones en memoria de conceptos relacionados con el dominio del problema. En AIBench se diferencian dos tipos de datos en función de su complejidad y naturaleza:

- Tipos de datos simples, generalmente introducidos por el usuario.
- Estructuras de datos complejas, generalmente obtenidas como resultado de la ejecución de una operación.

2.3. Vistas

En el campo de la IA es muy común el uso de representaciones gráficas de los datos, ayudando al investigador en la evaluación de una determinada técnica. Estas representaciones utilizan distintos formatos que van desde una simple vista tabular, pasando por representaciones de funciones o gráficos de barras, hasta complejas representaciones para el análisis de secuencias del genoma humano. La Figura 3 muestra un ejemplo de este último tipo de vistas.

Figura 3. Vista de datos compleja en AIBench

En la distribución actual de AIBench (AIBench SDK 1.9.1) ya se incluyen vistas por defecto para los tipos de datos que maneja internamente el framework. De este modo, el programador siempre puede hacer uso de estas vistas para visualizar rápidamente los resultados obtenidos con sus técnicas. No obstante, se pueden implementar e incorporar nuevas vistas personalizadas y adecuadas a los tipos de datos que se manejen en cada momento.

2.4. Integración entre operaciones: *clipboard*

AIBench conoce en todo momento el conjunto de datos con los que trabaja el programador, generados desde operaciones. Esto se consigue con la implementación de un espacio de intercambio de datos al que se denomina portapapeles o *clipboard*.

La finalidad última del clipboard es posibilitar la integración entre las distintas operaciones a nivel de intercambio de datos. El clipboard de AIBench actúa del mismo modo que los portapapeles disponibles en la mayoría de los sistemas operativos existentes. Todos los resultados generados tras la ejecución de una operación se incorporan al clipboard, donde se clasifican y catalogan en función de la clase Java a la que pertenecen (tipo de datos que implementan). Con esta estructura de intercambio, los datos generados por unas operaciones estarán disponibles como entrada para posteriores ejecuciones de otras operaciones. La Figura 4 presenta un ejemplo de esta funcionalidad donde "Load CSV" y "Classifier Train" son dos operaciones conectadas a través del clipboard.

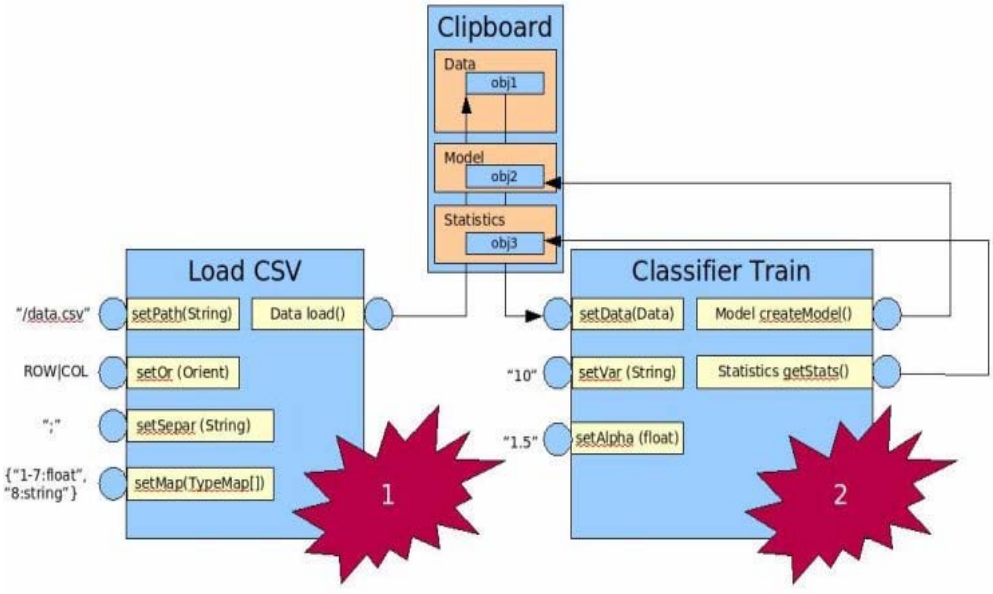


Figura 4. Integración de operaciones en AIBench utilizando el clipboard

3. Servicios de AIBench

3.1. Generación dinámica de la interfaz de usuario

El diseño e implementación de la interfaz de usuario de una aplicación, suele acarrear un esfuerzo considerable en relación a la totalidad del proyecto a desarrollar. Para reducir dicho esfuerzo, AIBench proporciona al programador un espacio de trabajo similar al presentado en la Figura 1, al que dinámicamente se incorporan las operaciones codificadas en sus correspondientes menús. Para conseguir esta funcionalidad sin necesidad de recompilar de nuevo la aplicación, AIBench hace uso de descriptores de operaciones utilizando ficheros con formato XML, así como anotaciones en las clases Java relacionadas con el mecanismo de introspección del que dispone el lenguaje JAVA.

Del mismo modo, los cuadros de diálogo que permiten al usuario indicar cuáles son los valores de los parámetros requeridos por las operaciones, también pueden ser generados dinámicamente por AIBench a partir de las anotaciones definidas en la implementación de la correspondiente operación. Como se comentaba anteriormente, los datos solicitados por las operaciones pueden ser de tipo simple o complejo. En el caso de tipos no simples, el usuario podrá seleccionar la instancia deseada de entre aquellas del mismo tipo existentes en el clipboard de AIBench.

La Figura 5 muestra un ejemplo de un cuadro de diálogo de entrada de datos, generado de forma automática por AIBench para la suma de números enteros.

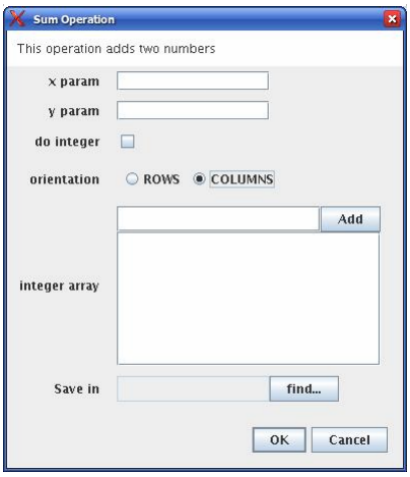


Figura 5. Cuadro de diálogo generado automáticamente por AIBench

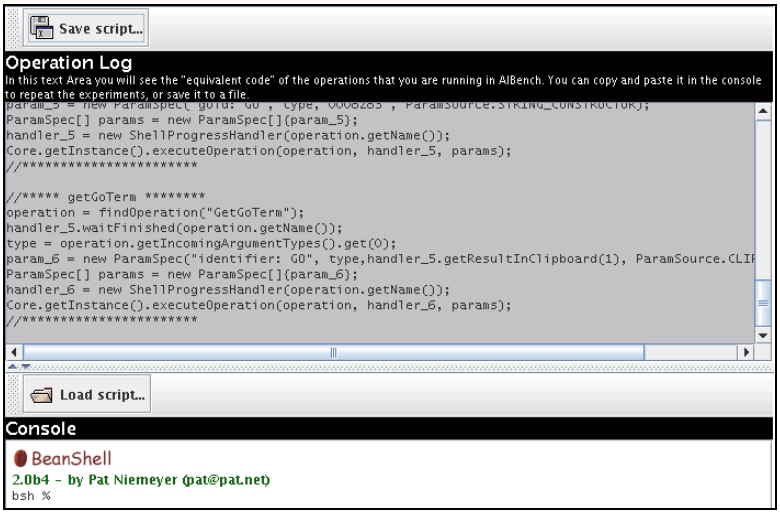


Figura 6. Shell scripting generado automáticamente para cada acción que el programador lleva a cabo en AIBench

3.2. Automatización de experimentos: *shell*

Una tarea común en la programación, prueba y validación de cualquier algoritmo, suele ser la ejecución reiterada de dicha técnica variando únicamente en cada ocasión algunos de los parámetros que determinan su comportamiento. Del mismo modo que tiene lugar la ejecución de procesos por lotes o scripts en un sistema

operativo, donde la interacción con el usuario es prácticamente nula una vez lanzada la tarea, sería de gran utilidad el poder disponer de algún mecanismo que diese soporte a la automatización de los experimentos en el marco de AIBench.

En este sentido, se ha integrado en AIBench un shell de libre distribución que permite dicha automatización [5]. Para cada una de las interacciones que tienen lugar entre el programador y AIBench, el framework genera dinámicamente el código fuente que define dicha secuencia de acciones (véase Figura 6). El *shell script* generado refleja las operaciones ejecutadas y las entradas de datos que el usuario ha realizado. Este código fuente se puede ejecutar posteriormente de nuevo a través de una consola incorporada en AIBench.

Con este servicio, el programador de AIBench puede guardar el código fuente generado a partir de un experimento inicial, cambiar ciertos parámetros en dicho código, y volver a ejecutarlo como si de un proceso por lotes se tratase.

4. Arquitectura Interna del Framework

Después de haber presentado los conceptos básicos que rodean AIBench, este apartado ofrece una visión integrada de su arquitectura modular reflejada en la Figura 7.

En la zona superior de la Figura 7 se encuentran los componentes que forman parte del núcleo (*core*) de AIBench: el clipboard, el historial de operaciones y el propio core. Además, existe un registro de las vistas implementadas que mantiene la asociación con los tipos de datos que pueden visualizar, y un *workbench* que se encarga de contener los elementos visuales que forman parte de la aplicación final desarrollada: menús, barras de herramientas, etc.

En la zona inferior de la Figura 7 se presenta, a modo de ejemplo, cómo se conectarían con el núcleo y la interfaz de usuario de AIBench un conjunto de operaciones concretas implementadas. En la Figura 7, los círculos intermedios representan las conexiones existentes entre los distintos componentes de la arquitectura. La conexión *plugins-core* es necesaria para que sea posible el registro y ejecución de las distintas operaciones implementadas. La conexión *plugins-workbench* permite que en los menús de la interfaz de usuario estén disponibles todas las operaciones implementadas.

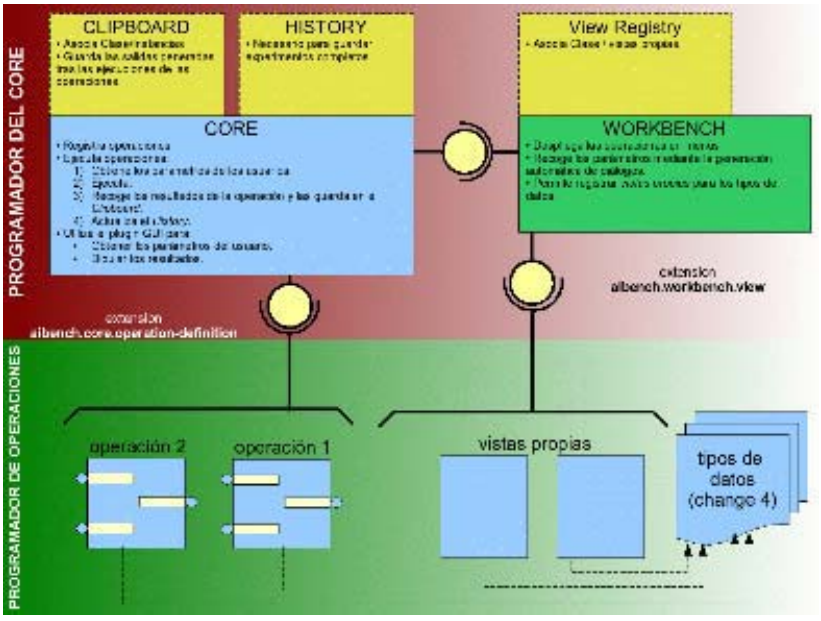


Figura 7. Arquitectura modular y extensible de AIBench

4.1. Comportamiento plug & play

Con anterioridad se ha comentado el hecho de que la arquitectura basada en plugins de AIBench permite la carga dinámica de nuevas operaciones implementadas. Esta funcionalidad se consigue empaquetando todos los componentes requeridos por cada plugin (ficheros binarios, iconos, etc.) en un fichero .JAR, junto con un descriptor XML que define unívocamente las características del plugin. Cada vez que se inicia AIBench, el framework lleva a cabo una búsqueda de todos los plugins disponibles, que se encuentran en un directorio concreto del sistema de ficheros. Para incorporar un nuevo plugin al sistema, únicamente se deberá codificar, crear su descriptor XML, empaquetarlo en un fichero .JAR y ponerlo accesible en el directorio de plugins de AIBench.

5. Conclusiones y Trabajo Futuro

En este artículo se propone AIBench como framework de programación para la enseñanza e interconexión de técnicas de IA en tercer ciclo, así como su posterior utilización por parte de los alumnos cuando son incorporadas a los grupos de investigación de los departamentos.

El objetivo final es facilitar al alumno la realización de las tareas asignadas en cada una de las materias del programa de doctorado. Su implantación real se está llevando a cabo en el primer curso del bienio de doctorado 2006-08 del programa de tercer ciclo TADSI perteneciente al Departamento de Informática de la Universidad de Vigo. En concreto, se ha planificado su utilización en las materias de "representación y análisis del conocimiento", "métodos de clasificación" y "sistemas de predicción".

A la vista de los resultados satisfactorios obtenidos dentro del grupo de investigación SING y de los proyectos asociados en los que está siendo utilizado, se espera un elevado nivel de satisfacción entre alumnos y docentes tras su implantación inicial en el programa en curso. A este respecto, actualmente están disponibles en el portal de AIBench los siguientes contenidos:

- La primera distribución pública del framework: *AIBench SDK 1.9.1*.
- Distintos plugins ya programados relacionados con el área de la bioinformática y la minería de datos.
- Un manual en línea para el programador de plugins, al que se remite al lector para profundizar en cuestiones concretas que caen fuera del alcance de este artículo.
- Un foro de discusión para que los usuarios puedan discutir sobre cualquier tema relacionado y resolver sus dudas.

Además, a través del portal de AIBench se anima a otros grupos de investigación a que envíen sus propios plugins a través del correo electrónico de contacto *contact@aibench.org*.

En cuanto al trabajo que se está llevando a cabo actualmente, destacar la creación de una nueva extensión del núcleo consistente en la codificación de un diseñador visual de experimentos. Se trata de un servicio íntimamente relacionado con la automatización de experimentos (apartado 3.2), pero que va un paso más allá. El objetivo es poner a disposición del programador un entorno en el que de modo gráfico (*drag & drop*), pueda diseñar la ejecución de sus operaciones. El diseño de un experimento consistirá en definir la secuencia de operaciones a ejecutar sobre un conjunto de datos dado, ofreciendo al usuario la posibilidad de guardar el experimento actual, recuperar los experimentos previamente almacenados, modificarlos, etc.

Agradecimientos

Los autores de este trabajo quieren agradecer y dejar constancia de la colaboración prestada por los proyectos asociados a AIBench. En concreto, al Departamento de Informática de la Universidad do Minho (Portugal), al Departamento de Informática de la Universidad de Coimbra (Portugal) y a la Fundación Biomédica del Complejo Hospitalario Universitario de Vigo (FICHUVI), sin cuya colaboración no sería posible la continua evolución y mejora de AIBench.

Referencias

[1] TADSI: *Tecnologías Avanzadas para el Desarrollo de Software Inteligente*. (2006). http://dellgwai.ei.uvigo.es/~postgrado/2005-07/Spanish/uno.html

[2] SING: *Sistemas Informáticos de Nueva Generación*. (2007). http://sing.ei.uvigo.es/

[3] AIBench: *the Artificial Intelligence workbench*. (2006). http://www.aibench.org/

[4] Platonos. (2004). http://platonos.sourceforge.net/

[5] BeanShell: *Lightweight Scripting for Java*. (2007). http://www.beanshell.org/