Universidad Central "Marta Abreu" de Las Villas

Facultad de Matemática, Física y Computación

TRABAJO DE DIPLOMA

Herramientas de apoyo al Sistema Inteligente de Ayuda al Diseño

Autor

Alejandro Sánchez Gutiérrez

Tutor

Dr. Daniel Gálvez Lio

“Año 58 de la Revolución”

Santa Clara

Curso 2016-2017

Declaración Jurada

El que suscribe, **Alejandro Sánchez Gutiérrez**, hago constar que el trabajo titulado "**Herramientas de Apoyo al Sistema Inteligente de Ayuda al Diseño**" fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de **Licenciatura en Ciencia de la Computación**, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____
Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____
Firma del Tutor(es) Firma del Jefe del Laboratorio

27 de septiembre de 2016

Resumen

El presente trabajo se centra en el desarrollo de un paquete de herramientas complementarias para el Sistema Inteligente de Ayuda al Diseño (SIAD) en su versión multiplataforma y en la realización de una propuesta para la integración de estas a manera de plugins al SIAD. Ante la variedad de sistemas operativos utilizados por los usuarios actualmente y siguiendo un patrón de compatibilidad con el SIAD, se decidió desarrollarlas, de forma que dieran como resultado un producto multiplataforma, con el fin de que el usuario se desprenda de las restricciones que impone el uso de un sistema operativo específico. Como resultado final de las herramientas implementadas, se cuenta con el GenSIAD App Creator, capaz de crear un ejecutable (APPSIAD) para una Base de Conocimiento (BC) específica que usa el Método de Solución de Problemas (MSP) del SIAD y presenta total autonomía del SIAD como aplicación. Además, se implementó el SIAD New Syntax Converter, capaz de transformar una BC escrita en la sintaxis del SIAD para MS-DOS a la nueva versión usada por el SIAD multiplataforma.

Abstract

The present work it is centered in the development of a complementary package tools for the Intelligent System of Help to the Design (SIAD) in its multi-platform version and in the realization of a proposal for the integration of these in a plugins way to the SIAD. Before the variety of operating systems used by the users at present and following a compatibility pattern with the SIAD, it was decided to develop them, so they gave as a result a multi-platform product, with the aim of that the user gets rid of the restrictions that the use of a specific operating system imposes. As a final result of the implemented tools, one counts with the GenSIAD App Creator, capable to create one executable (APPSIAD) for a (BC) specific Knowledge Base that uses Solution Method of Problems (MSP) of the SIAD and presents total autonomy of the SIAD like application. Furthermore, took effect the SIAD New Syntax Converter, capable to turn a BC written into the syntax of the SIAD for MS-DOS to the new version used by the multi-platform SIAD.

Contenido

Introducción

En los últimos tiempos han tomado un gran auge a nivel internacional las Técnicas de la Inteligencia Artificial, siendo muy comunes hoy en día los términos de “Sistemas basados en el conocimiento”, “Redes Neuronales”, etc. Dentro de los sistemas basados en el conocimiento los más comunes son los “Sistemas basados en reglas”, “Sistemas basados en Casos” y “Sistemas Híbridos”. Estas técnicas han sido aplicadas con éxito en diferentes ramas del saber. Los sistemas basados en reglas (SBR) parten de determinadas expresiones matemáticas y restricciones que caracterizan el objeto que se está proyectando. Los SBC parten de problemas resueltos en un dominio de aplicación y mediante un proceso de adaptación, encuentran la solución a un nuevo problema (Ferreiro et al., 2007).

A nivel internacional, un caso típico que puede ser esperado de la interacción con un sistema experto es ejemplificado con una ilustración de CONPHYDE (CONsulant for PHYsical property DEcisions), que es un prototipo de sistema experto cuya área de experticia es la selección del modelo termodinámico más apropiado para la evaluación de propiedades físicas en el cálculo del equilibrio vapor-líquido (Banares-Alcantara et al., 1984).

Como parte de las investigaciones acerca de los sistemas de Diseño Asistido por Computadora (CAD) en la Universidad Central “Marta Abreu” de las Villas (UCLV), y de la vinculación con el seminario de Inteligencia Artificial (IA) de la Facultad de Matemática, Física y Computación, comienzan los primeros trabajos en el país para aplicar técnicas de IA en el desarrollo de sistemas CAD a finales de la década de 1980. En tal sentido, se realizó una investigación para definir y desarrollar herramientas que apoyaran la creación de sistemas CAD basados en el conocimiento; una de estas herramientas desarrolladas fue el SIAD para el sistema operativo MS-DOS (Gálvez et al., 1993).

La versión del SIAD más difundida y utilizada ha sido la versión 2.50, la que estaba integrada por varias herramientas que formaban un paquete integrado para la construcción de sistemas CAD inteligentes: SIAD, ICASIAD y GENSIAD. En su conjunto se encargaban de todo el proceso: extracción del conocimiento de los

expertos, puesta a punto de la Base de Conocimiento (BC) y generación del sistema experto como un ejecutable para el usuario final.

Con esta versión se han logrado satisfacer en gran medida, durante muchos años las necesidades de los diseñadores, pero el avance tecnológico demandó nuevas versiones de este software en sistemas operativos más avanzados como el sistema operativo Windows.

En el año 2001 se desarrolló una nueva versión del SIAD para el sistema operativo Windows (González and Hernández, 2001a). Su implementación dependió de un archivo DLL pre compilado (del que no se disponen los archivos fuentes para recompilarlo) que limita la ejecución del SIAD en los sistemas operativos de 64 bits, instalados en la mayoría de las PCs actuales en los laboratorios docentes. Además, en este año no se intentó reprogramar ninguna de las restantes aplicaciones que integraban el paquete.

En el 2013 se crea una versión del ICASIAD para Windows (Ramos, 2013), su interfaz visual permite que un diseñador experimentado interactúe con esta mediante un diálogo acorde a los términos que conoce dicho usuario y le extrae el conocimiento necesario para construir una BC. Una vez construido el conocimiento almacenado puede ser fácilmente modificado. Pero entre los expertos pudieran surgir algunas dificultades para estructurar su conocimiento al nivel inicial que este software necesita.

En el año 2015 se desarrolla una nueva versión del SIAD (González, 2015) que le da un enfoque de lenguaje de dominio específico (***DSL*** del Inglés: ***Domain Specific Language***) y una implementación multiplataforma, la que incluye una sintaxis mejorada para la BC.

La reprogramación del GENSIAD del sistema operativo MS-DOS a Windows nunca fue intentada, por lo que en estos momentos no es posible obtener un ejecutable del Sistema de Basado en el Conocimiento (SBC) creado con el SIAD, esto provoca que no se pueda entregar al usuario final “un ejecutable”. La implementación de esta aplicación está muy asociada a características de MS-DOS y una versión para Windows debe ser totalmente rediseñada.

En este contexto se define como **objetivo general:**

Implementar herramientas para la nueva versión del SIAD que sirvan de apoyo a los actores involucrados en la creación de sistemas CAD inteligentes.

Objetivos específicos:

1. Analizar los principales referentes teóricos relacionados con el uso de la IA para el desarrollo de los sistemas CAD basados en el conocimiento y las tecnologías utilizadas en este ámbito.
2. Desarrollar el diseño e implementación de herramientas para la conversión de las bases de conocimientos a la nueva sintaxis de estas en el SIAD; así como para la generación de una aplicación para el cliente final de un sistema CAD basado en el conocimiento.
3. Proponer los mecanismos para incorporar las facilidades de las herramientas desarrolladas a la versión actual del SIAD.

Preguntas de investigación

¿Cómo implementar una aplicación capaz de convertir bases de conocimientos con una sintaxis en bases de conocimientos equivalentes para la nueva sintaxis del SIAD?
¿Qué facilidades del Sistema Operativo Windows y/o del Lenguaje Java pueden ser utilizados en la implementación de un generador de una aplicación para el cliente final de un sistema CAD basado en el conocimiento?
¿Cuáles mecanismos se pueden emplear para posibilitar la incorporación de las nuevas facilidades en la versión actual del SIAD?

Justificación

La integración de las aplicaciones (GenSIAD App Creator, APPSIAD y SIAD New Syntax Converter) han sido herramientas creadas con el propósito de obtener un ejecutable, usando el MSP del SIAD y una BC, además de permitir el cambio de sintaxis, a la nueva estructura de la sintaxis usada por el SIAD multiplataforma. Estas aplicaciones como parte del nuevo paquete de herramientas que componen los sistemas CAD son utilizadas por ingenieros mecánicos, y para la docencia de la asignatura de IA de la carrera de Ciencia de la Computación, por lo que lograr una versión estable de ellas, con las características apropiadas de las arquitecturas actuales de las computadoras resulta hoy de gran interés. Desde el punto de vista práctico se contará con herramientas libres de errores y con multiplataforma, con las cuales se podrían crear aplicaciones específicas. Ello permitirá no sólo satisfacer las necesidades de los usuarios; sino también elevar el valor del uso de esta herramienta con respecto a versiones anteriores.

Viabilidad

La creación de las aplicaciones (GenSIAD App Creator, APPSIAD y SIAD New Syntax Converter), posibilita mejorar una versión anterior; así como buscar compatibilidad con los sistemas operativos modernos, para mejorar funcionalidades. Teniendo en cuenta que los usuarios finales del sistema mantienen su interés por este y están en condiciones de aportar desde su perspectiva ingenieril al desarrollo de la nueva versión. Se cuenta con nuevas herramientas que pueden facilitar el desarrollo de este tipo de sistema, además de la experticia de los desarrolladores iniciales.

La tesis está estructurada en: Introducción, tres capítulos y la bibliografía consultada. El contenido de los capítulos puede resumirse de la forma siguiente:

Capítulo 1: Consideraciones teóricas sobre los sistemas CAD basados en el conocimiento y tecnologías utilizadas para la implementación del paquete integrado

En este capítulo, se hace una breve reseña de los sistemas CAD y los elementos de IA, específicamente los referentes a los SBC, se recogen los aspectos generales de las versiones anteriores del ICASIAD, SIAD, GENSIAD, así como las tecnologías utilizadas para la implementación de la nueva versión de las herramientas de apoyo al SIAD.

Capítulo 2: Diseño e implementación de herramientas para la nueva versión del SIAD

En este capítulo, son explicadas las tres aplicaciones que componen el nuevo paquete de herramientas del SIAD, son analizadas desde el diseño e implementación, usando la ingeniería de software pertinente; hasta el proceso interno que ocurre en cada una cuando son utilizadas.

Capítulo 3: Utilización de las herramientas complementarias para el SIAD y su extensión usando plugins

En este capítulo, son expuestos los requisitos de instalación de las herramientas desarrolladas, el uso de cada una; así como la importancia y ventajas de dicho uso; además se aborda el tema de plugins en Java con una perspectiva específica, sobre cómo pudieran integrarse dichas herramientas en futuras versiones al SIAD multiplataforma.

Capítulo 1. Consideraciones teóricas sobre los sistemas CAD basados en el conocimiento y tecnologías utilizadas para la implementación del paquete integrado

En el presente capítulo se analizan los elementos fundamentales de IA que sustentan la concepción con que se desarrolló el paquete integrado para la creación de Sistemas CAD basados en el conocimiento. Se describen componentes de la implementación de algunas de las herramientas que forman parte de este paquete y que son objeto de nueva programación en este trabajo, así como las tecnologías que fueron utilizadas en ese proceso.

1.1. Elementos de IA en el Sistema Inteligente de Ayuda al Diseño

Para poder resolver los complejos problemas a los cuales se enfrenta hoy la IA, se necesita disponer no sólo de métodos de solución apropiados, sino también de gran cantidad de conocimiento. Partiendo del criterio existente sobre el papel principal del conocimiento en la solución de problemas, es necesario considerar cuidadosamente cómo realizar la edición de este. Para ello se ha creado un rol que se encarga del proceso de estructurar el conocimiento sobre un dominio de aplicación de modo que se logre una captación correcta del conocimiento necesario en función de solucionar el problema, este rol se conoce como ingeniería del conocimiento para los sistemas basados en el conocimiento. En este capítulo se analizan elementos de la ingeniería de conocimiento y de la automatización de dicho proceso, según el formalismo de representación del conocimiento escogido.

1.1.1 Sistemas Basados en el Conocimiento

Un SBC, según Matilde Inés Césari (Inés, 2012b), es aquel que tiene como finalidad la resolución de problemas del dominio para el que ha sido creado, a partir de la aplicación de técnicas de razonamiento sobre el conocimiento que alberga su BC para ese dominio especifico.

En (Pino et al., 2001) un Sistema Experto (SE) se puede definir como aquel programa de ordenador que contiene la erudición de un especialista humano versado en un determinado campo de aplicación. En este sentido, los expertos escasean y su contratación supone una gran inversión económica, por lo que se intenta construir un sistema de manera que los conocimientos del experto se

representen de forma tal que el ordenador pueda procesar. Esto constituye un modelo computarizado de las capacidades de razonamiento y habilidades en resolución de problemas del especialista humano.

Un sistema experto pretende hacer emular la actividad de los expertos humanos, por lo que debería ser capaz, en principio, de (Hayes et al., 1983):

- Resolver el problema que se plantea de la misma manera que el experto humano.
- Trabajar con datos incompletos o información insegura (como hace el experto humano en ocasiones).
- Explicar el resultado obtenido.
- Aprender conocimientos nuevos sobre la marcha.
- Reestructurar los conocimientos de que dispone en función de datos nuevos.
- Saltarse las normas, cuando se llega a la conclusión de que éstas no son aplicables a nuestro caso concreto.

La característica fundamental de un sistema basado en el conocimiento (Edwards, 1991), es que separa los conocimientos almacenados en la BC del programa que los controla (motor de inferencia) y los datos propios de un determinado problema se almacenan en una base de datos aparte (base de hechos).

1.1.2. Arquitectura de un SBC

Un SBC puede describirse en términos de (Friendland, 1985):

- la BC
- el método de solución de problemas (MSP)
- la interfaz de usuario (IU).

De los tres componentes anteriores, los dos primeros son los más importantes y el tercero dependerá fundamentalmente del usuario al que esté dirigida dicha interfaz, desarrollada para el SBC.

En ocasiones suele describirse esta arquitectura considerando cuatro componentes básicos: la BC, la base de hechos, el motor de inferencia y la interfaz de usuario. Además, pueden aparecer otros como el módulo de cálculo de la certeza con que se obtienen las soluciones, el módulo de explicación a las soluciones obtenidas y

módulo de autoaprendizaje. Estos cuatro componentes aparecen bien descritos en (Russell and Norving, 2003).

El problema de los SBC es la creación de la BC, pues ella representa el conocimiento que posee con algunos de los formalismos, y se construye a partir de diferentes fuentes de información, pero principalmente los expertos son los que aportan mayor conocimiento y es necesario "extraerles" el conocimiento que poseen y transformarlo en el formalismo de la BC para que posteriormente esta pueda ser utilizada eficientemente en el proceso de inferencia o de búsqueda de soluciones. Este proceso de extracción puede ser complejo y ha designado un tipo de "actor" que tiene como responsabilidad llevarlo a cabo, este actor se conoce como ingeniero del conocimiento.

1.1.3. Ingeniería del conocimiento

La ingeniería del conocimiento (Alvares, 1994) se describe como la actividad de construir la BC y los procedimientos necesarios para manejarla. Es el subcampo de la IA concerniente a la adquisición, representación y aplicación del conocimiento, o como la disciplina de Ingeniería por la cual el conocimiento se integra dentro de un sistema de computador para resolver problemas complejos que normalmente requieren un alto nivel de experiencia humana.

La Ingeniería del conocimiento (Hayes et al., 1983) se define como el conjunto de principios, métodos y herramientas que permiten aplicar el saber científico y de experiencia, a la utilización de los conocimientos y de sus fuentes, mediante construcciones útiles para el hombre. Enfrenta el problema de construir sistemas computacionales con destreza, aspirando primeramente a adquirir los conocimientos de distintas fuentes y, en particular, a concluir los conocimientos de los expertos, y luego organizarlos en una implementación efectiva.

El ingeniero de conocimientos y el experto intervienen en la creación de un sistema basado en el conocimiento. Ellos pueden o no ser la misma persona, por lo que no existe un criterio único al respecto. En las primeras ocasiones en que el experto se enfrenta a la creación de un sistema basado en el conocimiento, es necesario que sea guiado u orientado por un ingeniero de conocimiento, de modo que se pueda familiarizar con esta nueva forma de pensar. Un ingeniero del conocimiento realiza

todo lo que sea necesario para garantizar el éxito de un proyecto de desarrollo de un sistema experto:

- La adquisición del conocimiento.
- La representación del conocimiento.
- La construcción de prototipos.
- La construcción del sistema.

La adquisición del conocimiento es la transferencia y transformación de la experiencia en la solución de un problema desde varias fuentes a un programa.

Según Maikel León Espinosa (León, 2007) los modos de adquisición de conocimiento son:

experto → ingeniero del conocimiento → BC.

experto → programa editor inteligente → BC.

libros → programa de inducción → BC.

libros → procesamiento de textos → BC.

En el primero, el experto interactúa con el ingeniero del conocimiento para construir la BC. En el segundo, el experto puede interactuar más directamente con el sistema experto a través de un programa editor inteligente, capacitado con diálogos sofisticados y un conocimiento acerca de la estructura de las bases de conocimiento. En el tercero, la BC puede ser construida parcialmente por un programa de inducción a partir de casos descritos en libros y experiencias pasadas. En el cuarto, un método de adquisición del conocimiento más avanzado es el aprendizaje directo desde libros.

El proceso relacionado con la ingeniería del conocimiento, pudiera automatizarse a través de un Ingeniero del Conocimiento Automatizado (ICA) que es un programa que automatiza el proceso de adquirir, formalizar, codificar y estructurar en el ordenador el conocimiento de las personas especializadas de amplia y demostrada experiencia sobre un tema para que pueda ser procesado eficientemente por un sistema de BC. Un ICA tiene como actividad liberar al usuario del formalismo de representación que se emplee para estructurar la BC que usa el sistema basado en

el conocimiento para resolver problemas complejos que normalmente requieren un alto nivel de experiencia humana.

Las etapas en el desarrollo de un sistema basado en el conocimiento provocan que la automatización de la ingeniería de conocimiento no pueda verse como un producto general válido para cualquier FRC, de ahí que aparezcan diversos ingenieros de conocimientos automatizados tomando como base un formalismo y un dominio o área específicos. Ejemplos desarrollados en tal sentido son ICA, que maneja mapas cognitivos difusos (León, 2007) , e ICASIAD (González and Hernández, 2001b).

1.1.4 La Base de Conocimiento

Una BC (Inés, 2012a) es una estructura de datos que contiene una gran cantidad de información sobre un tema específico, generalmente introducida por un experto en dicho tema, sobre la cual se desarrolla la aplicación. Este conocimiento pudiera estar constituido por la descripción de los objetos a tener en cuenta y sus relaciones, así como por casos particulares o excepciones y diferentes estrategias de resolución con sus condiciones de aplicación.

Se habla de BC como el componente donde se almacena el conocimiento de un campo específico o dominio, la cual contiene el conocimiento del tema que se desea incorporar al SBC, generalmente proporcionado por un experto u otras fuentes de conocimientos, convenientemente formalizado y estructurado. Es el pilar central de un SBC, ya que es el depósito donde se almacena el conocimiento del que este dispone sobre un dominio específico.

En la forma de representar el conocimiento en una BC hay que tener en cuenta tres niveles (Alvares, 1994):

Cognitivo: ¿Cómo alcanzan y procesan el saber los hombres?

Representación: ¿Cómo se formaliza el conocimiento?

Implementación: ¿Cómo se procesa por un ordenador?

Si se analizan los tres niveles anteriores se puede apreciar que los dos primeros niveles están en función del tercero, es decir, están condicionados y enfocados a su posible manipulación por ordenadores.

1.1.5. Formas de representación del conocimiento

La cuestión básica de la representación del conocimiento es el desarrollo de una notación suficientemente y precisa con la cual representar este. A esa notación se le llama Forma de Representación del Conocimiento (FRC) (Brachman and Levesque, 2004).

No existe actualmente una FRC general, capaz de ser usada en todo tipo de aplicación con éxito; las formas disponibles están limitadas a más o menos un dominio específico. Ante una aplicación y la oferta de las FRC existentes, es necesario realizar la selección de la más adecuada para lo cual se pueden considerar los criterios siguientes (Bello et al., 2003):

- Debe permitir describir el dominio de una manera natural, reflejando, tanto como sea posible, la estructura de los objetos, los hechos y las relaciones entre ellos.
- Debe aceptar conocimiento empírico, teórico o heurístico, y combinar el conocimiento declarativo con el procedimental, de acuerdo con los requerimientos de la aplicación.
- Debe permitir estratificar el conocimiento de acuerdo con su significado y funciones.
- Debe permitir que el conocimiento de los expertos se represente fácilmente.
- Los expertos en la aplicación deben poder comprender el conocimiento que está almacenado.
- El conocimiento almacenado debe poder usarse con efectividad.

Hay muchas maneras posibles de representar el conocimiento de un dominio. Para manejo del conocimiento simbólico se tienen entre otras (Edwards, 1991):

1. Reglas de producción.
2. Marcos (*frames*).
3. Redes semánticas.
4. Objetos.

Los esquemas de representación del conocimiento han sido generalmente clasificados como declarativos y procedimentales. Un esquema declarativo se usa

para representar hechos y aserciones; mientras que la representación procedimental especifica un procedimiento para resolver los problemas, incluye procedimientos o algoritmos. Los métodos para la representación del conocimiento declarativo abarcan: lógica, reglas de producción, redes semánticas, marcos y *scripts* (Bello et al., 2003).

En lo adelante se profundiza en las reglas de producción y en los marcos ya que estos son las FRC que se combinan para representar el conocimiento en los SBC construidos con el SIAD.

1.1.5.1. Reglas de producción

Una regla de producción (Brachman and Levesque, 2004) no es más que un par (**A**, **B**) que puede representarse en el cálculo proposicional como **A** → **B**. Estas reglas de producción expresan siempre una condicional, con un antecedente y un consecuente. La interpretación de una regla es que, si el antecedente se satisface, entonces se obtiene el consecuente. La regla de producción representa una unidad relativamente independiente de conocimiento.

Las reglas de producción pueden utilizar un formato **IF- THEN** para representar el conocimiento, la parte **IF** de una regla es una condición (también llamada premisa o antecedente), y la parte **THEN** de la regla (también llamada acción, conclusión o consecuente) permite inferir un conjunto de hechos nuevos si se verifican las condiciones establecidas en la parte **IF**.

Al usar las reglas de producción como FRC se pueden desarrollar sistemas con las propiedades siguientes (Bello et al., 2003):

Modularidad**:** cada regla define una pequeña, pero relativamente independiente, unidad de conocimiento.

Incrementabilidad**:** se pueden añadir nuevas reglas con una relativa independencia del resto.

Modificabilidad**:** las reglas viejas pueden ser modificadas con relativa independencia.

Habilidad**:** la habilidad del sistema crece proporcionalmente a la cantidad de reglas.

Transparencia**:** pueden explicar el camino que llevó a la solución.

Refinamiento**:** el conocimiento existente puede ser refinado.

1.1.5.2. Marcos

Un marco es un concepto originalmente propuesto por Marvin Minsky en 1975 (Minsky, 1975) como una FRC, que es capaz de representar los atributos de un objeto o concepto de una manera más rica, de lo que es posible, usando reglas. Típicamente incluye varios aspectos, cada uno de los cuales pueden contener un valor o se puede dejar en blanco. El número y el tipo de aspectos se elegirán de acuerdo con el conocimiento particular que está siendo representado.

Un marco (Edwards, 1991) es una estructura de datos compleja que contiene un agregado de información acerca de un objeto. La información almacenada se distribuye en diferentes campos llamados aspectos, cada aspecto contiene la información sobre un atributo del objeto que se modela. A cada aspecto se le puede asociar varios tipos de información, llamados facetas del aspecto. Se puede plantear que la información almacenada tiene cuatro niveles de detalle: el marco, los aspectos del marco, las facetas de cada aspecto y los valores almacenados en cada faceta. La representación debe tener dos capacidades: expresividad adecuada y una eficiencia de razonamiento. La expresividad y el razonamiento le confieren la capacidad adecuada para ser considerado como una alternativa útil para la representación del conocimiento.

El formalismo de marcos (Inés, 2012b) se ha revelado como la técnica de representación del conocimiento más utilizada en la IA, cuando el conocimiento del dominio está organizado en base de conceptos. Minsky definió los marcos como una estructura de datos que representa situaciones estereotipadas, construidas sobre situaciones similares ocurridas anteriormente; permitiendo así aplicar a situaciones nuevas el conocimiento de situaciones, eventos y conceptos previos.

El conocimiento que se expresa en la estructura de datos es el conocimiento declarativo del dominio. Dentro del marco existe conocimiento procedimental que se refiere a cómo utilizar el marco que se espera que suceda a continuación, así como el conjunto de acciones que se deben realizar, tanto si las expectativas se cumplen, como si estas fallan. Los marcos organizan el conocimiento del dominio en árboles, también llamados jerarquías, o en grafos, ambos construidos por especialización de conceptos generales en conceptos más específicos.

1.1.5.3. Representación del Conocimiento en el SIAD

Según (Bello et al., 1994) en el formalismo utilizado en el Sistema Inteligente de Ayuda al Diseño (SIAD) el modelo físico del producto se describe por medio de dos tipos de marcos. El primero se llama marco del Objeto de Diseño, el cual permite almacenar el conocimiento acerca de un producto de una manera general. El segundo tipo de marco se llama marco de Parámetro de Diseño, y se utiliza para representar el conocimiento en relación con cada parámetro del producto.

Por lo tanto, una BC para el SIAD que almacena el modelo físico de un determinado producto, consistirá de un marco Objeto de Diseño y tantos marcos Parámetro de Diseño como se necesiten en dependencia de los requerimientos del producto.

Un marco de objeto de diseño está formado por los siguientes aspectos (González and Hernández, 2001b):

- Nombre del producto.
- Descripción del producto.
- Pre acciones del diseño.
- Parámetros de diseño importados.
- Restricciones de diseño.
- Funciones para evaluar la solución.
- Parámetros para generar información para AutoLisp.
- Post acciones del diseño.

Cada marco de parámetro de diseño está formado por los siguientes aspectos:

- Nombre del parámetro.
- Descripción del parámetro.
- Dominio del parámetro.
- Pre acciones de evaluación.
- Parámetros antecesores.
- Procedimientos para el cálculo del valor del parámetro.
- Media acciones de evaluación
- Procedimientos para el cálculo del peso del parámetro.
- Post acciones de evaluación.

1.1.6. Métodos de Solución de Problemas

En los métodos de búsqueda de soluciones (Russell and Norving, 2003) es posible la estrategia de búsqueda primeramente a lo ancho (***BFS*** del inglés: ***Breath First Search***), (Russell and Norving, 2003) la cual se basa en expandir el nodo raíz primero, luego se expanden todos los sucesores del nodo raíz y así sucesivamente. En general todos los nodos en un mismo nivel de profundidad son expandidos, antes de expandir los nodos del nivel siguiente.

Por otra parte, está la búsqueda en profundidad (***DFS*** del inglés: ***Depth First Search***), (Russell and Norving, 2003) que funciona siempre expandiendo el nodo más profundo de la frontera actual en el árbol de búsqueda. Así, la búsqueda procede inmediatamente al nivel más profundo del árbol de búsqueda, donde los nodos no tienen sucesores. A medida que estos nodos son expandidos, son sacados de la frontera, entonces la búsqueda regresa al siguiente nodo más profundo que aún contiene sucesores sin explorar.

Estas dos estrategias de búsqueda se pueden realizar de tres formas, hacia adelante (del inglés: ***forward***), (Russell and Norving, 2003) desde el estado inicial hasta el estado objetivo; hacia atrás (del inglés: ***backward***), (Russell and Norving, 2003) desde el estado objetivo hasta el estado inicial; o ambas al mismo tiempo o sea bidireccional, una hacia adelante desde el estado inicial y la otra hacia atrás desde el estado objetivo, el algoritmo para cuando las dos búsquedas se encuentran en el medio. La idea es reducir la complejidad temporal de cualquiera de estas dos direcciones de búsqueda por sí solas, pero no siempre puede ser aplicable.

1.1.6.1. Métodos de Solución de Problemas en el SIAD

En el caso del MSP usado por la nueva versión multiplataforma del SIAD (González, 2015), la estrategia de búsqueda usada es la estrategia en profundidad con dirección de búsqueda hacia adelante, este método que se aplica hace retroceso (del inglés: ***backtracking***) para recalcular soluciones, una vez que le es imposible continuar buscando soluciones con la búsqueda en profundidad.

Según el algoritmo empleado en el SIAD para la búsqueda de soluciones, inicialmente es posible que el usuario introduzca un conjunto de datos iniciales, a partir de los cuales se desencadena el proceso de búsqueda, usando una estrategia

primeramente en profundidad con encadenamiento adelante (forward). Si en algún momento de este proceso es imposible calcular el valor de un parámetro de diseño, entonces el algoritmo selecciona un parámetro de diseño de los que ya tienen valor desde donde reiniciar el proceso, este mecanismo incorporado se conoce como retroceso o *backtracking*, le da flexibilidad al algoritmo y permite explorar mejor el espacio de búsqueda de soluciones.

1.2. Creación de sistemas CAD basados en el conocimiento

La versión del SIAD más difundida y utilizada ha sido la versión 2.50. Esta versión estaba integrada por varias herramientas que formaban un paquete integrado para la construcción de sistemas CAD inteligentes: SIAD, ICASIAD y GENSIAD. En su conjunto se encargaban de todo el proceso: extracción del conocimiento de los expertos, puesta a punto de la BC y generación del sistema experto como un ejecutable para el usuario final. Sobre la concepción, implementación y estado actual de dichas herramientas trata este epígrafe.

1.2.1. Sistema Inteligente de Ayuda al Diseño (SIAD)

El SIAD 2.50 (Gálvez et al., 1993) se concibió como un entorno de desarrollo y puesta a punto de sistemas CAD inteligentes (o basados en el conocimiento). Estaba orientado principalmente al ingeniero de conocimiento de manera que tuviera facilidades para:

- la creación, edición y modificación de las BC.
- la verificación de que las BC estuvieran bien escritas en el formalismo definido.
- la prueba de la corrección de las BC a través del cálculo de las soluciones y las facilidades de trazas del proceso de ejecución.

El SIAD 2.50 fue desarrollado en lenguaje C, para MS-DOS hace ya más de 25 años, por lo que muchos de los elementos utilizados en su desarrollo no se ajustan a las nuevas concepciones. Sin embargo, en su momento garantizó una buena interfaz para el usuario; el control sobre todos los dispositivos de entrada/salida y chequeo de errores; la implementación eficiente de un compilador construido manualmente y de un ejecutor de un método de solución de problemas de IA; el manejo de múltiples soluciones; la edición de la BC, usando diferentes editores de textos con propósitos; un editor gráfico en el que visualmente se podían construir

prototipos de los diseños de las piezas calculadas desde una BC e integrarse con el proceso de inferencia; así como la incorporación del seguimiento (traza o *debug*) al proceso de inferencia y otras funcionalidades.

Se desarrollaron múltiples aplicaciones que acompañaban al SIAD, también en lenguaje C sobre MS-DOS, además de las conocidas que se analizan a continuación, se contaba con un editor de letras (FON) descritas vectorialmente como editor gráfico y un programa de ayuda de todas las herramientas del sistema.

1.2.1.1. SIAD sobre Windows

Para el sistema operativo Windows fue desarrollada una versión del SIAD en 2001 (González and Hernández, 2001a). Esta implementación sobre Borland Delphi (Object Pascal) dependió de un archivo DLL pre compilado (del que no se disponen los archivos fuentes para recompilarlo) que limita la ejecución del SIAD en los sistemas operativos de 64 bits instalados en la mayoría de las PCs actuales, incluyendo las que se encuentran en los laboratorios docentes.

En el año 2015 se desarrolla una nueva versión del SIAD (González, 2015) que le da un enfoque de lenguaje de dominio específico (***DSL*** del Inglés: ***Domain Specific Language***) (Parr, 2007) al formalismo de la BC, con este enfoque se decide modificar sintaxis para acercarla más a los usuarios finales y automatizar el proceso de creación del compilador de la BC utilizando herramientas y técnicas actuales como las que aporta ANTLR 4.

Esta implementación del SIAD se realiza utilizando el lenguaje Java en el IDE NetBeans, lo que garantiza su uso multiplataforma.

1.2.2. Ingeniero de conocimiento automatizado

Un ICA (León, 2007) es un programa que automatiza el proceso de adquirir, formalizar, codificar y estructurar en el ordenador el conocimiento de las personas especializadas de amplia y demostrada experiencia sobre un tema para que puedan ser procesadas eficientemente por un sistema de BC. Un ICA tiene como actividad liberar al usuario del formalismo de representación que se emplee para estructurar la BC que usa el sistema basado en el conocimiento para resolver problemas complejos que normalmente requieren un alto nivel de experiencia humana.

Las etapas en el desarrollo de un sistema basado en el conocimiento provocan que la automatización de la ingeniería de conocimiento no pueda verse como un producto general válido para cualquier FRC, de ahí que aparezcan diversos ingenieros de conocimientos automatizados, tomando como base un formalismo y un dominio o área específico. Ejemplos desarrollados en tal sentido son ICA que manejan mapas cognitivos difusos (León, 2007).

En particular para el SIAD fue desarrollado un ICA en 1992, que formaba parte del Paquete integrado para la creación de Sistemas CAD basados en el conocimiento. Este ICA fue programado en C para MS-DOS y en un diálogo interactivo con los expertos o ingenieros del conocimiento iba preguntando por todo el conocimiento a colocar en una BC del SIAD, luego organizaba todo el conocimiento y lo mostraba en forma de tabla desde la cual los usuarios podían actualizarlo para finalmente almacenarlo con la sintaxis de las BC de la versión 2.5 del SIAD.

1.2.2.1. ICASIAD versión sobre Windows

El ICA para el SIAD del sistema operativo Windows (González and Hernández, 2001b) se le llama ICASIAD, comienza su trabajo proponiendo al usuario un menú. En dependencia de la opción seleccionada, el sistema comenzará a trabajar en uno de sus dos modos de operación:

1. Modo asistente.
2. Modo interactivo.

El modo asistente sólo está disponible cuando se desea crear una BC, y se activa cuando el usuario opta por las opciones “ICA” o “Nuevo” del menú. Este modo guía el trabajo del usuario del sistema, haciéndole un conjunto de preguntas en un orden determinado que le permite al ICASIAD obtener la información básica necesaria para generar de forma automática la BC. La información solicitada por este modo de trabajo sigue el siguiente orden:

1. Nombre del objeto de diseño.
2. Descripción (comentario) sobre el objetivo de la BC.
3. Descripción de cada parámetro que forma parte de la BC.
4. Para cada parámetro dado se pide el código que lo identificada (nombre del parámetro).

5. Para cada parámetro dado se piden los valores normados o de uso más frecuentes.
6. Para cada parámetro dado se analiza si es o no calculable, de manera que si lo es, el sistema pasa a pedir la información para conformar los procedimientos de cálculo del parámetro.
7. Restricciones de diseño entre los parámetros.
8. Para cada parámetro se piden los predecesores, es decir, los parámetros que influyen implícitamente en el cálculo del parámetro.

Cuando el usuario le haya contestado al ICASIAD todo lo anterior, el sistema pasa al modo interactivo.

El menú presenta, además, una opción para cargar una BC desde un fichero (.bcs) creado con anterioridad, y otras opciones como “Salvar” y “Salvar como…” que como sus nombres indican le permiten al usuario salvar en memoria toda la información creada para su BC.

El modo interactivo está disponible cuando se desea modificar la información dada en el modo asistente o cuando se desea modificar la información de una BC cuyo archivo ya se encontraba almacenado en disco.

Limitaciones:

Al diseñar interfaces de usuario deben tenerse en cuenta las habilidades cognitivas y de percepción de las personas, y adaptar el programa a ellas. Esta versión del ICASIAD hace que los usuarios dependan de su propia memoria, forzándolos a recordar cosas innecesarias (por ejemplo, información que apareció en una pantalla anterior). Los usuarios demoran en aprender a usar el sistema y se hace difícil recordar sobre el uso del sistema en un período de tiempo.

- ✓ Sólo favorece el uso del *mouse* limitando el del teclado.
- ✓ Los usuarios quedan atrapados en una tarea en muchas ocasiones, lo que provoca que terminen la ejecución de la aplicación y la reinicien para poder continuar con su trabajo.
- ✓ El diseño de la interfaz visual no muestra en una sola ventana de forma gráfica todas las componentes descritas de la BC.

1.2.3. Generador para el SIAD

En el paquete integrado, el generador del SIAD recibía como entrada un archivo texto (.bcs) con la descripción de una BC y daba como resultado un ejecutable (.exe) que contenía el mecanismo de inferencia del SIAD, acoplado a la codificación interna de la BC de entrada, de tal manera que el ejecutable obtenido, sólo era capaz de cargar esa BC ya compilada, y ejecutarla para obtener soluciones. Esta aplicación resultaba muy útil para poder distribuir los diferentes sistemas basados en el conocimiento desarrollados con el SIAD de manera ejecutable.

Para comprender la implementación del GENSIAD para MS-DOS, se describe en las etapas siguientes:

1ro: Se creó la biblioteca SIADCOMP.lib con los módulos que formaban al compilador de la BC y que generaban la forma interna de la BC del SIAD.

2do: Se programó la función `guardarBCFI()` capaz de tomar la forma interna de una BC y escribirla en disco (equivalente a un proceso de serialización de la forma interna).

3ro: Se creó la biblioteca SIADEJEC.lib con los módulos que formaban al ejecutor de la forma interna de la BC del SIAD (mecanismo de inferencia y tratamiento de soluciones).

3ro: Se creó la biblioteca VISUALEJEC.lib una interfaz visual basada en la interfaz del SIAD que solo ofrecía a los usuarios la posibilidad de evaluar una BC y mostrar las soluciones obtenidas.

4to: Se programó la función `cargarBCFI()` capaz de tomar desde un archivo en disco la forma interna de una BC y armar en memoria su representación.

5to: Se combinó la interfaz visual en VISUALEJEC.lib, la biblioteca SIADEJEC.lib y la función `cargarBCFI()` en un programa ejecutable llamado EMPTYSIADRUN.exe, este programa contenía en su función principal (`main`) un algoritmo para buscar el texto "***BC***" en el propio archivo en disco del programa, e indicarle a la función `cargarBCFI()` que leyera del disco, a partir de esa marca la forma interna de BC salvada. Luego que este proceso se ejecute correctamente se invoca a las funciones de la interfaz visual en VISUALEJEC.lib, quien, en dependencia de las opciones seleccionadas por el usuario final, activa las funciones de ejecución en SIADEJEC.lib; si este

proceso de carga de la BC no se ejecuta correctamente, se crea un mensaje de error al usuario y se termina su ejecución.

6to: Se combinó la biblioteca SIADCOMP.lib y la función `guardarBCFI()` en el programa llamado GENSIAD.exe, el que contenía en su función principal (`main`) un algoritmo para obtener de la línea de comandos el nombre del archivo de entrada con la BC y el nombre del archivo de salida que contendrá el ejecutable, luego le pasaba el nombre de la BC a las funciones de SIADCOMP.lib para que se compile la BC y se obtenga la forma interna de la misma si no se presentaron errores sintácticos; de existir estos, se indican después y el programa termina. Finalmente se activaba una función que se encargaba de buscar en el propio archivo en disco GENSIAD.exe la marca "***ESIADR***" después de la cual se encontraba el código binario del programa EMPTYSIADRUN.exe, leía todo ese código y lo guardaba con el nombre del archivo de salida indicado en la línea de comandos y a continuación se activaba la función `guardarBCFI()` para que siguiera escribiendo en el archivo de salida la marca ***BC***DGLIO** y la salva de la forma interna de la BC compilada.

1.2.3.1. GENSIAD sobre Windows

La versión para Windows de esta herramienta nunca fue enfrentada anteriormente, debido, fundamentalmente, a que la solución dada en su versión para MS-DOS era muy dependiente del sistema operativo, por lo que su implementación para Windows no cumplía objetivo, siendo esta igual de dependiente si se hubiera realizado.

1.3. Tecnologías utilizadas para la creación de la nueva versión de las herramientas de apoyo al SIAD

Las tecnologías usadas en el desarrollo de las herramientas de apoyo al SIAD son parte del lenguaje de programación Java, específicamente de la versión 1.8 del JDK, por lo tanto, son todas software libre del cual se puede disponer para el desarrollo de aplicaciones.

1.3.1. Serialización de Java y sus fundamentos

El proceso de salvado de objetos a un flujo (del inglés: ***stream***) es llamado serialización (del inglés: ***serialization***)(Horstmann, 2008), porque a cada objeto le es asignado un número de serie en el flujo. Si el mismo objeto es salvado dos veces, el número de serie sólo es escrito la segunda vez. Cuando los objetos son leídos, los números de series duplicados son restaurados como referencia al objeto. Los objetos salvados a un flujo de objeto (del inglés: ***object stream***) deben pertenecer a clases que implementen la interfaz `Serializable`.

Por razones de seguridad algunos programadores no serializan clases con contenido confidencial, ya que una vez se hace serializable una clase, cualquiera puede escribir sus objetos a disco y analizar el archivo.

En el lenguaje Java, los objetos que el programador desea salvar o recuperar de un flujo deben ser objetos pertenecientes a clases que implementen la interfaz `Serializable` de Java. El lenguaje también permite que no todos los atributos de una clase serializable sean salvados o recuperados, para ellos se utiliza el modificador `transient` en la declaración del atributo.

1.3.2. Trabajo con archivos Zip

Como parte del lenguaje, Java ofrece un extenso paquete de utilidades conocido como `java.lang.util`, el cual contiene las clases del *framework* de colecciones del modelo de eventos, para las facilidades de acceso a la hora y fecha y otras clases de utilidades misceláneas (Oracle, 2015b), entre las que se encuentra el conjunto de clases del paquete `java.lang.util.zip` que ofrece clases para la lectura y escritura de archivos comprimidos con los formatos estándares ZIP y GZIP(Oracle, 2015c). En la figura 1 se muestra el paquete `java.lang.util.zip` y su relación con otros paquetes de Java.

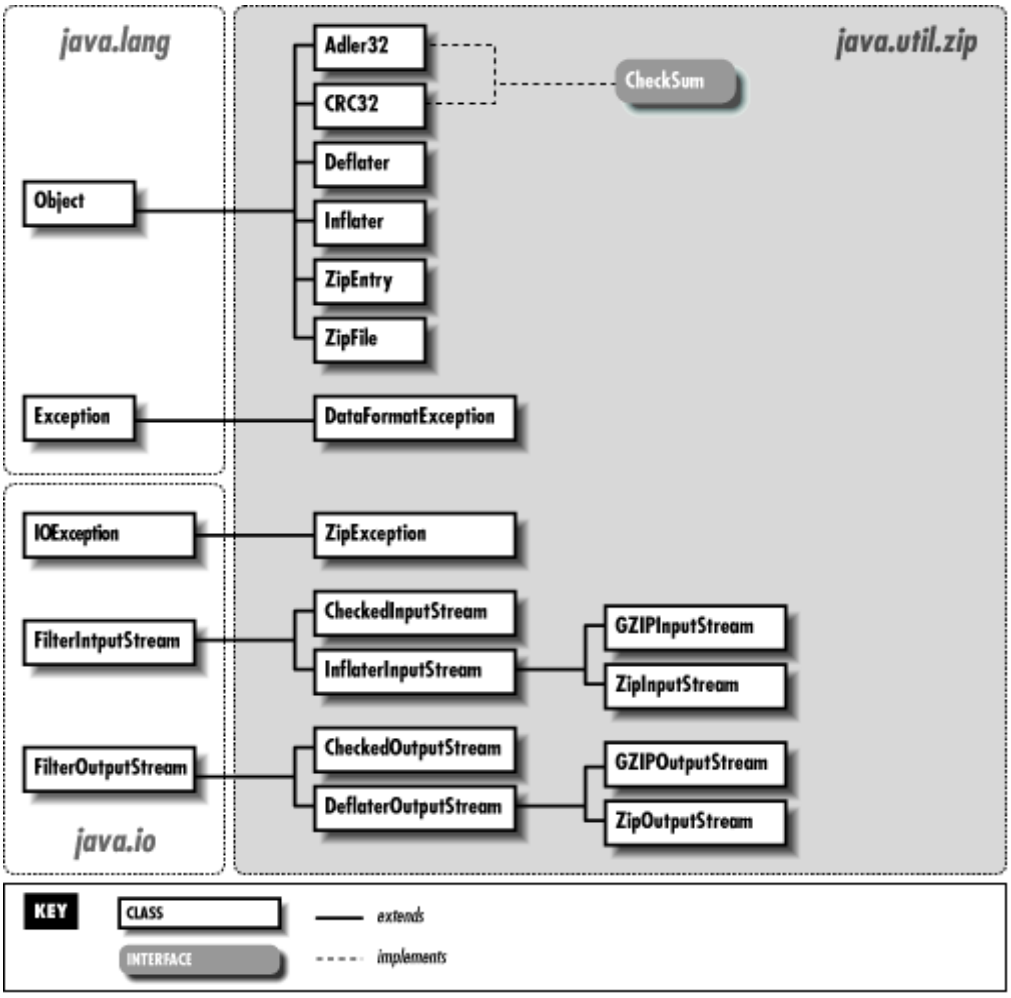


Figura 1: Jerarquía de paquetes: java.util.zip (Oracle, 2015c)

Gzip y deflater (en español: ***desinflador***) son formatos de compresión. Por su parte Zip es ambos, formato de compresión y archivo. Esto significa que un solo archivo (.zip), puede contener más de un archivo comprimido o descomprimido, junto con información acerca de nombres, permisos, fechas de creación y modificación, y otras informaciones acerca de cada archivo contenido en el fichero (.zip). Esto hace a la lectura y escritura de archivos (.zip), de alguna manera, más compleja y menos amena que un flujo de datos a leer o escribir archivos (.gzip) o desinflados (del inglés: ***deflated***) (Eliotte, 1999).

El uso de esta tecnología de Java, permite revisar en tiempo real los archivos contenidos dentro de un ejecutable de java, o sea un (.jar), ya que también pueden ser interpretados como archivos (.zip), esto facilita el acceso a los recursos

incrustados en una aplicación, permite el empaquetamiento de dichos recursos en un solo archivo y el conocimiento de la organización de la estructura física de los archivos que conforman el programa, de esta manera se puede modificar dicha información y con ello modificar también el programa de manera dinámica. Estos cambios deben realizarse con cuidado y garantizando el correcto funcionamiento del programa modificado.

En un (.jar) de Java todos los recursos contenidos en él son referidos a través de sus caminos relativos (que empiezan en la raíz del programa) y no absolutos; por lo que en el proceso de compresión/descompresión es necesario saber el camino completo donde se encuentra el programa en disco, también esto es necesario si se desea modificar dinámicamente la estructura interna del archivo (.jar).

1.3.3. Plugins en Java

Independientemente de las funciones que tenga una aplicación, existen ocasiones en que un usuario necesita resolver problemática no prevista por el desarrollador original y que no está incluida en la aplicación, o incluso peor, puede ser una problemática realmente especializada que solamente él necesite, y que nunca será agregado por el comerciante del software. Una solución para esto sería a través de los plugins (Dittmer, 2010): Trozos de código externo que son accedidos a través de la aplicación, y usan su Application Program Interface (API) para actuar sobre una función que la misma aplicación no puede.

1.3.3.1. Proceso de creación de plugins en Java

Generalmente hablando, hay cuatro pasos involucrados en el proceso de creación de un plugin en Java (Oracle, 2015a):

1. Definir una API, (ejemplo: una interfaz de Java) para implementar el plugin.
2. Determinar como la aplicación anfitriona llega a saber cuál plugin está disponible, donde encontrarlos y cuando cargarlos.
3. Escribir una clase de carga (del inglés: ***ClassLoader***) que cargue las clases del plugin.
4. Escribir un manejador de seguridad (del inglés: ***SecurityManager***) que gobierne que plugins están autorizados a hacer que.

Podríamos también dejar que los plugins jugaran un rol más activo, creando una interfaz de eventos que permita a los plugins ser llamados cuando ciertos eventos del sistema ocurren, además, los parámetros pasados a un plugin pudieran incluir estructuras de datos internos de la aplicación que permitan manipular la aplicación anfitriona en formas muy fundamentales.

1.4. Conclusiones Parciales

En el ámbito de aplicación de la IA, se evidenció el papel principal que asume el conocimiento para resolver las complejidades del mundo actual, de ahí que la construcción de sistema CAD inteligentes resultó viable, en función de la adquisición, representación y aplicación del conocimiento.

En el paquete integrado, el generador del SIAD recibía como entrada un archivo texto (.bcs) con la descripción de una BC, y daba como resultado un ejecutable (.exe) que contenía el mecanismo de inferencia del SIAD acoplado a la codificación interna de la BC de entrada, de tal manera que el ejecutable obtenido sólo era capaz de cargar esa BC ya compilada, y ejecutarla para obtener soluciones. A partir de esta idea, se desarrollará una nueva versión de esta herramienta y usará Java y varias de sus tecnologías para liberarse de cualquier atadura que tuvieran antiguas versiones, restringidas a un sistema operativo en específico.

La necesidad de poder interpretar las BC escritas en la antigua sintaxis del SIAD para MS-DOS y para Windows, dieron paso a la creación de una nueva herramienta que permitirá transformar estas BC a la nueva sintaxis usada por el SIAD multiplataforma.

Capítulo 2. Diseño e implementación de herramientas para la nueva versión del SIAD.

En este capítulo se hace un enfoque teórico de las funcionalidades y capacidades de las herramientas complementarias para el SIAD: APPSIAD, GenSIAD App Creator y SIAD New Syntax Converter. Se aborda su implementación, de forma que se puedan desentrañar con facilidad sus usos y limitaciones, y así favorecer la comprensión del trabajo de dichas herramientas.

2.1. Generador de aplicaciones del SIAD

El generador de sistemas CAD inteligentes GenSIAD App Creator, toma de entrada una BC del SIAD archivo (.bcs) y genera un programa ejecutable que la contiene. Ese programa ejecutable es capaz de activar el proceso de inferencia del SIAD para obtener múltiples soluciones, sólo para la BC que contiene. Puede considerarse como un ejecutable para esa BC, y está orientado al usuario final del sistema inteligente construido con el SIAD.

2.1.1. Diseño e Implementación del GenSIAD App Creator

El GenSIAD App Creator utiliza el patrón de diseño Modelo Vista Controlador(MVC). En el paquete `Model`,de su estructura de paquete (Figura 2) se encuentra el paquete `TemplateApp` que contiene a su vez la muestra incrustada el APPSIAD.jar sin BC. Dentro del paquete `Model/TemplateApp` se encuentra el paquete `lib` donde se ubican todas las bibliotecas de referencia necesarias para que el APPSIAD.jar funcione correctamente.

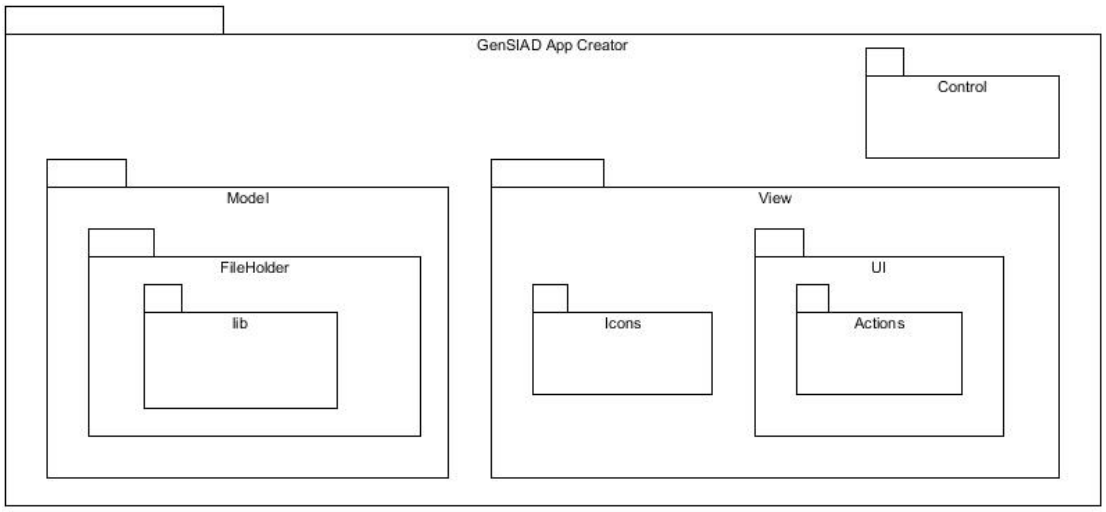


Figura 2: Estructura de paquetes del GenSIAD App Creator

El paquete `View` contiene todos los elementos de la interfaz gráfica y la clase principal (ej: iconos, paneles de diálogo de Java, etc.).

Por último, está el paquete `Control`, en el cual están las clases `Logger` y Merger, que se encargan de crear el archivo de registro que contiene la información generada por el compilador SIAD.Compiler.And.Executer y del proceso de serialización, copia e incrustación de datos respectivamente.

El engranaje principal del GenSIAD App Creator es la clase `Merger`, con sus métodos dedicados al des empaquetamiento, empaquetamiento, copia, serialización y compilación de datos, los cuales luego de usarse en el orden correcto, dan como resultado una aplicación de APPSIAD completamente funcional, en el anexo 7 se muestra la relación de sus componentes y el grado de dependencia que presenta con el MSP del SIAD y la BC que recibe como entrada. En la figura 3, son presentados los casos de uso del GenSIAD App Creator, sus capacidades consisten en: cargar una BC, nombrar la aplicación resultante y crear un ejecutable para esta aplicación resultante.

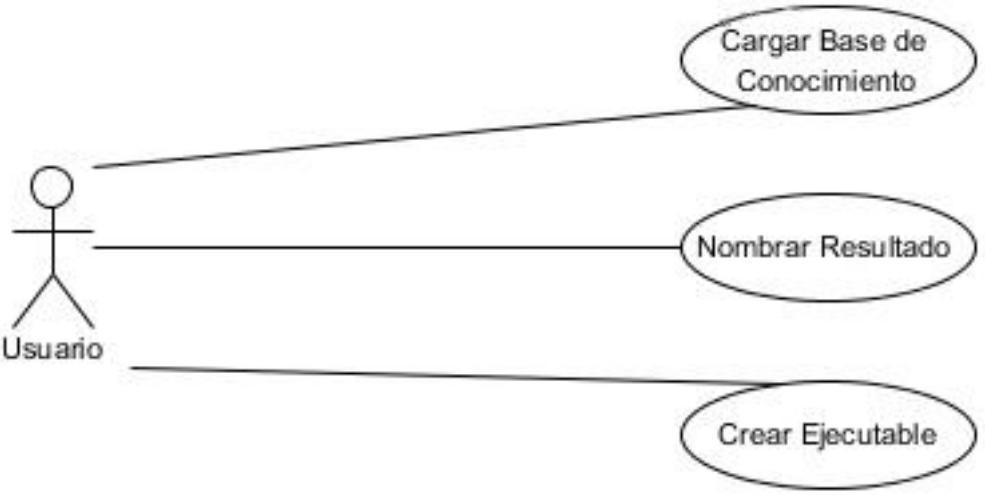


Figura 3: Casos de uso del GenSIAD App Creator

2.1.2. Proceso de compilación, serialización, empaquetado y copiado

Cuando se oprime el botón de crear en la interfaz visual, lo primero que hace el programa es crear una instancia de la clase Merger con el archivo (.bcs) como parámetro, luego se ejecuta la siguiente serie de métodos en el orden en que se muestran:

- `public void Compiler_Serializer()`: compila la BC usando el SIAD. Compiler And Executer, la serializa y copia el objeto serializado al archivo temporal data.dat, para después ser incrustado.
- `public void Zipper(String)`: compacta el archivo data.dat que contiene la BC serializada y lo coloca en `Model/FileHolder` dentro de la estructura de paquetes del APPSIAD, además extrae el APPSIAD y crea una copia de este en la cual ubica el (.dat) renombrando el archivo (.jar) con la cadena que se le pasa a este método como parámetro.
- `public void Copier()`: este método extrae todas las bibliotecas que se encuentran en `/Model/TemplateApp/lib` dentro del (.jar) de esta aplicación y las copia a una carpeta que se crea en el momento de construcción del objeto `Merger`, llamada BuildedApps.

2.1.3. Creación del LogFile

Esta característica del GenSIAD App Creator permite chequear el proceso de compilación de la BC. Usando el MSP del SIAD (González, 2015), el GenSIAD App

Creator es capaz de redireccionar sus mensajes de error en tiempo de compilación a un archivo, aportándole al usuario información útil del estado de conclusión de este proceso.

Su implementación es muy sencilla, basta con crear un archivo en la carpeta raíz del GenSIAD App Creator con el nombre `logfile.log` el cual recibe la salida estándar (`System.Out`) y de error de Java (`System.Err`) a través del redireccionamiento de las mismas usando `System.setOut` y `System.setErr`. El código que define su implementación es el siguiente:

```
public static void Make_Log()
{
     PrintStream pout = null;

     try
     {
          File tmpFile = new File("logfile.log");
          pout = new PrintStream(tmpFile);
     } catch (IOException ex) {}
          System.setOut(pout);
          System.setErr(pout);
     }
}
```

Este método estático es el único que posee la clase `Logger`, y es llamado una vez el usuario crea el ejecutable.

2.1.4. Cambio de Idioma

El GenSIAD App Creator permite el cambio de idioma a través de varias bifurcaciones que se encuentran distribuidas por el código de la Interfaz Gráfica que chequean el valor del deslizador (del inglés: ***slider***) `slider_language`. El deslizador está programado para tener 2 valores, 0 o 1. En 0, significa que el lenguaje que debe mantener la interfaz es el inglés. En 1, sería el lenguaje opcional

establecido, en este caso el español. Un ejemplo de código de una de las bifurcaciones mencionadas, demuestra cómo se realiza esta comprobación:

```
if(slider_language.getValue()==0)
{
     Object obj_info = "Has Ilegal Chars, Can't Contains:
     / \\ : * ? \" < > |";
     WebOptionPane.showMessageDialog
                    (field_appName,
                    obj_info,
                    "Error in Name Setting",
                    WebOptionPane.ERROR_MESSAGE);
}

if(slider_language.getValue()==1)
{
     Object obj_info = "Tiene Caracteres Ilegales, No
     Puede Contener: / \\ : * ? \" < > |";
     WebOptionPane.showMessageDialog
                    (field_appName,
                    obj_info,
                    "Error Estableciendo Nombre",
                    WebOptionPane.ERROR_MESSAGE);
}
```

2.2. Aplicación incrustada APPSIAD

APPSIAD es una aplicación incrustada (del inglés*:* ***emmbeded***) en el GenSIAD App Creator, forma parte del Kit de herramientas del SIAD, su función principal es la manipulación de la información contenida en una BC del SIAD (.bcs). Cuando una aplicación es generada, o sea, el ejecutable APPSIAD y una BC particular se unen, permite el cálculo de la BC, es decir, permite ejecutar el MSP a la BC para obtener

una o varias soluciones, salvar las soluciones encontradas, etc., este proceso se realiza sobre la BC serializada que a su vez ha sido incrustada en el APPSIAD.

2.2.1. Diseño e Implementación del APPSIAD

La jerarquía de paquetes que componen al APPSIAD (figura 4) se rige por el patrón de diseño MVC. Donde encontramos en el paquete `View` contiene todo lo referente a la Interfaz visual, el paquete `Model` contiene una clase que implementa un tipo de tabla en la cual la primera fila no es editable, útil para mostrar los datos de las soluciones en el frame `ShowSolution` de la interfaz visual, además dicho paquete contiene un paquete vacío: `FileHolder`, que será el lugar de almacenaje de la BC serializada.

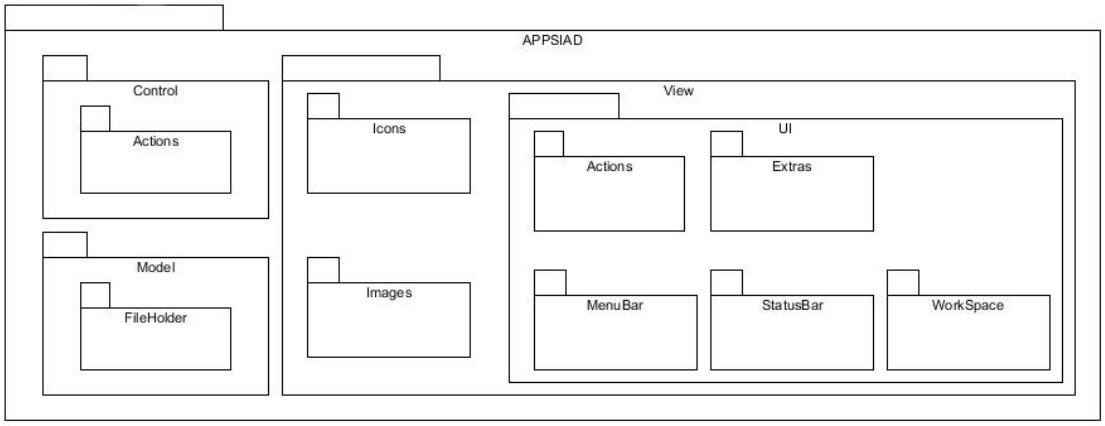


Figura 4: Estructura de paquetes del APPSIAD

Por último está el paquete `Control` en el cual yacen 2 clases encargadas de unificar la interfaz visual con la BC, una clase es capaz de cargar el archivo (.bcs) de la BC serializada,`KnowledgeBase_Loader` y la otra, `KnowledgeBase_n_UI_Controller` está compuesta por varios métodos útiles para el trabajo con el objeto generado, una vez que se des serializa la BC y el llenado de tablas en la Interfaz Visual. El paquete `Control` es el puente en el paquete `Model` y el paquete `View`, apoyando así la disposición del modelo vista controlador.

En el paquete `Model/FileHolder` se debe encontrar un archivo data.dat, una vez generada la aplicación, el cual representa la jerarquía de clases de una BC con

los objetos necesarios ya construidos y listos para el cálculo de soluciones y análisis de datos usando el SIAD.CompilerAndExecuter, una vez se desserialice este archivo, usando el método `Load_bcs` de la clase contenida en `Model/KnowledgeBase_Loader.java` queda instanciado un objeto `BaseC` que es una BC ya compilada y en su forma interna, esto le permite a los métodos de la clase `Model/KnowledgeBase_n_UI_Controller.java` realizar acciones sobre el objeto `BaseC` cómo: validar datos, construir el dominio, calcular soluciones únicas o múltiples, ordenar los datos de una solución por etapas o por parámetros y llenar los datos de las ventanas `Initial_Data` y `Show_Solution`. Para hacer posible este trabajo, en la implementación de la jerarquía de clase del SIAD y el SIAD.CompilerAndExecuter se modificaron todas las clases de las cuales dependía la clase `BaseC` y sus atributos que no fueran tipos primarios de Java para que implementaran la interfaz `Serializable` de Java; además se incluyó el `SIAD.CompilerAndExecuter.jar` como librería de dependencia del APPSIAD.

Todo el procesamiento que le permite al APPSIAD comprender cómo funciona y se utiliza una BC, es responsabilidad de la biblioteca `SIAD.CompilerAndExecuter.jar`, la cual está incluida en las dependencias del APPSIAD. Las clases que permiten estos cálculos, la comunicación con esta biblioteca y el paso de los datos necesarios a la interfaz visual son: `KnowledgeBase_Loader` que realiza el proceso de des serialización como ya se explicó en el epígrafe 1.3.1 y `KnowledgeBase_n_UI_Controller` que permite todo lo demás. El alcance de las funcionalidades del APPSIAD para el usuario es expuesto mediante el siguiente diagrama de casos de uso (figura 5) donde se observa que el usuario tiene permitido: introducir datos, calcular soluciones una vez que los datos han sido insertados correctamente, y guardar las soluciones.

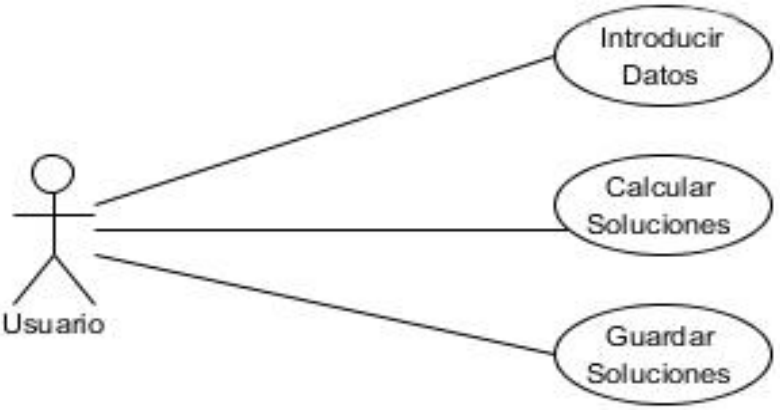


Figura 5: Casos de uso del APPSIAD

Para comprender mejor el funcionamiento del APPSIAD, en el anexo 8 se evidencia la relación entre sus componentes, aquí se observa el uso de la clase que define una tabla con la primera fila “no editable”; así como el uso del MSP del SIAD en casi todos los componentes que permiten al APPSIAD trabajar con el objeto de la BC, la cual esta serializada y ubicada dentro del paquete `FileHolder`.

2.2.2. Implementación del Puente entre SIAD y APPSIAD

La clase `Knowledge_n_UI_Controller` del APPSIAD, contiene 6 atributos privados, necesarios para el almacenaje de información referente a la BC sobre la cual se trabaja:

- `ArrayList<String> domain`: en esta variable, se guarda el dominio de todos los parámetros existentes en la BC, que contengan un dominio previamente declarado.
- `Solution rootSolution`: contiene la solución raíz de la BC dados unos datos iniciales, será la solución de iteración para llenar el árbol de soluciones.
- `SolutionTree treeSolution`: árbol de soluciones, contiene todas las soluciones encontradas.
- `MI infMachine`: la máquina de inferencia es la encargada de calcular las soluciones y almacenarlas en `rootSolution`, en caso de ser válidas.

- `int solutionSelected`: su función es sencilla, guiarse por el árbol de soluciones para saber qué solución se desea mostrar en la interfaz gráfica de la aplicación.
- `String errorInitilD`: errores introducidos en los datos iniciales requeridos para iniciar el cálculo.

Los métodos más complejos presentes en esta clase encargados de llenar los datos de estos atributos o tratar directamente con los datos de la interfaz visual son:

- `private boolean DataValidation(Object, String)`: dada la firma de este método privado, es usado por el método `Set_Data_from_InitialD_Frame()`, para saber cuándo los datos introducidos en las tablas son válidos o no, de ahí que su valor de retorno sea un boolean.
- `public void Domain_Build()`: construye el dominio de todos los parámetros y los inserta en el atributo `domain`.
- `public void Set_Data_from_InitialD_Frame()`: si los valores pasados a través de las tablas son válidos, los pasa a la BC para su posterior procesamiento.
- `public void Build_Solution()`: llena la tabla de solución, basándose en la solución seleccionada en la interfaz gráfica en caso de que exista más de una y el estado de la solución seleccionada cambie.
- `public void Build_Solution(ArrayList<String>)`: llena la tabla de solución, basándose en la solución seleccionada en la interfaz gráfica y muestra solamente los parámetros que hayan sido seleccionados en la lista de parámetros a mostrar.
- `public void Show_Solution()`: llena la tabla de solución una vez se active la opción “”Show Solution” del menú Solution (figura 25, página 52).
- `public void Sort_Solution()`: ordena la solución mostrada de acuerdo al criterio de orden que se especifique de los dos válidos, por etapas o por parámetros.

2.2.3. Implementación de los mensajes de depuración

La herramienta APPSIAD aprovechando el uso de del MSP del SIAD (González, 2015), resume en tiempo de ejecución los mensajes de error que el compilador informa sobre el manejamiento de la BC. Estos mensajes son útiles para el usuario, permitiéndole al mismo tener conocimiento de cualquier error o advertencia de su trabajo. Estos son mostrados en la barra de tareas del APPSIAD, específicamente a la derecha.

Su implementación es muy parecida a la del `LogFile`, dado que está basada en el redireccionamiento de la salida estándar (`System.Out`) y de error de Java (`System.Err`), a diferencia que los resultados se guardan en un archivo temporal para luego ser pasados a la interfaz visual, cada 1 segundo y manteniendo la información actual por 5 segundos en pantalla, esto se puede lograr con el uso de hilos en Java. En el Anexo 6 se muestra el código que define la implementación de estos mensajes.

2.3. Renovando Sintaxis con SIAD New Syntax Converter

Con la modificación de la sintaxis del SIAD, buscando una mejor adaptación a su lenguaje por parte del usuario, llegó también la necesidad de convertir a la nueva sintaxis, las bases de conocimientos ya construidas para las versiones anteriores del SIAD, de aquí que surgiera la idea de automatizar este proceso de transformación, por lo cual se desarrolla el SIAD New Syntax Converter.

2.3.1. Diseño e Implementación del SIAD New Syntax Converter

Siguiendo el patrón de diseño MVC (figura 6), sin la necesidad de un paquete `Model` y teniendo un paquete `View` donde se encuentran las clases y recursos de la interfaz visual, y un paquete `Control` donde se ubica la clase `SyntaxCheck` que transforma la sintaxis. Desde el punto de vista del usuario, tiene permitido: Buscar una BC en la sintaxis del SIAD para MS-DOS y convertirla a la nueva sintaxis del SIAD multiplataforma (figura 7).

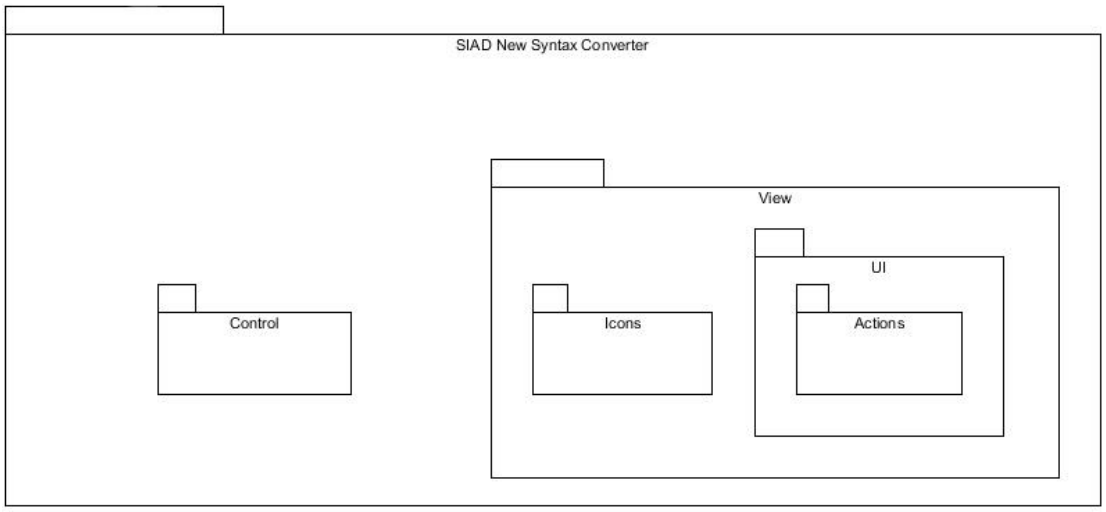


Figura 6: Estructura de Paquetes del SIAD New Syntax Converter Multiplataforma

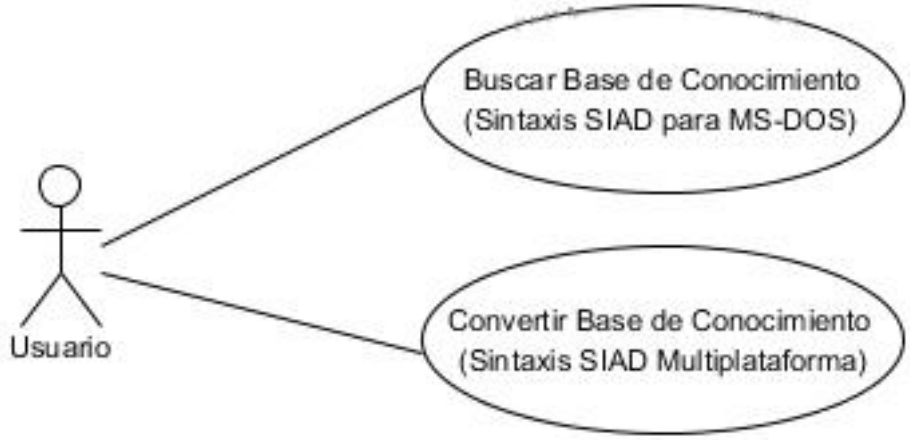


Figura 7: Casos de uso del SIAD New Syntax Converter

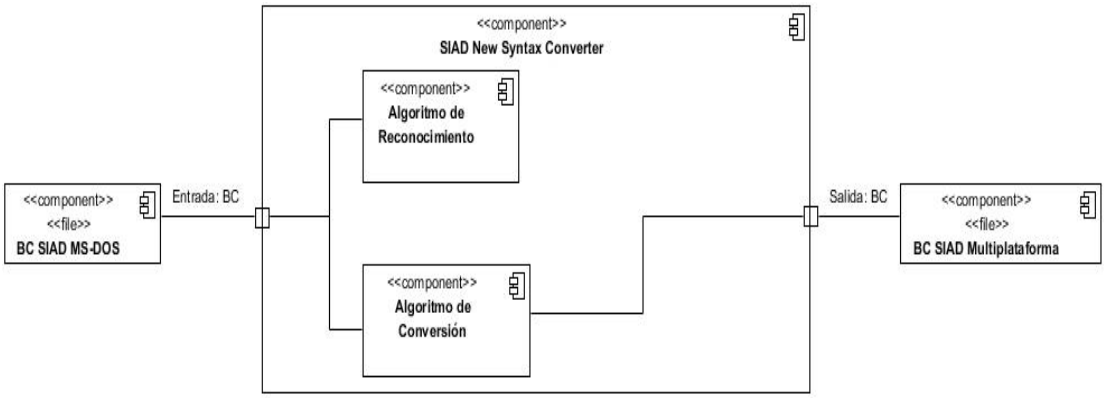


Figura 8: Diagrama de componentes del SIAD New Syntax Converter

Funcionalmente los componentes que integran al SIAD New Syntax Converter (figura 8), solo necesitan de la BC de entrada, cuando sus acciones culminan una BC con sus tokens transformados a la nueva sintaxis del SIAD Multiplataforma es creada, este proceso se describe más detalladamente en el siguiente epígrafe.

De acuerdo a la sintaxis establecida para el SIAD, (Gálvez et al., 1993), la escritura de la BC en un fichero texto (Anexo 2) tiene que satisfacer las siguientes reglas:

- La BC es un fichero texto en el cual se escriben los *frames* correspondientes al Objeto Independiente (OI) y a los Objetos Elementales (OE). El frame que describe el OI se delimita por llaves ({ información sobre el OI }). El frame de cada OE se delimita por corchetes ([información del OE]), el frame correspondiente al último OE termina con dos corchetes ([información del ultimo OE]]).
- Cada elemento de la BC (nombre del OI o de los OE, comentarios, restricciones, PC, PCP, acciones, etc.) se escribe en una línea del fichero texto.
- Cada conjunto de elementos de un mismo tipo tiene un delimitador que precede y sucede al conjunto.
- El primer carácter de un comentario es %.

Los *slots* o campos del frame OI se delimitan según su significado de la forma siguiente:

- Los parámetros importados se delimitan por el carácter $.
- Los parámetros para AutoLISP se delimitan por el carácter @.
- Las restricciones se delimitan por el carácter &.
- La función de evaluación se delimita por ?.
- Las acciones se delimitan por !.

Los *slots* o campos de cada frame OE se delimitan según su significado de la forma siguiente:

- El dominio se delimita por el carácter @.
- Los parámetros antecesores se delimitan por el carácter $.
- Los procedimientos de cálculo se delimitan por el carácter /.
- Los procedimientos para el cálculo del peso se delimitan por el carácter ?.
- Las acciones se delimitan por !.

Los cambios en la sintaxis hechos para la nueva versión de SIAD Multiplataforma (González, 2015)básicamente siguen las mismas reglas que los de la versión de SIAD, la diferencia son los delimitadores, los cuales fueron cambiados en su totalidad, buscando una mejor adaptación por parte del usuario al lenguaje, los cambios están expresados en el Anexo 1.

2.3.2. Implementación del algoritmo de chequeo y transformación de sintaxis usando StringTokenizer

En la teoría de compiladores:

1. Análisis Lexicológico (Aho et al., 2007): El análisis lexicológico o scanner, consiste en la determinación de los componentes básicos e indivisibles de un programa, los cuales hemos denominado tokens o lexemas, usualmente el trabajo de un scanner, puede ser descompuesto en dos procesos:
    - Exploración (Scanning)
    - Simplificación (Screnning)

    La exploración consiste de determinar las subcadenas de caracteres que representen unidades elementales del lenguaje o tokens, dentro de estos se encuentran:
    - Palabras reservadas.
    - Identificadores
    - Operadores
    - Signos de puntuación

    La simplificación consiste en eliminar de la cadena fuente aquellos símbolos que no resultan significativos, tales como las cadenas de espacios y los comentarios.
2. Analizador Sintáctico (Aho et al., 2007): La entrada del analizador sintáctico o parser es la secuencia de tokens generada por el *scanner*. El parser analiza solamente la primera componente de cada token. En este proceso se examina la secuencia de tokens para determinar si cumple ciertas convenciones estructurales de la definición sintáctica del lenguaje.

El algoritmo realiza el primer proceso de un scanner, la exploración. La simplificación no fue incluida dentro de su rango de acción pues no era interés del conversor limpiar el archivo de caracteres innecesarios, tampoco se realiza el

proceso de análisis sintáctico, debido a que el MSP del SIAD Multiplataforma los realiza ambos una vez le es pasada la BC transformada.

Como el chequeo y la transformación de la sintaxis no requieren de objetos o métodos adicionales, el proceso se resume al uso de dos métodos estáticos programados en la clase `SyntaxCheck` del SIAD New Syntax Converter.

El primero de estos métodos, el encargado de chequear que sea una sintaxis válida (no siendo un chequeo como el del parser de teoría de compiladores), simplemente abre el archivo (.bcs) y lo desglosa en tokens.

Como el inicio de estas dos sintaxis tanto la del SIAD Multiplataforma como la del SIAD para MS-DOS, siempre es el token del objeto de diseño, basta con determinar si dicho token es:

- "`{{`" para la sintaxis del SIAD.
- "`.OD`" para la sintaxis del SIAD Multiplataforma.

Se puede encontrar un segmento del código usado para realizar la transformación de sintaxis en el Anexo 3. El algoritmo se basa en recorrer el archivo, línea por línea, desglosándolo también en tokens. Cada vez que encuentra un token de la sintaxis del SIAD para MS-DOS, lo sobrescribe por su equivalente de la nueva (SIAD Multiplataforma). En caso de que sea una expresión, el algoritmo tiene bien definido donde puede existir una, por lo que no importa si los tokens están juntos o separados.

El algoritmo busca funciones de la sintaxis del SIAD para MS-DOS, que siempre se escriben con minúscula y las sobrescribe por sus equivalentes en la sintaxis del SIAD Multiplataforma las cuales se escriben con mayúsculas y casi siempre tienen los mismos nombres. En cuanto a los operadores, el único operador que cambia entre las dos sintaxis, es el operador distinto (no igual):

- "`<>`" para la sintaxis del SIAD MS-DOS.
- "`!=`" para la sintaxis del SIAD Multiplataforma.

Por lo expuesto anteriormente se afirma que el mecanismo de transformación implementado basado en `StringTokenizer` es equivalente a la realización de un scanner en la teoría de compiladores.

2.4. Conclusiones Parciales

Se implementó el generador de aplicaciones del SIAD (GenSIAD App Creator) que permitió la creación de aplicaciones específicas usando una BC, que serializa y le integra a la aplicación resultado como parte de su proceso de uso.

Se implementó el APPSIAD, aplicación embebida en el GenSIAD App Creator, que es extraída con una BC integrada y serializada, funciona como ejecutable y aplicación específica para un usuario final, permitiendo la manipulación del conocimiento impregnado en la BC que se le integra, desde el punto de vista de introducción de datos, análisis de soluciones y salva de resultados.

Se implementó el SIAD New Syntax Converter, herramienta capaz de transformar una BC escrita en la antigua sintaxis, a una escrita en la sintaxis del SIAD multiplataforma, permitiéndole al GenSIAD App Creator, reconocerla e integrarla a su proceso de creación de un ejecutable.

Las tres aplicaciones son desarrolladas usando Java, y dando como resultado un paquete de herramientas complementarias multiplataforma para la nueva versión del SIAD multiplataforma.

Capítulo 3. Utilización de las herramientas complementarias para el SIAD y su extensión usando plugins

En este capítulo se exponen las principales razones por las cuales se desarrollan estas herramientas y las ventajas que trae su uso tanto para el usuario final como para el ingeniero del conocimiento. También se describen algunos elementos relacionados con el uso de las herramientas propuestas y con la interfaz de usuario que estas ofrecen.

3.1. Requisitos de instalación de las herramientas complementarias multiplataforma para el SIAD

Para garantizar la ejecución de la versión multiplataforma de las herramientas complementarias sobre cualquier Sistema Operativo, se requiere tener instalado una Máquina Virtual de Java en su versión 8.0 o superior.

Para el correcto funcionamiento de las herramientas, la carpeta lib, con los archivos:

antlr-4.5-complete.jar

SIAD.CompilerAndExecuter.jar

weblaf-complete-1.28.jar

3.2. Uso de la herramienta GenSIAD App Creator

El GenSIAD App Creator tiene como funcionalidad serializar una BC con una sintaxis del SIAD válida y generar una instancia de la aplicación APPSIAD capaz de usarla, un usuario final sin conocimientos de programación puede ejecutar este proceso en segundos.

Para chequear que el proceso concluyó correctamente, es generado un archivo de registro en el cual quedan plasmados los errores y advertencias del compilador al intentar compilar la BC, en caso de que no haya sido compilada usando el SIAD con anterioridad por un ingeniero del conocimiento, lo cual ratificaría su validez, dando como resultado una aplicación completamente funcional.

3.2.1. GenSIAD App Creator en acción

Como la funcionalidad principal del GenSIAD App Creator es crear una nueva instancia de la aplicación APPSIAD con una BC serializada incrustada, se necesita

un archivo (.bcs) (figura 9) escrito en la sintaxis del SIAD Multiplataforma, un ejemplo de este tipo de archivo se muestra en la figura 10.

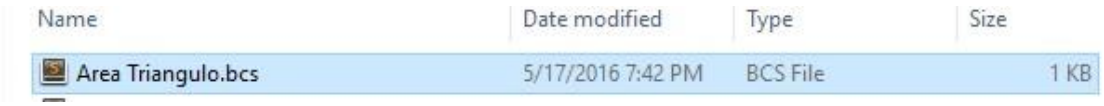


Figura 9: Archivo Ejemplo de una BC

```
.OD
  Triangulo
  .RD
    A < 20
  .FRD
.FOD

.PD
  H
  .DOMP
    4 6 8 10
  .FDOMP
.FPD

.PD
  B
  .PCP
    -> H/2
    -> H/3
  .FPCP
.FPD

.PD
  A
  .PCP
    -> B/2*H
  .FPCP
.FPD
```

Figura 10: Código Ejemplo de una BC

La Aplicación está estructurada por un archivo ejecutable de Java GenSIAD_App_Creator.jar (figura 11) y una carpeta con las bibliotecas de dependencias que se resumen en el epígrafe 3.1 con el nombre `lib`.

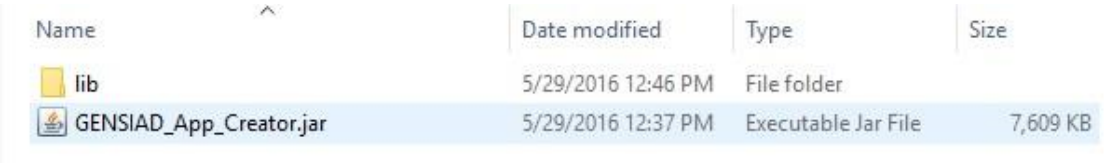


Figura 11: Archivos del GenSIAD App Creator

3.2.2. Interfaz visual del GenSIAD App Creator

La interfaz visual del GenSIAD App Creator (figura 12), es sencilla y muy intuitiva, por defecto la misma viene en inglés, pero se puede cambiar en tiempo de ejecución a español.

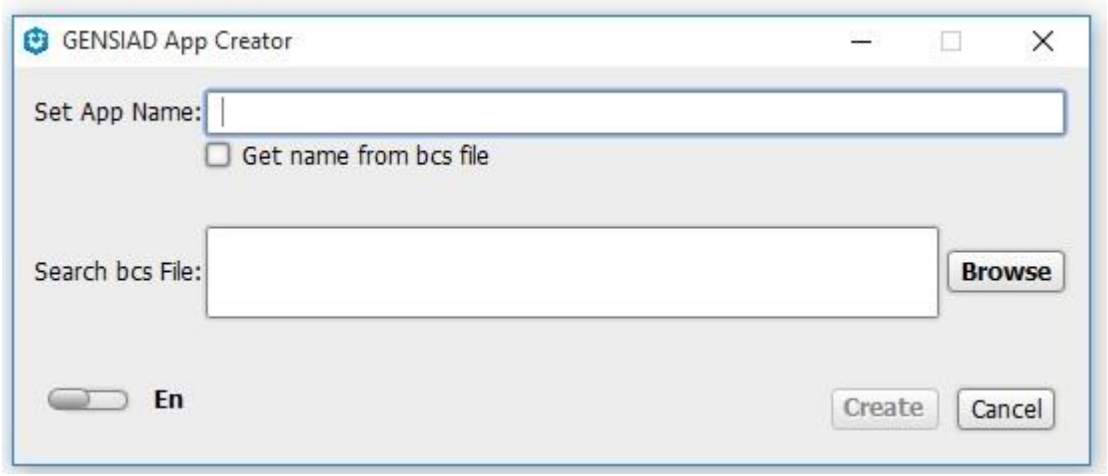


Figura 12: Interfaz Visual del GenSIAD App Creator

En la interfaz visual existen dos campos de texto, el superior está destinado a otorgarle un nombre al archivo (.jar) que se genera o ejecutable de la aplicación APPSIAD en cuestión, y el inferior es completado con la dirección absoluta del archivo (.bcs) que le será integrada al programa, para hacer esto último se pulsa el botón de buscar y se selecciona un archivo válido para cargar (figura 13).

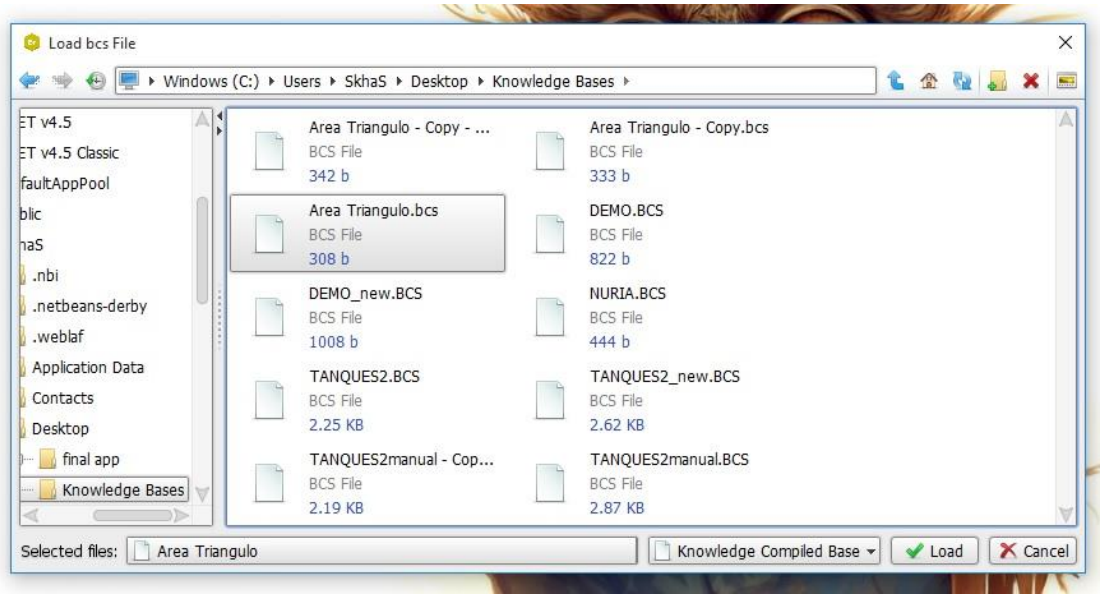


Figura 13: Panel de Diálogo para cargar un archivo (.bcs)

Luego se pulsa el botón de crear y es creada, una carpeta llamada BuildedApps en el mismo directorio donde se encuentre el GenSIAD App Creator, en ella estarán todas las aplicaciones con bases de conocimientos integradas, también se creará un archivo logfile.log con los registros del compilador a la hora de serializar la BC en la carpeta raíz de la aplicación (figura 14).

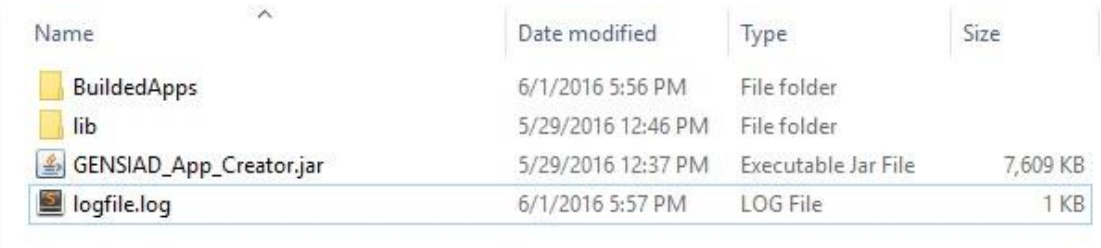


Figura 14: GenSIAD App Creator y los archivos resultantes

Si todo sale bien, el logfile lucirá como el mostrado en la figura 15.

```
1  EXCEPTION -> [null]
2
```

Figura 15: Archivo de registro o logfile sin errores

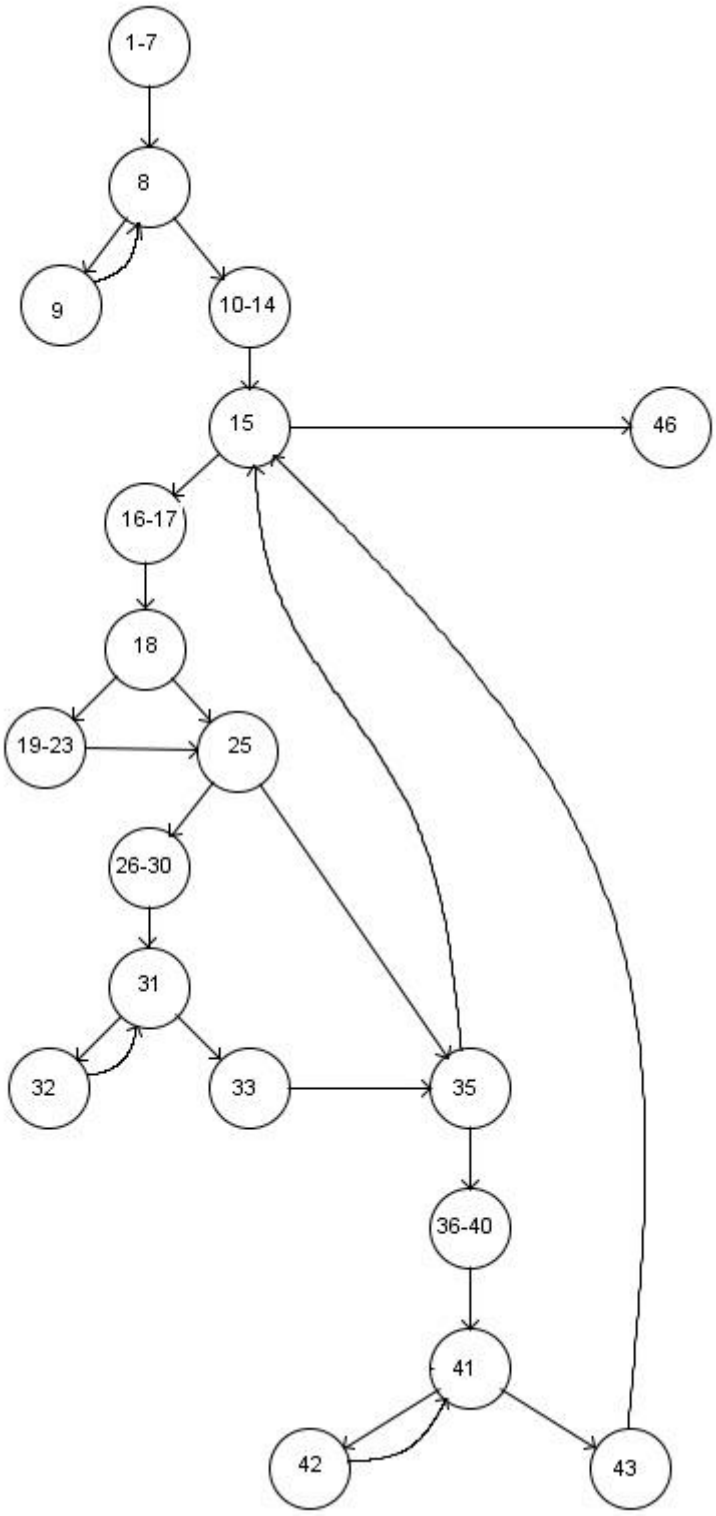
3.2.3. Aplicándole esquema de pruebas: caja blanca al GenSIAD App Creator, específicamente, cobertura de caminos

Los componentes fundamentales del GenSIAD App Creator son los métodos donde se encuentran programados los algoritmos de empaquetamiento, desempaquetamiento y compilación, serialización. De estos componentes se muestran los códigos en las figuras 16 y 17 respectivamente.

```
public void Zipper(String s){
    InputStream is = ClassLoader.getSystemResourceAsStream("Model/TemplateApp/APPSIAD.jar");
    try {
        File tmpf = File.createTempFile("gensiadtmp", null);
        FileOutputStream fos = new FileOutputStream(tmpf);
        int readed = 0;
        byte buffer [] = new byte [4096];
        while((readed=is.read(buffer, 0, buffer.length)) != -1)
            fos.write(buffer, 0, readed);
        ZipFile zfile = new ZipFile(tmpf);
        Enumeration zenum = zfile.entries();
        bcsFile = new File("data.dat");
        FileInputStream finput = new FileInputStream(bcsFile);
        ZipOutputStream zoutput = new ZipOutputStream(new FileOutputStream(new File(homePath,s + ".jar")));
        while(zenum.hasMoreElements()){
            ZipEntry zentry = (ZipEntry) zenum.nextElement();
            ZipEntry tmp = new ZipEntry(zentry);
            if(zentry.isDirectory()){
                tmp.setCompressedSize(0);
                tmp.setSize(0);
                tmp.setCrc(0);
                zoutput.putNextEntry(tmp);
                zoutput.closeEntry();
            }
            if(!zentry.isDirectory()){
                zoutput.putNextEntry(tmp);
                InputStream inputs = zfile.getInputStream(zentry);

                readed = 0;
                buffer = new byte [4096];
                while((readed = inputs.read(buffer,0,buffer.length))!= -1 )
                    zoutput.write(buffer,0,readed);
                zoutput.closeEntry();
            }
            if(zentry.getName().equals("Model/FileHolder/")){
                zentry = new ZipEntry("Model/FileHolder/data.dat");
                zoutput.putNextEntry(zentry);

                readed =0;
                buffer = new byte[4096];
                while((readed = finput.read(buffer,0,buffer.length))!= -1 )
                    zoutput.write(buffer,0,readed);
                zoutput.closeEntry();
            }
        }
        zfile.close(); finput.close(); zoutput.close(); bcsFile.delete();
    } catch (IOException ex) {}
}
```

Figura 16: Código del método de empaquetamiento, desempaquetamiento

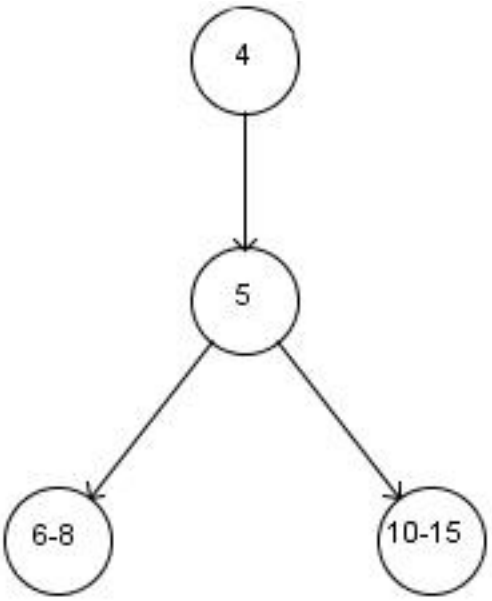
Los grafos 1 y 2 representan el seguimiento que se hizo, con el uso del esquema de prueba de caja blanca, enfocado en la cobertura de caminos a los mencionados componentes respectivamente. Un análisis de estos grafos, demuestra la seguridad y confianza que presentan dichos componentes, que constituyen la columna vertebral del GenSIAD App Creator.



Grafo 1: Prueba de caja blanca, cobertura de caminos del método de empaquetamiento, desempaquetamiento

```
public void Compiler_Serializer()
{
    try{
        CharStream stream = new ANTLRInputStream(new FileInputStream(bcsFile));
        if(stream == null){
            Object obj_info = "ANTLRv4 couldn't open the file";
            WebOptionPane.showMessageDialog(mainFrame, obj_info,"Error Opening File",WebOptionPane.ERROR_MESSAGE);
            mainFrame.Button_Create_Disable();
        }else{
            BuildDatos interpreter = new BuildDatos(stream);
            interpreter.execute();
            bcs = interpreter.getBc();
            ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream("data.dat"));
            os.writeObject(bcs);
            os.close();
        }
    } catch (FileNotFoundException ex) { }
      catch (IOException ex) { }
}
```

Figura 17: Código del método de compilación, serialización



Grafo 2: Prueba de caja blanca, cobertura de caminos del método de compilación, serialización

3.3. Importancia y ventajas del uso e implementación del APPSIAD

Dado que el usuario final, no necesariamente tiene por qué conocer la sintaxis para programar una BC, sin embargo, si puede tener conocimiento técnico del objeto de diseño en cuestión, por lo cual, si tuviera una aplicación con la descripción de los datos y la posibilidad de modificarlos y usar la máquina de inferencia para hacer los cálculos sin tener que recurrir al SIAD.

De este planteamiento surge la idea de crear el APPSIAD, una aplicación con una interfaz de usuario y capaz de realizar los mismos cálculos que el SIAD, pero sin la necesidad de programar nada, ya que se adapta a la BC que se le integre, permitiendo a los ingenieros del conocimiento guardar y automatizar el proceso de análisis de una pieza previamente construida con el SIAD.

Una BC compilada por el SIAD puede ser integrada al APPSIAD sin necesidad de modificar, dado que ya fue compilada, lo que prueba su estado de validez. Ahora si se integra, su modificación es imposible, pero permite jugar con los valores establecidos, ya que existen dominios y reglas que permiten una, ninguna o varias soluciones, cada una de ellas puede ser guardadas para su posterior análisis con el conjunto de datos iniciales que se usaron.

3.3.1. Usando APPSIAD

Comencemos con los archivos y restricciones necesarias para su correcta ejecución.

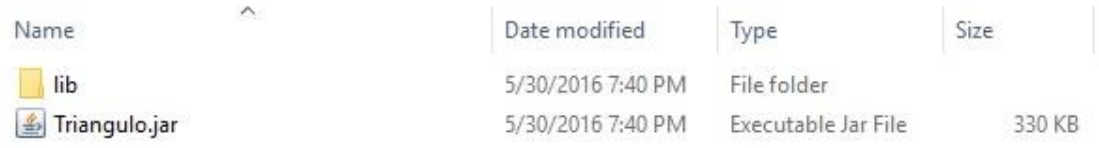


Figura 18: Archivos de una aplicación generada compuesta por el APPSIAD.jar y una BC: “Area Triangulo.bcs”

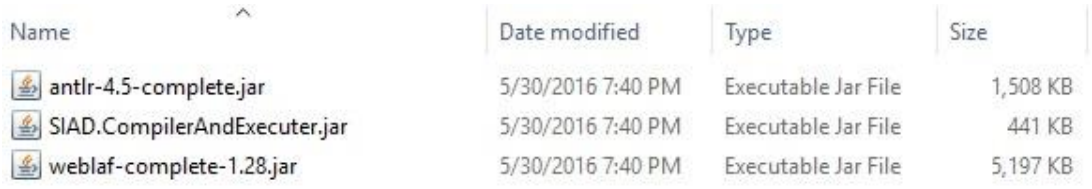


Figura 19: Bibliotecas de dependencia de cualquier aplicación generada de APPSIAD.

En la figura 18 aparecen el ejecutable de una aplicación generada compuesta por el APPSIAD.jar y una BC: “Area Triangulo.bcs” (dando como resultado una

aplicación(.jar) con un nombre escogido por el usuario, en este caso: Triangulo.jar), además de las bibliotecas necesarias para su correcta ejecución (figura 19).

Figura 20: Pantalla de carga del APPSIAD

Una vez ejecutado Triangulo.jar, se mostrará la pantalla de carga del APPSIAD (figura 20) la cual crea y carga todos los componentes de la interfaz visual, así como la BC serializada.

3.3.2. Interfaz visual del APPSIAD

La interfaz visual del APPSIAD (figura 21), presenta un ambiente de trabajo fluido y agradable presenta un diseño basado en un espacio de trabajo y usa marcos internos (del inglés: ***internal frames***) de java, los cuales permiten ser chequeados en cualquier momento sin ser cerrados, siempre que se cumplan las condiciones para su habilitación y funcionamiento.

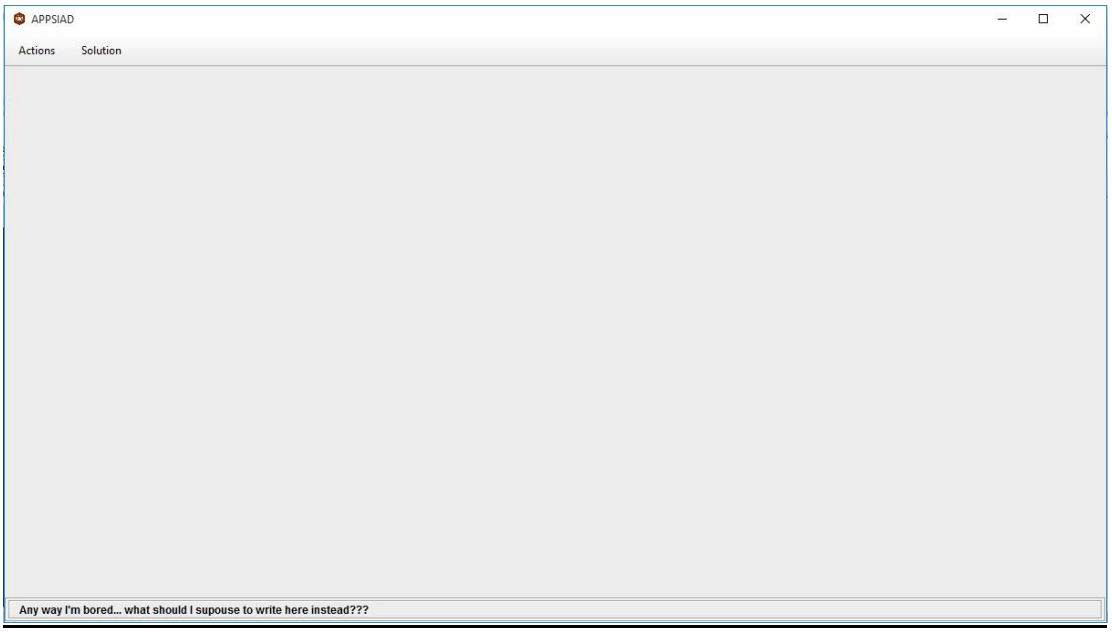


Figura 21: Interfaz gráfica del APPSIAD, para la aplicación específica Triangulo.jar

Además, cuenta con una barra de estado la cual a su derecha proporciona información en tiempo real relacionada con el compilador y sus acciones, ya sea simple información o errores.

Una vez cargada la interfaz visual, el proceso es sencillo, se abre el marco interno Initial Data (figura 22) a través del menú Actions (figura 23), o usando el acceso a teclas rápidas (Ctrl+D), válida la aclaración, todos los componentes del menú del APPSIAD cuentan con teclas de acceso rápido.

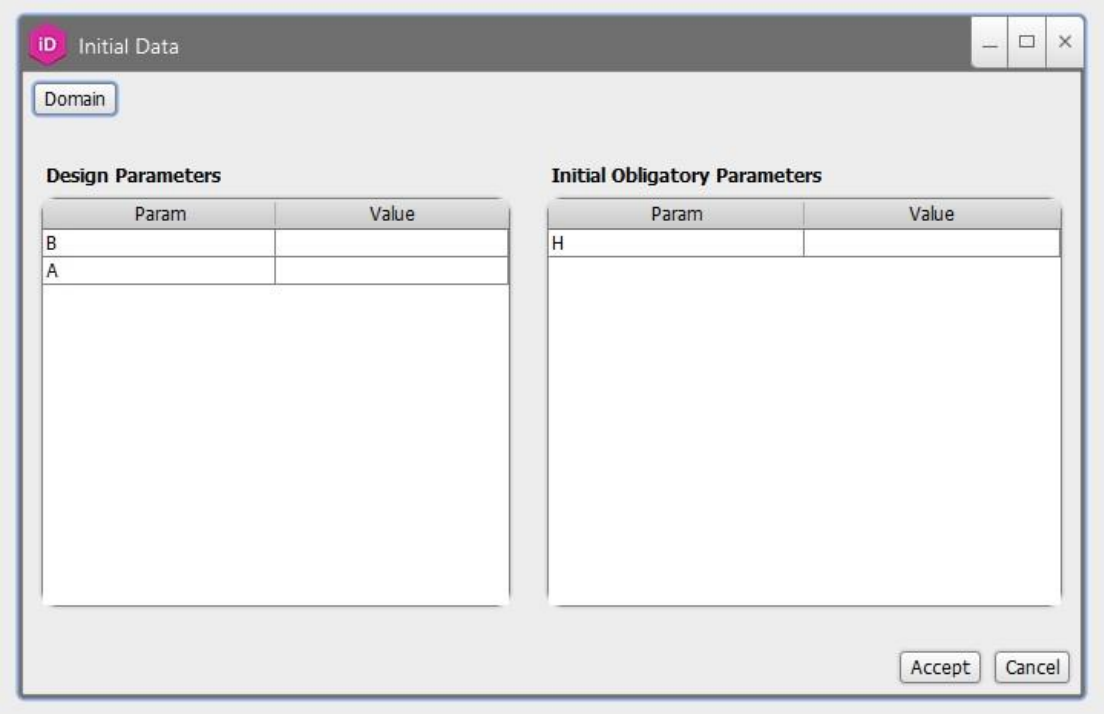


Figura 22: Marco interno de datos iniciales del APPSIAD

Figura 23: Menú de acciones del APPSIAD

En caso de ser necesario, se puede chequear el dominio de los parámetros, medianteel botón Domain (figura 24) esquina superior derecha del marco interno

Initial Data, este hace una búsqueda en la BC y rellena el panel con los nombres de los argumentos seguidos por sus respectivos dominios.

Figura 24: Panel de dominio.

Una vez que se hayan introducido correctamente los valores iniciales para los parámetros que los requieran o los admitan, podemos proseguir con el cálculo de las soluciones, usando la opción Calc del menú Actions (figura 23). A continuación el programa mostrará un cartel indicando si se encontraron soluciones o no; en caso de ser cierto, existen varias opciones, es posible ir a la opción Show Solution del menú Solution (figura 25), o bien puede probarse si existe una siguiente solución alternativa con la opción Next Solution del menú Actions (figura 23), si se desea probar más soluciones con otros datos iniciales, podemos cambiar los datos, a continuación calcular y usar la opción Multiple Solutions del menú Actions (figura 23).

Figura 25: Menú de soluciones del APPSIAD

Si se siguió el proceso correctamente y se encontró al menos una solución, se debe pasar entonces a estudiar los datos en el marco interno Show Solution (figura 26).

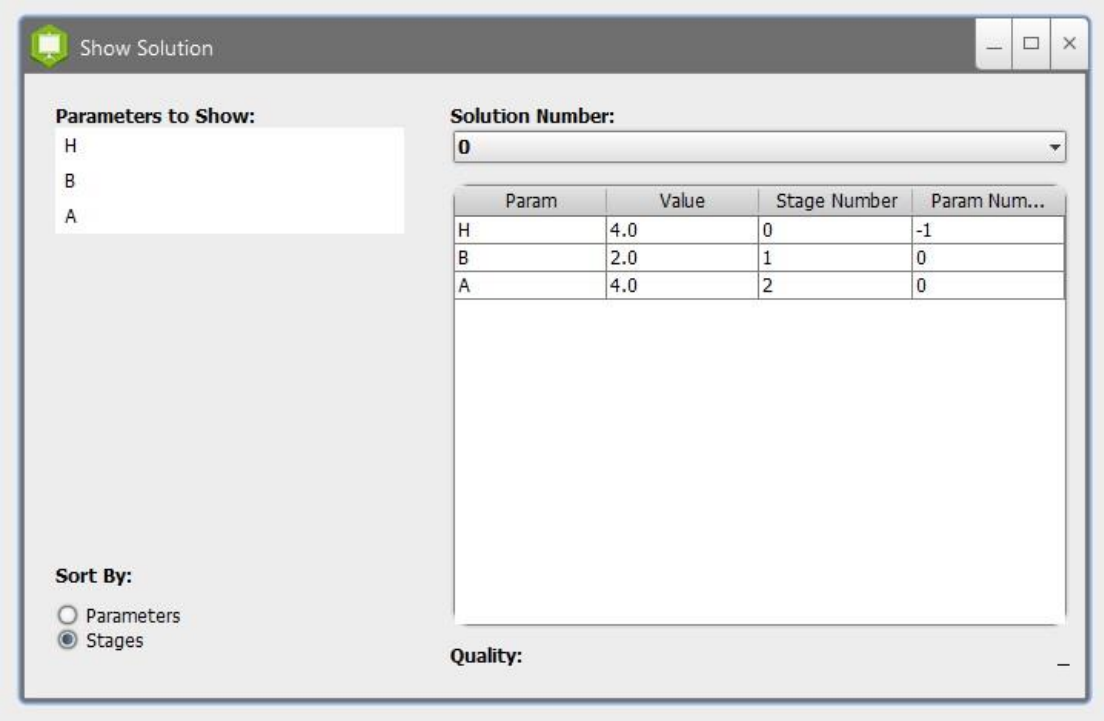


Figura 26: Marco interno, mostrar solución del APPSIAD

Se tienen varios componentes en este marco interno, empezando por la esquina superior derecha, donde se permite seleccionar la solución a mostrar, debajo está la tabla que muestra la solución seleccionada, con cuatro encabezados de columnas: parámetros (del inglés: ***parameters***), valor (del inglés: ***value***), número de etapa (del inglés: ***stage number***) y número del parámetro (del inglés: ***parameter number***). En la esquina inferior derecha se encuentra una etiqueta: calidad (del inglés: ***quality***) seguida por otra etiqueta con el texto "_", la cual nunca cambia, debido a que la función de cálculo de calidad de una solución dada, nunca fue programada en el compilador, pero se mantuvo el componente en caso de que se implemente dicha función en futuras versiones del SIAD. En la esquina superior izquierda se encuentra una lista, que permite selecciones múltiples, la cual muestra en la tabla de la derecha, solo los parámetros que hayan sido seleccionados en esta lista. Por último, tenemos en la esquina inferior izquierda para seleccionar entre el tipo de ordenamiento que deseamos aplicar a la tabla donde se muestra la solución, ordenamiento por parámetros o por etapas. Si se está satisfecho con los datos de una solución y se desea guardar, para su posterior estudio u otro fin, se debe acceder a la opción Save Solution del menú Solution (figura 25), la cual abrirá un

panel con la opción de guardar la solución seleccionada, con los datos que se estén mostrando actualmente en la tabla, en un archivo (.txt) de su elección (figura 27).

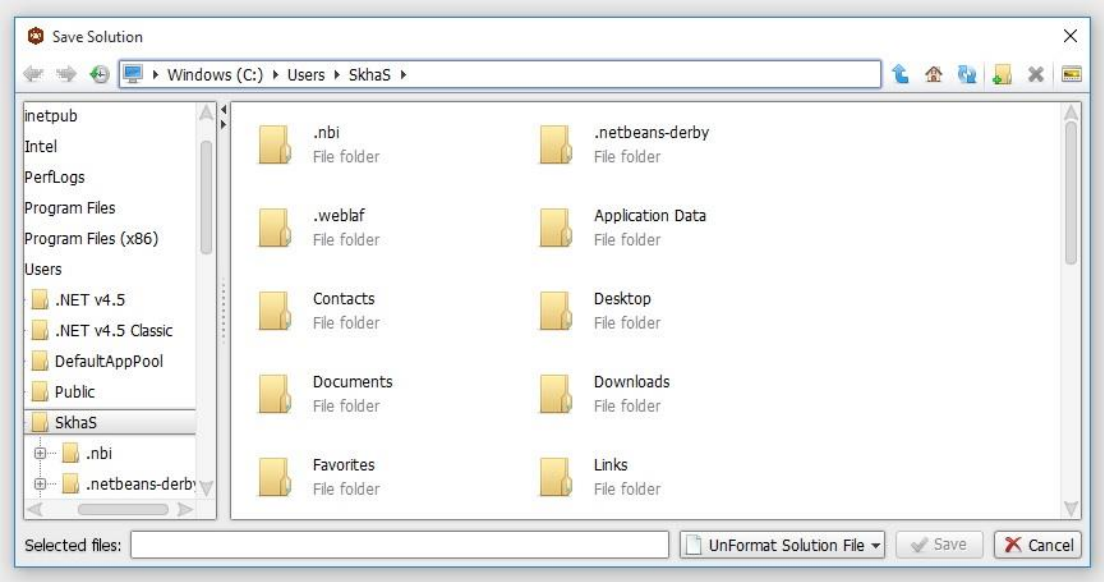


Figura 27: Panel salvar solución del APPSIAD

Si la solución se guardó correctamente entonces el archivo generado, internamente lucirá como este (figura 28).

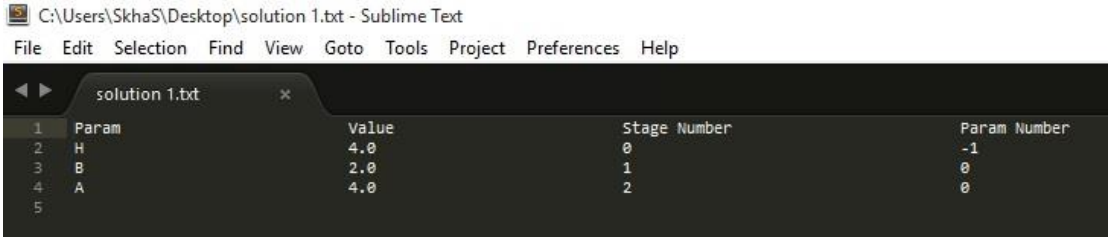


Figura 28: Ejemplo de una solución salvada por el APPSIAD

3.3.3. Propuesta de aplicación del esquema de pruebas: caja negra al APPSIAD, específicamente, prueba de entrada/salida

Como referencia para demostrar la funcionalidad, seguridad y estabilidad de las aplicaciones desarrolladas, se propone la aplicación del esquema de pruebas: caja negra, específicamente la prueba de entrada/salida al APPSIAD.

El APPSIAD cuenta con múltiples funcionalidades, por lo cual sería engorroso y extenso aplicarle una prueba de cobertura de caminos al igual que se le realizó al GenSIAD App Creator.

La utilización del APPSIAD se centra principalmente en el uso y manipulación de los datos que se le introducen en la interfaz visual; por cuanto, una prueba de caja negra en general sería lo óptimo para demostrar la funcionalidad de dicha aplicación.

3.4. Importancia y ventajas del uso e implementación del SIAD New Syntax Converter

Por haber sido usado durante tanto tiempo el SIAD para DOS y para Windows, basados en otra sintaxis de este lenguaje, surge la necesidad de traducir las BC guardadas a la nueva sintaxis del SIAD y sus herramientas multiplataforma. Ello evidencia la importancia del desarrollo del SIAD New Syntax Converter, que automatiza el proceso de conversión de una BC programada en la antigua sintaxis a una en la nueva, lo cual permite la compatibilidad con: el SIAD, el APPSIAD y el GenSIAD App Creator.

Su uso es muy intuitivo, basta con cargar una BC y presionar el botón de convertir, el programa avisa si la sintaxis ya pertenece a la nueva, si no es válida o en caso contrario de no ocurrir ninguna de estas situaciones, genera un archivo (.bcs) con el mismo nombre más el sufijo "_new", con la BC transformada.

En el Ejemplo 3, del Anexo 4, se observa la conversión de las sintaxis, el proceso se completa perfectamente sin errores, se observa que la versión nueva de la BC convertida sigue al pie de la letra la estructura de la Sintaxis del SIAD Multiplataforma. Sin embargo, en el Anexo 5 se presentan los resultados del logfile que se crea una vez que el GenSIAD App Creator, intenta crear una instancia de APPSIAD usando la BC generada.

Después de varias pruebas, se llegó a la conclusión de que el `SIAD.CompilerAndExecuter.jar` presenta algunas deficiencias a la hora de procesar comentarios, específicamente comentarios que son operadores en la nueva sintaxis, ejemplo: `Peso específico del líquido en Kg/m3.`

Este comentario presenta el carácter '/' que representa el operador de división en la nueva sintaxis. Usando la misma BC, se eliminaron los comentarios como este, que tuvieran caracteres equivalentes a operadores en la nueva sintaxis y el proceso funcionó correctamente, o sea el GenSIAD App Creator, generó correctamente una instancia del APPSIAD, con la BC transformada incrustada.

3.5. Flujo de trabajo e integración del paquete de herramientas complementarias y el SIAD

El flujo de trabajo del paquete de herramientas complementarias y el SIAD es unidireccional y su integración es evidenciado a través de las BC. Ninguna de las herramientas se relaciona directamente con otra o el SIAD. En la figura 27, se emplea un diagrama para ejemplificar lo anteriormente expuesto.

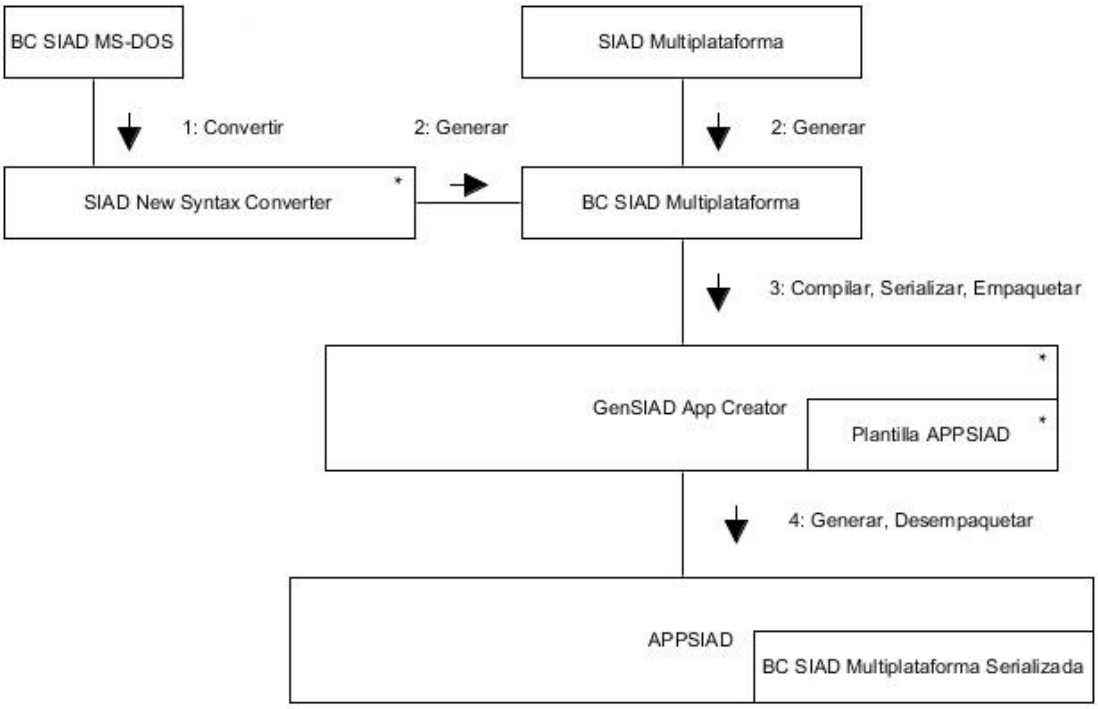


Figura 27: Diagrama del flujo de trabajo e integración del paquete de herramientas complementarias y el SIAD (* producto del desarrollo de este trabajo)

Descripción del Flujo de Trabajo:

1. Convertir: Se usa el SIAD New Syntax Converter para la transformación de la sintaxis de una BC del SIAD para MS-DOS.

2. Generar: Se genera la BC convertida por el SIAD New Syntax Converter como un archivo (.bcs).
2. Generar: Se genera un archivo (.bcs) de una BC escrita en el SIAD Multiplataforma.
3. Compilar, Serializar, Empaquetar: Se compila la BC, luego se serializa y posteriormente es empaquetada en el paquete `FileHolder` de la plantilla del APPSIAD, todo esto es efectuado como parte del proceso de uso del GenSIAD App Creator.
4. Generar, Desempaquetar: Se genera un ejecutable de java (.jar) usando el APPSIAD como plantilla con la BC ya serializada y empaquetada, Se desempaquetan el APPSIAD y las bibliotecas necesarias para su ejecución.

3.6. Propuesta de herramientas a reescribir como plugins

Para futuras versiones de estas herramientas, se pudieran integrar al SIAD Multiplataforma en forma de plugins, aumentando así el campo de acción de este y mejorando la interacción con el usuario.

Herramientas a integrar:

- GenSIAD App Creator
- SIAD New Syntax Converter

3.6.1. Incorporación de las herramientas al SIAD

Siguiendo el proceso de creación de plugins en Java:

1. Es necesario crear una interfaz (del inglés: ***interface***) que todos los plugins deben implementar, esta debe constar de como mínimo cuatro métodos principales:
    - `public String getPluginName()`: devuelve el nombre del plugin, de manera que el usuario pueda identificarlo de alguna forma.
    - `public void setParameters(List <Object> params)`: le pasa al plugin una lista de parámetros, para trabajar con ellos, sin tener en cuenta cual es la acción que desea realizar con estos.

- `public List <Object> getResult()`: a través de este método, la aplicación principal puede obtener el resultado del trabajo realizado por el plugin.
- `public boolean hasError()`: determina si la llamada anterior al plugin, no fue satisfactoria.

2. Determinar cómo se adquieren los plugins y en qué momento la aplicación debe escanear para descubrir nuevos: una vía muy sencilla, práctica y usada en las aplicaciones de escritorio, es crear una carpeta `plugins` en el mismo directorio raíz de la aplicación y escanear en el momento de carga de esta y en ejecución; solo cuando el usuario use el menú `plugins`.
3. Crear una `ClassLoader`: extendiendo de `java.lang.ClassLoader`, en la cual solamente hay un método que implementar (`loadClass`), y se le pasa como parámetro un `String` con el nombre de la clase a cargar y devuelve un objeto de tipo `Class` de la clase que carga, de este objeto la aplicación puede instanciar en tiempo de ejecución objetos de la clase del plugin.
4. Finalmente, se necesita un `SecurityManager`: extendiendo de `java.lang.SecurityManager`, solamente es necesario sobrescribir (del inglés: ***override***) los métodos que especifican que permisos dar y que permisos denegar, como son: permisos de entrada/salida (del inglés: ***input/output***) del sistema operativo, acceso a la impresora, portapapeles (del inglés: ***clipboard***), ejecución de código externo, acceso a propiedades del sistema y cerrar la aplicación.

En el anexo 9 se presenta un diagrama de clases que sigue el proceso de creación de plugins para el SIAD, en este se resumen las clases, relaciones y métodos necesarios a implementar para la puesta en práctica de esta propuesta.

3.7. Conclusiones parciales

Se evidentó la utilidad de las herramientas desarrolladas (GenSIAD App Creator, APPSIAD, SIAD New Syntax Converter).

Se le aplicó el esquema de pruebas: caja blanca, al GenSIAD App Creator; específicamente la cobertura de caminos en sus dos métodos principales.

Se propuso la aplicación del esquema de pruebas: caja negra, al APPSIAD, para demostrar su validez y la continuación del cumplimiento de las características principales de su predecesor (la plantilla del ejecutable que creaba el GENSIAD).

Se abordó el tópico de plugins, desde una perspectiva futurista con respecto a las capacidades que se desean para el paquete de herramientas. Se expusieron las principales acciones a seguir para el cumplimiento de estas tareas, de forma más específica. Se planteó un proceso para poder incorporar dichas herramientas al SIAD en forma de plugins, agregándole funcionalidad a este.

Conclusiones

Con este trabajo se logró el diseño e implementación de un grupo de herramientas de apoyo al SIAD que son necesarias para aumentar la capacidad de esta herramienta y permiten conformar nuevamente un paquete de herramientas similar al existente para DOS.

Se desarrolló:

- El SIAD New Syntax Converter, que se encarga de la conversión automática de la vieja sintaxis de las BC del SIAD a la nueva sintaxis.
- El GenSIAD App Creator que es capaz de generar un ejecutable en Java para una BC particular, utilizando para ello la aplicación incrustada APPSIAD que contiene una interfaz estándar para el usuario final y el MSP del SIAD.

Como parte de este trabajo se propuso el uso del patrón plugin para extender la versión actual del SIAD de manera que pueda incorporar estas herramientas.

Recomendaciones

- Modificar las dos aplicaciones (GenSIAD App Creator ySIAD New Syntax Converter) de manera que satisfagan el patrón plugins e integrarlas al SIAD.

- Implementar la opción de personalización en el GenSIAD App Creator en función de personalizar la interfaz gráfica de las aplicaciones a generar.

- Realizar la implementación de un analizador sintáctico con el apoyo de teoría de compiladores como apoyo al método de conversión actual que presenta el SIAD New Syntax Converter el cual no posee esta característica.

- Aplicar pruebas funcionales automatizadas a las herramientas o con la ayuda de un especialista.

Bibliografía

AHO, A. V., LAM, M. S., SETHI, R. & ULLMAN, J. D. 2007. Compilers. Principles, Techniques, & Tools 2nd ed.: Addison Wesley.

ALVARES, M. L. 1994. Fundamentos de Inteligencia Artificial.

BANARES-ALCANTARA, R., VENKATASUBRAMANIAN, V., WESTERBERG, A. W. & RYCHENER, M. D. 1984. Knowledge-based expert systems : an emerging technology for CAD in chemical engineering. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1066&context=cheme.

BELLO, P. R., GÁLVEZ, L. D. & GARCÍA, L. M. M. 1994. Knowledge representation form in mechanical engineering.

BELLO, P. R., GÁLVEZ, L. D., GARCÍA, V. Z., GARCÍA, L. M. M. & LOBATO, R. A. 2003. Aplicaciones de la Inteligencia Artificial.

BRACHMAN, R. J. & LEVESQUE, H. J. 2004. The basics of knowledge representation and reasoning. Elsevier.

DITTMER, U. 2010. *Adding Plugins to a Java Application* [Online]. Available: http://www.javaranch.com/contact.jsp#UlfDittmer [Accessed].

EDWARDS, J. 1991. Building Knowledge-Based Systems.

ELIOTTE, R. H. 1999. Java I/O First Edition. 1st ed.: O'Reilly.

FERREIRO, A. M. B., RODRÍGUEZ, J. L. M., MÉNDEZ, C. A. C., DELFRADES, A. I. M. & PÉREZ, J. A. V. 2007. Aplicaciones de los Sistemas Basados en el Conocimiento al diseño y descifrado de reductores de ciclo para la industria azucarera. Available: https://www.researchgate.net/publication/278783532.

FRIENDLAND, P. 1985. Knowledge-based systems. Comm of ACM.

GÁLVEZ, L. D., BELLO, P. R. & GARCÍA, L. M. M. 1993. Manual de usuario del SIAD version 2.50.

GONZÁLEZ, E. A. & HERNÁNDEZ, L. A. A. 2001a. Desarrollo de una nueva versión del SIAD sobre Windows.

GONZÁLEZ, E. A. & HERNÁNDEZ, L. A. A. 2001b. Ingeniero del Conocimiento Automatizado para el SIAD (versión 1.0 para Windows).

GONZÁLEZ, L. E. C. 2015. *Desarrollo de una versión multiplataforma del Sistema Inteligente de Ayuda al Diseño.*

HAYES, R. F., WATERMAN, A. D. & LENAT, B. D. 1983. Building Expert Systems.

HORSTMANN, C. 2008. Big Java (For programmers) Third Edition. 3rd ed.: Wiley.

INÉS, C. M. 2012a. *Formalismo o Esquema de Representación del Conocimiento* [Online]. Available: http://dharma.frm.utn.edu.ar/cursos/ia/2012/material/APUNTES-FILMINAS/U2/RCyFormalizacionCesari.pdf [Accessed].

INÉS, C. M. 2012b. *Sistemas Expertos* [Online]. Available: http://dharma.frm.utn.edu.ar/cursos/ia/2012/material/APUNTES-FILMINAS/U2/SEapuntesCesari.pdf [Accessed].

LEÓN, E. M. 2007. Ingeniería del Conocimiento Automatizada en la creación del Modelo del Estudiante de los Sistemas de Enseñanza-Aprendizaje Inteligentes.

MINSKY, M. 1975. A Framework for Representing Knowledge.

ORACLE. 2015a. *JavaDoc, java.lang.ClassLoader and java.ij.io.PluginClassLoader classes* [Online]. Available: https://imagej.nih.gov/ij/developer/loader-javadoc/ij/io/PluginClassLoader.html#PluginClassLoader%28java.lang.String%29 [Accessed].

ORACLE. 2015b. *JavaDoc, util package* [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html [Accessed].

ORACLE. 2015c. *JavaDoc, zip package* [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/zip/package-summary.html [Accessed].

PARR, T. 2007. The Definitive ANTLR Reference. *Building Domain-Specific Languages.*

PINO, D. R., GÓMEZ, G. A. & LABAJO, M. N. 2001. Introducción a la Inteligencia Artificial.

RAMOS, E. G. 2013. *Ingeniero de conocimiento automatizado para el diseño en ingeniería.*

RUSSELL, S. & NORVING, P. 2003. Artificial Intelligence A Modern Approach Third Edition. 3rd ed.

Anexos

Anexo 1: Cambios en la sintaxis de la escritura de las bases de conocimientos con respecto a la versión para Windows.

Delimitadores del Objeto Diseño.			
Delimitadores Anteriores		Delimitadores Agregados	
Inicio	Fin	Inicio	Fin
‘{’	‘}’	‘.OD’	‘.FOD’
‘%’	‘\n’	‘.DD’	‘.FDD’
‘$’	‘$’	‘.PID’	‘.FPID’
‘@’	‘@’	‘.PALD’	‘.FPALD’
‘!’	‘!’	‘.PREAD’	‘.FPREAD’
‘&’	‘&’	‘.RD’	‘.FRD’
‘?’	‘?’	‘.FED’	‘.FFED’
‘#’	‘#’	‘.POSTAD’	‘.FPOSTAD’

Delimitadores del Objeto Elemental u Objeto Parámetro.			
Delimitadores Anteriores		Delimitadores Agregados	
Inicio	Fin	Inicio	Fin
'['	']'	'.PD'	'.FPD'
'%'	'%'	'.DP'	'.FDP'
'@'	'@'	'.DOMP'	'.FDOMP.'
'$'	'$'	'.PAP'	'.FPAP'
'/'	'/'	'.PCP'	'.FPCP'
'^'	'^'	'.MAP'	'.FMAP'
'?'	'?'	'.PCPP'	'.FPCPP'
'#'	'#'	'.POSTAP'	'.FPOSTAP'
'!'	'!'	'.PREAP'	'.FPREAP'

Anexo 2: Ejemplo 1, Base de Conocimientos escrita usando la sintaxis para Windows.

```
{{
ESPESOR_PARED
%Cálculo del espesor mínimo requerido de las planchas de
las paredes de los tanques de almacenamiento atmosférico.
&
V<50000 or V=50000
D<66
Tp>20 and Tp<100
n<Cn or n=Cn
Pe<1000 or Pe=1000
Smin>3.4
&
}
[
V
%Capacidad del tanque en m3.
]
[
D
%Diámetro interior del tanque en m3.
$
Ht
$
/
->pow((4*V)/(3.14*(Ht-1)),0.5)
/
]
[
Ht
%Altura del tanque en m.
@
18
12
9
7.   5
@
/
V>30000 or V=30000->18
V=5000 or V>5000 and V<30000->12
V=1000 or V>1000 and V<5000->9
V<1000->7.5
/
```

```
]
[
Cn
%Cantidad de anillos que forman la pared.
$
Ht
$
/
->int(Ht/1.5)
/
]
[
L
%L¡quido que se almacena.
]
[
T
%Temperatura de almacenamiento en grados celsios.
]
[
Pe
%Peso específico del líquido en Kg/m3.
/
L="Agua"->1000
L="Fuel oil" ->920
L="Petroleo crudo" or L="Fuel oil marino" or L="Bunker"-
>860
L="Diesel" or L="Kerosina" or L="Combustible RT"->760
L="Gasolina" or L="Nafta"->680
L="Hexano"->550
/
]
[
H
%Altura de la columna líquida en m.
$
Ht
$
/
->Ht-1.5*(n-1)-W
/
]
[
n
%Anillo al cual se le calculará el espesor.
@
```

```
1
2
3
4
5
6
7
8
9
10
11
12
@
]
[
f
%Eficiencia de las soldaduras.
/
V<2000 or V>2000 and V<30000 and n>5->0.65
V>1000 and V<30000 and n>1 and n<6->0.9
V>20000 or V>1000 and V<30000 and n=1->1
/
]
[
Fp
%Límite de fluencia permisible del material en kg/cm2.
$
M
Tp
$
/
M="Acero BCt3"->((3/4)*(20-Tp)+140)*10
M="Acero 09G2C"->((23/80)*(20-Tp)+183)*10
/
]
[
M
%Material de las planchas del anillo.
/
V>30000 and n>7 or V>20000 and V<30000 and n>5 or V=30000
and n>5 or V>20000 and n>3 or V=20000 and n>3 or V<20000-
>"Acero BCt3"
V>30000 and n<8 or V>20000 and V< 30000 and n<6 or
V>20000 and V=30000 and n<6 or V>20000 and n<4 or V=20000
and n<4->"Acero 09G2C"
/
```

```
]
[
Tp
%Temperatura de la pared en grados celsios.
/
T>39->T-(T/8)
T<39 or T=39->T
/
]
[
S
%Espesor de las planchas en mm.
$
D
Ht
Pe
H
f
Fp
M
Tp
W
$
/
->(1.1*Pe*D*H)/(23*f*Fp)
/
]
[
Smin
%Espesor mínimo permitido en mm.
$
D
Ht
Pe
H
f
Fp
M
Tp
S
W
$
/
S>3.5 or S=3.5->S
S<3.5->3.5
/
```

```
]
[
Y
%Tiene techo flotante o pont el tanque?(S/N)
]
[
W
%Coeficiente
/
L<>"Agua" and Y="N"->1.4
L="Agua" and Y="N"->0.8
L<>"Agua" and Y="S"->1.8
/
]]
```

Anexo 3: Ejemplo 2, segmento de código usado en el SIAD New Syntax Converter para realizar el cambio de sintaxis.

```
public static void New_Syntax_Convert(File bcsFile)
{

try
{
FileReader fr = new FileReader(bcsFile);
BufferedReader br = new BufferedReader(fr);
StringBuilder sb = new
StringBuilder(bcsFile.getPath());sb.insert(bcsFile.getPat
h().length()-4, "_new");
FileWriter fw = new FileWriter(new File(sb.toString()));
PrintWriter pw = new PrintWriter(fw);
String lineBuffer;
boolean designObject = true, searchFunc = false;
Stack <String> toWrite = new Stack<>();
toWrite.push(" ");

while((lineBuffer = br.readLine())!=null)
{
StringTokenizer stk = new StringTokenizer(lineBuffer);

while(stk.hasMoreTokens())
{
String token = stk.nextToken();
System.out.println(token);

switch (token)
{
     case "{{":
          pw.write(".OD");
break;
case "}" :
          pw.write(".FOD");
          designObject = false;
     break;
     case "%" :
          if(designObject)
          {
               pw.write(".DD");
               toWrite.push(".FDD");
          }else
          {
```

```
                pw.write(".DP");
                toWrite.push(".FDP");
            }
        break;
        case "[" :
            pw.write(".PD");
            break;
        case "]" :
            pw.write(".FPD");
        break;
        case "]]" :
            pw.write(".FPD");
        break;
        case "$" :
            if(designObject)
                if(toWrite.peek().equals(".FPID"))
                    pw.write(toWrite.pop());
                else
                {
                    pw.write(".PID");
                    toWrite.push(".FPID");
                }
            else
                if(toWrite.peek().equals(".FPAP"))
                    pw.write(toWrite.pop());
else
{
                    pw.write(".PAP");
                    toWrite.push(".FPAP");
}
        break;
        case "@" :
            if(designObject)
                if(toWrite.peek().equals(".FPALD"))
            pw.write(toWrite.pop());
                else
        {
                    pw.write(".PALD");
                    toWrite.push(".FPALD");
                }
            else
        if(toWrite.peek().equals(".FDOMP"))
                    pw.write(toWrite.pop());
                else
                {
                    pw.write(".DOMP");
```

```
                    toWrite.push(".FDOMP");
               }
     break;
     case "!" :
          if(token.charAt(0) == '!')
               if(designObject)
                    if(toWrite.peek().equals(".FPREAD"))
                    {
                         pw.write(toWrite.pop());
                         searchFunc = false;
                    }else
                    {
                         pw.write(".PREAD");
                         searchFunc = true;
     toWrite.push(".FPREAD");
                    }
               else
                    if(toWrite.peek().equals(".FPREAP"))
                    {
                         pw.write(toWrite.pop());
searchFunc = false;
                    }else
                    {
                         pw.write(".PREAP");
                          searchFunc = true;
                         toWrite.push(".FPREAP");
   }
     break;
case "&" :
          if(toWrite.peek().equals(".FRD"))
{
     pw.write(toWrite.pop());
searchFunc = false;
}else
          {
               pw.write(".RD");
               searchFunc = true;
               toWrite.push(".FRD");
          }
     break;
     case "?" :
     if(designObject)
               if(toWrite.peek().equals(".FFED"))
               {
               pw.write(toWrite.pop());
                    searchFunc = false;
```

```
}else
                {
     pw.write(".FED");
                     searchFunc = true;
                    toWrite.push(".FFED");
                }
else
                    if(toWrite.peek().equals(".FPCPP"))
                    {
                         pw.write(toWrite.pop());
                         searchFunc = false;
     }else
                    {
                         pw.write(".PCPP");
                         searchFunc = true;
     toWrite.push(".FPCPP");
                    }
     break;
     case "#" :
          if(designObject)
               if(toWrite.peek().equals(".FPOSTAD"))
               {
     pw.write(toWrite.pop());
     searchFunc = false;
               }else
               {
                    pw.write(".POSTAD");
                    searchFunc = true;
                    toWrite.push(".FPOSTAD");
}
          else
               if(toWrite.peek().equals(".FPOSTAP"))
               {
                     pw.write(toWrite.pop());
                     searchFunc = false;
               }else
               {
                     pw.write(".POSTAP");
                     searchFunc = true;
                     toWrite.push(".FPOSTAP");
               }
     break;
     case "/" :
     if(toWrite.peek().equals(".FPCP"))
          {
     pw.write(toWrite.pop());
```

```
                searchFunc = false;
}else
{
     pw.write(".PCP");
     searchFunc = true;
     toWrite.push(".FPCP");
          }
     break;
     case "^" :
     if(toWrite.peek().equals(".FMAP"))
          {
     pw.write(toWrite.pop());
                searchFunc = false;
          }else
          {
                pw.write(".MAP");
                searchFunc = true;
                toWrite.push(".FMAP");
}
     break;
default:
          if(token.charAt(0) == '%')
          if(designObject)
                {
          pw.write(".DD ");
     pw.write(token.substring(1));
     toWrite.push(".FDD");
                }else
                {
     pw.write(".DP ");
          pw.write(token.substring(1));
                     toWrite.push(".FDP");
                }
          else
          {
     if (searchFunc)
                {

                if(token.contains("or"))
                token = token.replaceAll("or", "||");

                if(token.contains("and"))
                token = token.replaceAll("and", "&&");

                if(token.contains("pow"))
                token = token.replaceAll("pow", "POW");
```

```
if(token.contains("sin"))
token = token.replaceAll("sin", "SIN");

if(token.contains("cos"))
token = token.replaceAll("cos", "COS");

if(token.contains("asin"))
token = token.replaceAll("asin", "ASIN");
if(token.contains("acos"))
token = token.replaceAll("acos", "ACOS");

if(token.contains("tan"))
token = token.replaceAll("tan", "TAN");

if(token.contains("atan"))
token = token.replaceAll("atan", "ATAN");

if(token.contains("inv"))
token = token.replaceAll("inv", "INV");

if(token.contains("abs"))
token = token.replaceAll("abs", "ABS");

if(token.contains("exp"))
token = token.replaceAll("exp", "EXP");

if(token.contains("ln"))
token = token.replaceAll("ln", "LN");

if(token.contains("log"))
token = token.replaceAll("log", "LOG");

if(token.contains("int"))
token = token.replaceAll("int", "INT");

if(token.contains("np"))
token = token.replaceAll("np", "NP");

if(token.contains("ne"))
token = token.replaceAll("ne", "NE");

if(token.contains("lee"))
token = token.replaceAll("lee", "READ");

if(token.contains("min"))
```

```
                token = token.replaceAll("min", "MIN");

                if(token.contains("max"))
                token = token.replaceAll("max", "MAX");

                if(token.contains("<>"))
                token = token.replaceAll("<>", "!=");

                }

                pw.write(token);
           }
     break;
}
     pw.write(" ");
     pw.flush();
}

if(toWrite.peek().equals(" .FDD")
||toWrite.peek().equals(" .FDP"))
{
     pw.write(toWrite.pop());
     pw.write("\n");
     pw.flush();
}else
{
     pw.write("\n");
     pw.flush();
}
}
} catch (FileNotFoundException ex) { }
catch (IOException ex) { }
}
```

Anexo 4: Ejemplo 3, Base de Conocimientos usada en el ejemplo 1, transformada a la sintaxis de SIAD Multiplataforma usando el SIAD New Syntax Converter.

```
.OD
ESPESOR_PARED
.DD Cálculo del espesor mínimo requerido de las planchas de las paredes de los tanques de almacenamiento atmosférico .FDD
.RD
V<50000 || V=50000
D<66
Tp>20 && Tp<100
n<Cn || n=Cn
Pe<1000 || Pe=1000
SMIN>3.4
.FRD
.FOD
.PD
V
.DP Capacidad del tanque en m3.  .FDP
.FPD
.PD
D
.DP Diámetro interior del tanque en m3.  .FDP
.PAP
Ht
.FPAP
.PCP
->POW((4*V)/(3.14*(Ht-1)),0.5)
.FPCP
.FPD
.PD
Ht
.DP Altura del tanque en m.  .FDP
.DOMP
18
12
9
7.5
.FDOMP
.PCP
V>30000 || V=30000->18
V=5000 || V>5000 && V<30000->12
V=1000 || V>1000 && V<5000->9
```

```
V<1000->7.5
.FPCP
.FPD
.PD
Cn
.DP Cantidad de anillos que forman la pared.  .FDP
.PAP
Ht
.FPAP
.PCP
->INT(Ht/1.5)
.FPCP
.FPD
.PD
L
.DP Líquido que se almacena.  .FDP
.FPD
.PD
T
.DP  Temperatura  de  almacenamiento  en  grados  celsios.
.FDP
.FPD
.PD
Pe
.DP Peso específico del líquido en Kg/m3.  .FDP
.PCP
L="Agua"->1000
L="Fuel oil" ->920
L="Petroleo crudo" || L="Fuel oil marino" || L="Bunker"-
>860
L="Diesel" || L="KeroSINa" || L="Combustible RT"->760
L="Gasolina" || L="Nafta"->680
L="Hexano"->550
.FPCP
.FPD
.PD
H
.DP Altura de la columna líquida en m.  .FDP
.PAP
Ht
.FPAP
.PCP
->Ht-1.5*(n-1)-W
.FPCP
.FPD
.PD
```

```
n
.DP Anillo al cual se le calculará el espesor.  .FDP
.DOMP
1
2
3
4
5
6
7
8
9
10
11
12
.FDOMP
.FPD
.PD
f
.DP Eficiencia de las soldaduras.  .FDP
.PCP
V<2000 || V>2000 && V<30000 && n>5->0.65
V>1000 && V<30000 && n>1 && n<6->0.9
V>20000 || V>1000 && V<30000 && n=1->1
.FPCP
.FPD
.PD
Fp
.DP Límite de fluencia permisible del material en kg/cm2.
.FDP
.PAP
M
Tp
.FPAP
.PCP
M="Acero BCt3"->((3/4)*(20-Tp)+140)*10
M="Acero 09G2C"->((23/80)*(20-Tp)+183)*10
.FPCP
.FPD
.PD
M
.DP Material de las planchas del anillo.  .FDP
.PCP
V>30000 && n>7 || V>20000 && V<30000 && n>5 || V=30000 &&
n>5  ||  V>20000  &&  n>3  ||  V=20000  &&  n>3  ||  V<20000-
>"Acero BCt3"
```

```
V>30000 && n<8 || V>20000 && V< 30000 && n<6 || V>20000
&& V=30000 && n<6 || V>20000 && n<4 || V=20000 && n<4-
>"Acero 09G2C"
.FPCP
.FPD
.PD
Tp
.DP Temperatura de la pared en grados celsios.  .FDP
.PCP
T>39->T-(T/8)
T<39 || T=39->T
.FPCP
.FPD
.PD
S
.DP Espesor de las planchas en mm.  .FDP
.PAP
D
Ht
Pe
H
f
Fp
M
Tp
W
.FPAP
.PCP
->(1.1*Pe*D*H)/(23*f*Fp)
.FPCP
.FPD
.PD
Smin
.DP Espesor mínimo permitido en mm.  .FDP
.PAP
D
Ht
Pe
H
f
Fp
M
Tp
S
W
.FPAP
```

```
.PCP
S>3.5 || S=3.5->S
S<3.5->3.5
.FPCP
.FPD
.PD
Y
.DP Tiene techo flotante o pont el tanque?(S/N)  .FDP
.FPD
.PD
W
.DP Coeficiente  .FDP
.PCP
L!="Agua" && Y="N"->1.4
L="Agua" && Y="N"->0.8
L!="Agua" && Y="S"->1.8
.FPCP
.FPD
```

Anexo 5: Resultados de la conversión de sintaxis del ejemplo 1 al ejemplo 3

```
line 3:4 mismatched input 'Cálculo' expecting '.FDD'
.DD Cálculo del espesor mínimo requerido de las planchas
de las paredes de los tanques de almacenamiento
atmosférico.  .FDD

line 15:4 mismatched input 'Capacidad' expecting '.FDP'
.DP Capacidad del tanque en m3.  .FDP

line 19:4 mismatched input 'Diámetro' expecting '.FDP'
.DP Diámetro interior del tanque en m3.  .FDP

line 29:4 mismatched input 'Altura' expecting '.FDP'
.DP Altura del tanque en m.  .FDP

line 45:4 mismatched input 'Cantidad' expecting '.FDP'
.DP Cantidad de anillos que forman la pared.  .FDP

line 55:4 mismatched input 'Líquido' expecting '.FDP'
.DP Líquido que se almacena.  .FDP

line 59:4 mismatched input 'Temperatura' expecting '.FDP'
.DP Temperatura de almacenamiento en grados celsios.
.FDP

line 63:4 mismatched input 'Peso' expecting '.FDP'
.DP Peso específico del líquido en Kg/m3.  .FDP

line 63:43 mismatched input '.FDP' expecting {'/', '||',
'&&', '=', '!=', '<', '<=', '>', '>=', '->', '*', '+', '-
'}
.DP Peso específico del líquido en Kg/m3.  .FDP

line 65:1 extraneous input '=' expecting {'/', 'SIN',
'ASIN', 'COS', 'ACOS', 'TAN', 'ATAN', 'INV', 'ABS',
'EXP', 'LN', 'LOG', 'INT', 'FRAC', 'POW', 'NE', 'NP',
'READ', 'READFCH', 'READFLN', 'MIN', 'MAX', 'CONSULT',
TK_NEG, '->', '*->', '*', '+', '-', '(', ID, BOOLEAN,
STRING, INT, DOUBLE}
L="Agua"->1000

line 71:0 mismatched input '.FPCP' expecting {'/', 'SIN',
'ASIN', 'COS', 'ACOS', 'TAN', 'ATAN', 'INV', 'ABS',
'EXP', 'LN', 'LOG', 'INT', 'FRAC', 'POW', 'NE', 'NP',
'READ', 'READFCH', 'READFLN', 'MIN', 'MAX', 'CONSULT',
```

```
TK_NEG, '->', '*->', '*', '+', '-', '(', ID, BOOLEAN,
STRING, INT, DOUBLE}
.FPCP

line 75:4 mismatched input 'Altura' expecting '.FDP'
.DP Altura de la columna líquida en m.  .FDP

line 85:4 mismatched input 'Anillo' expecting '.FDP'
.DP Anillo al cual se le calculará el espesor.  .FDP

line 103:4 mismatched input 'Eficiencia' expecting '.FDP'
.DP Eficiencia de las soldaduras.  .FDP

line 112:4 mismatched input 'Límite' expecting '.FDP'
.DP Límite de fluencia permisible del material en kg/cm2.
.FDP

line 112:59 mismatched input '.FDP' expecting {'/', '||',
'&&', '=', '!=', '<', '<=', '>', '>=', '->', '*', '+', '-
'}
.DP Limite de fluencia permisible del material en kg/cm2.
.FDP

line 116:0 mismatched input '.FPAP' expecting {'/', '||',
'&&', '=', '!=', '<', '<=', '>', '>=', '->', '*', '+', '-
'}
.FPAP

line 118:1 extraneous input '=' expecting {'/', 'SIN',
'ASIN', 'COS', 'ACOS', 'TAN', 'ATAN', 'INV', 'ABS',
'EXP', 'LN', 'LOG', 'INT', 'FRAC', 'POW', 'NE', 'NP',
'READ', 'READFCH', 'READFLN', 'MIN', 'MAX', 'CONSULT',
TK_NEG, '->', '*->', '*', '+', '-', '(', ID, BOOLEAN,
STRING, INT, DOUBLE}
M="Acero BCt3"->((3/4)*(20-Tp)+140)*10

line 120:0 extraneous input '.FPCP' expecting {'/',
'SIN', 'ASIN', 'COS', 'ACOS', 'TAN', 'ATAN', 'INV',
'ABS', 'EXP', 'LN', 'LOG', 'INT', 'FRAC', 'POW', 'NE',
'NP', 'READ', 'READFCH', 'READFLN', 'MIN', 'MAX',
'CONSULT', TK_NEG, '->', '*->', '*', '+', '-', '(', ID,
BOOLEAN, STRING, INT, DOUBLE}
.FPCP

line 124:4 mismatched input 'Material' expecting '.FDP'
.DP Material de las planchas del anillo.  .FDP
```

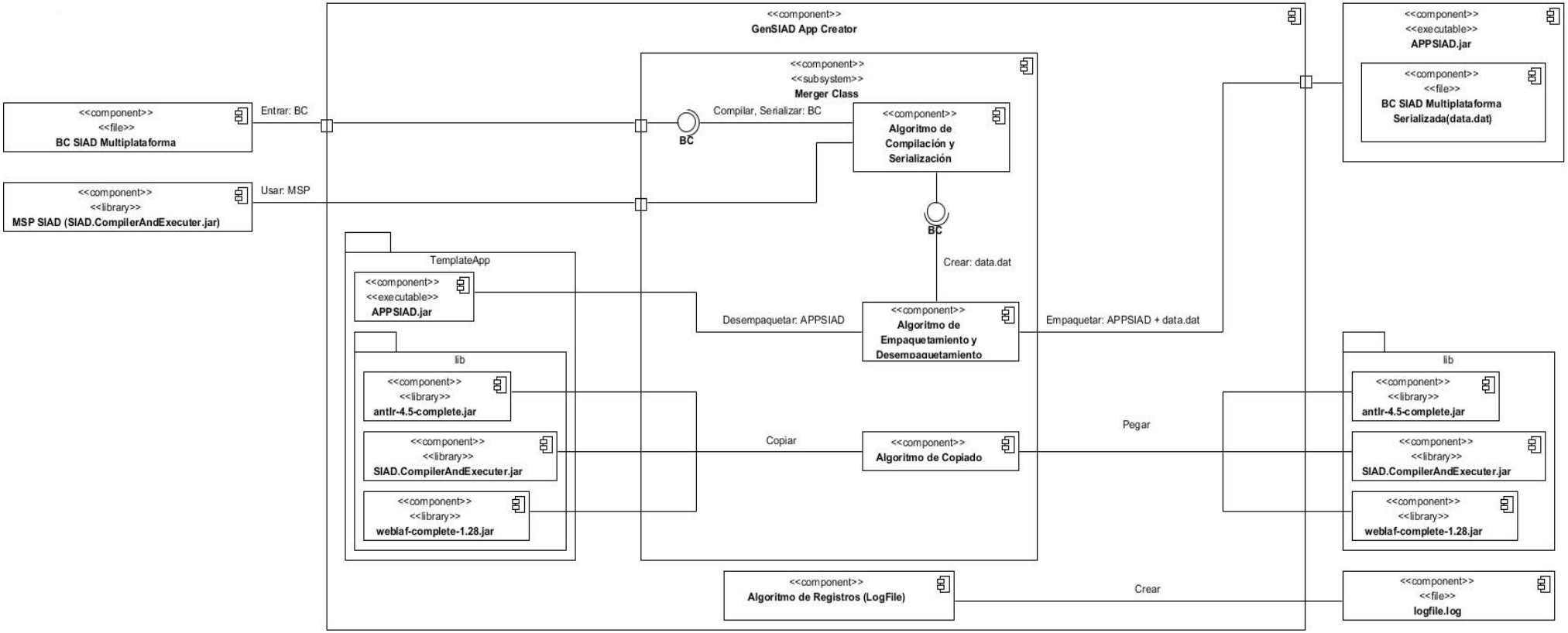
```
line 132:4 mismatched input 'Temperatura' expecting
'.FDP'
.DP Temperatura de la pared en grados celsios.  .FDP

line 140:4 mismatched input 'Espesor' expecting '.FDP'
.DP Espesor de las planchas en mm.  .FDP

line 158:4 mismatched input 'Espesor' expecting '.FDP'
.DP Espesor mínimo permitido en mm.  .FDP

line 178:4 mismatched input 'Tiene' expecting '.FDP'
.DP Tiene techo flotante o pont el tanque?(S/N)  .FDP

line 178:46 mismatched input ')' expecting {'/', '||',
'&&', '=', '!=', '<', '<=', '>', '>=', '->', '*', '+', '-
'}
.DP Tiene techo flotante o pont el tanque?(S/N)  .FDP

line 182:4 extraneous input 'Coeficiente' expecting
'.FDP'
.DP Coeficiente  .FDP

EXCEPTION -> [null]
EXCEPTION -> [null]
EXCEPTION -> [null]
EXCEPTION -> [null]
EXCEPTION -> [null]
```

Anexo 6: Ejemplo 4, segmento de código empleado en la implementación de los mensajes de depuración del APPSIAD

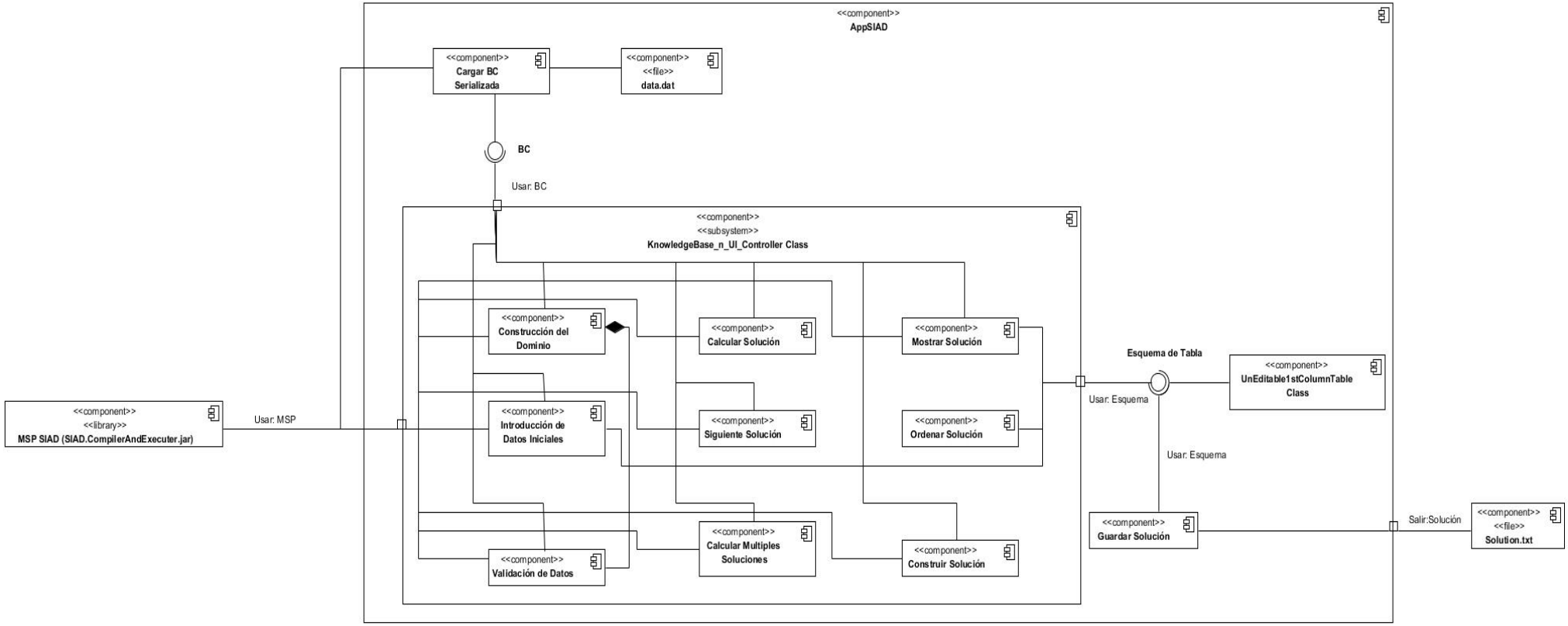
```
private class Thread_Compiler_Console extends Thread
{
  @Override
  public void run()
  {
    File tmpFile = null;
    PrintStream pout = null;
    FileInputStream finput = null;

    try
    {
      tmpFile = File.createTempFile("gensiad",null);
      pout = new PrintStream(tmpFile);
      finput = new FileInputStream(tmpFile);
    } catch (IOException ex) { }

    System.setOut(pout);
    System.setErr(pout);
    byte buffer [];
    String sdata;
    int readed = 0;

    while(true)
    {
      buffer = new byte [4096];
      sdata = "";
```

```
        try
        {
          thread_tiper.sleep(1000);
while((readed=finput.read(buffer,0,buffer.length))!= -1)
  for(int i=0;i<readed;i++)
    sdata += (char)buffer[i];

           if(!sdata.isEmpty())
           {
             sdata = sdata.replaceAll("\n", "   ");
             jlabel_Info_right.setText(sdata);
             sleep(5000);
             jlabel_Info_right.setText("");
           }
        } catch (InterruptedException ignore) { }
          catch (IOException ex) { }
     }
  }
}
```

Anexo 7: Diagrama de componentes del GenSIAD App Creator



Anexo 8: Diagrama de componentes del APPSIAD



Anexo 9: Diagrama de clases representando los métodos más significativos para la propuesta de herramientas a integrar al SIAD en forma de plugins