Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Cálculo de propiedades termodinámicas para mezclas multicomponente en el entorno de programación Android

Autor:

Salamanca Farto, Carlos

Tutor(es):

García Serna, Juan

Cabeza Sánchez, Álvaro

Ingeniería Química y Tecnología del Medio Ambiente

Valladolid, Septiembre, 2014

Si A es el éxito en la vida, entonces A es igual a x más y más z. El trabajo es la x; la suerte es la y; y la z mantiene tu boca cerrada.

Albert Einstein

Agradecimientos

En primer lugar al tutor, Juan García Serna, por facilitarme la posibilidad de realizar mi Trabajo de Fin de Grado acerca de la programación en Android, ya que es un tema que despertó mi interés hace tiempo.

También a Álvaro Cabeza, por la gran ayuda que me ha ofrecido tanto en los aspectos de la programación como en la comprensión de conceptos relacionados con la química y la termodinámica.

A mis compañeros de estudios, que nunca tuvieron duda de que lograría alcanzar mis objetivos, y que han supuesto un pilar fundamental en mi trabajo.

Por supuesto, a mi familia y mis amigos, que siempre han sido un gran apoyo, y que creyeron en mi capacidad para realizar mis estudios.

Muchas gracias a todos.

Resumen

Este trabajo consiste en la realización de una aplicación sujeta a una licencia libre GNU para el cálculo de propiedades termodinámicas en el entorno Android.

La aplicación constituye una herramienta de cálculo de propiedades energéticas y equilibrio para mezclas miscibles, en condiciones alejadas del punto crítico y formadas por sustancias no muy dispares químicamente.

Se hace una descripción del entorno de programación así como cuáles son sus distintas versiones hasta la fecha. Además, se explica qué son las ecuaciones de estado, su uso y por qué se han seleccionado determinadas ecuaciones y metodologías de resolución del equilibrio líquido-vapor. Implementación de tareas asíncronas, ecuación de volúmenes trasladados y base de datos en SQL como mejora a la versión anterior, presentando así una herramienta distinta a las que podemos encontrar actualmente en la PLAY STORE.

Finalmente, se desarrollan una serie de ejemplos para demostrar su funcionamiento al igual que unos manuales de uso de las aplicaciones.

Palabras Clave

Android, SQL, Equilibrio de Fases, Tareas Asíncronas, Propiedades Termodinámicas.

Abstract

This Project consists on the completion of an application subject to a free GNU license to calculate thermodynamic properties in Android programming environment.

The app is a tool to calculate energetic and balance properties for miscible mixtures, in conditions remotes of critical point and formed by similar substances.

For the programming environment is made thus a description of it as what are its different versions so far. Similarly explains what are the equations of State, its use and why he selected certain equations of State and resolution of the vapor-liquid equilibrium methodologies. Use of AsyncTask, SQL database and volume translated equation to complete de app's last version, obtaining an original application that was not available on PLAY STORE.

Finally, a number of examples are developed to demonstrate its operation as well as a few manuals to facilitate the use of the applications.

Keywords

Android, SQL, Phases Equilibrium, AsyncTask, Transport Properties.

Índice

Índice de figuras

Índice de Tablas

Índice de diagramas

1. Introducción

1.1. Antecedentes

El desarrollo de este Trabajo de Fin de Grado está basado en el Proyecto de Fin de Carrera de Álvaro Cabeza Sánchez, titulado 'Cálculo de propiedades termodinámicas y simulación de reactores en tiempo real en el entorno de programación Android'. Sobre su aplicación 'Calculator of Termodynamic Propierties' se han implementado una serie de mejoras y añadido nuevas funcionalidades, que serán explicadas posteriormente en el apartado de 'Objetivos'. (Cabeza Sánchez, 2012).

1.2. Introducción al entorno Android

Android es un entorno de programación para dispositivos móviles y tablets que, gracias a los esfuerzos de Google y Open Handset Alliance (unión de asociaciones que buscan un mercado de telefonía móvil más abierto y libre) es una de las principales plataformas utilizadas hoy en día (Ableson, Sen, King, & Ortiz, 2012).

Es un software de código abierto que incluye un sistema operativo (basado en Linux), un middleware (software que permite la interactuación de la aplicación con otras) y las bibliotecas API (Meier, 2009).

Por otro lado y para evitar confusiones muy generalizadas cabe reseñar que Android no es:

- Una aplicación de Java ME (la parte de Java orientada a la programación de dispositivos móviles). Si bien es cierto que usa el lenguaje de Java ME, sus aplicaciones no pueden ejecutarse en su máquina virtual.

- Una parte de Open Handset Alliance (OMA) o de Linux Phone Standards Forum (LiPS).

- La respuesta de Google al iPhone. Esta afirmación se fundamenta en que mientras que el software del iPhone es de código cerrado y perteneciente a Apple, Android es de código abierto y puede ser usado por cualquier medio.

La arquitectura de Android es la siguiente (Pablo Basanta Val, 2012) :

- Aplicaciones: Este nivel contiene, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

- Framework de Aplicaciones: Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework, representado por este nivel. Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

    - Activity Manager: Conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.

    - Window Manager: Gestiona las ventanas de las aplicaciones y utiliza la librería Surface Manager.

    - Telephone Manager: Incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).

    - Content Provider: Permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.

    - View System: Proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, "check-boxes", tamaño de ventanas, control de las interfaces mediante teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.

    - Location Manager: Posibilita a las aplicaciones la obtención de información de localización y posicionamiento.

    - Notification Manager: Mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje

recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada Intent, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.

- XMPP Service: Colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.

- Librerías: La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android. Entre las librerías más importantes ubicadas aquí, se pueden encontrar las siguientes:

  - Librería libc: Incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.

  - Librería Surface Manager: Es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.

  - OpenGL/SL y SGL: Representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android. OpenGL/SL maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, SGL proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.

  - Librería Media Libraries: Proporciona todos los códec necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.).

  - FreeType: Permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.

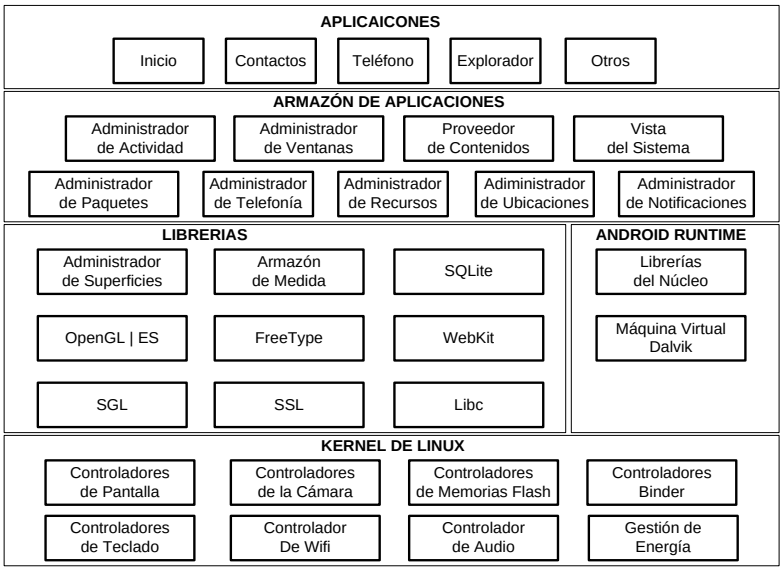
- Librería SSL: Posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.

- Librería SQLite: Creación y gestión de bases de datos relacionales.

- Librería WebKit: Proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.

- Tiempo de ejecución de Android: Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases Java y la máquina virtual Dalvik.

➢ Núcleo Linux: Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.



:

Figura 1: Arquitectura de Android (Pablo Basanta Val, 2012).

Una vez que Android ha sido definido se procede a exponer qué elementos básicos componen una aplicación (Lequerica, 2011):

- Activity:

Las actividades (activities) son el componente principal de la interfaz gráfica de una aplicación. Su finalidad es interactuar con el usuario y una aplicación puede tener cualquier número de ellas.

Una vez iniciada, la actividad pasa por una serie de estados:

- Activa: cuando es plenamente visible y el usuario interactúa sobre ella.
- En pausa: cuando es parcialmente visible y el usuario no puede interactuar sobre ella.
- Parada: cuando no es visible.

Durante los cambios de estado la plataforma invoca una serie de métodos:

- OnCreate: llamado al crear la actividad.
- OnStart: llamado cuando la actividad está a punto de ser visible.
- OnPause: llamado cuando la actividad se va a poner en background.
- OnResume: llamado cuando se recupera una actividad en background y el usuario va a empezar a interactuar con ella.
- OnStop: llamado cuando la actividad no es visible.
- OnRestart: llamado cuando la actividad se vuelve a hacer visible.
- OnDestroy: llamado antes de eliminar la actividad y su contenido.
- OnSaveInstanceState: sirve para guardar los datos ante un cambio en la actividad.

Este conjunto de métodos y los estados posibles conforman el ciclo de vida de una activity (diagrama 1).

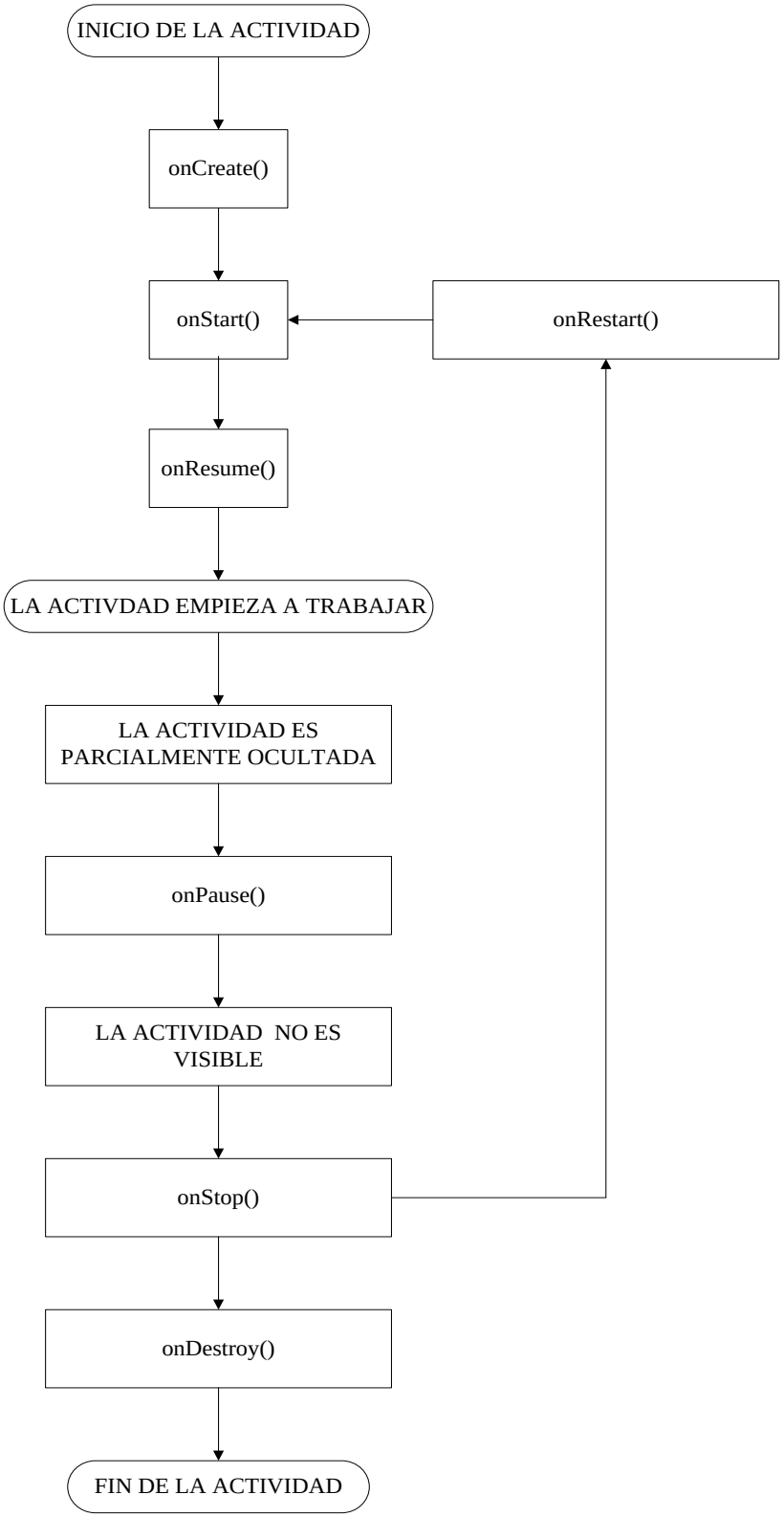


Diagrama 1: ciclo de vida de una Activity.

- View:

Las vistas (view) son los componentes básicos con los que se construye la interfaz gráfica de la aplicación. Son el principal nexo de unión entre las Activity y el usuario.

- Threads:

Un thread (o hilo) no es un componente en sí de Android, sino una forma de ejecutar tareas que requieran una gran cantidad de memoria sin bloquear la interactuación del usuario con la aplicación.

Se basa en dividir el proceso en dos o más subprocesos: el hilo principal (main) que es el que interactúa con el usuario y los hilos secundarios que envían sus resultados al principal.

Aunque el empleo de hilos puede resultar de gran utilidad, en ciertos casos se ha optado por emplear tareas asíncronas, como se puede ver a lo largo del trabajo.

- Service

Los servicios (service) son componentes sin interfaz gráfica que se ejecutan en segundo plano. Esto no significa que sean un sustituto de los thread, ya que todas las operaciones asignadas a ellos se ejecutan en el hilo principal.

Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (p.ej. actividades) si se necesita en algún momento la interacción con del usuario.

Su principal característica es que el método onDestroy de la actividad que los ha llamado no los finaliza y siguen ejecutándose después del fin de ésta.

- Content Provider

Ya definido en la arquitectura.

- Broadcast Receiver

Un broadcast receiver es un componente cuya función es detectar y reaccionar a mensajes o eventos globales generados por el sistema (Batería baja, Tarjeta SD insertada…) o por otras aplicaciones.

- Widget

Los widgets son elementos visuales que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas.

- Intent

Un intent es el elemento básico de comunicación entre los distintos componentes Android. Son los mensajes enviados entre los distintos componentes de una aplicación o entre ellas.

Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

Finalmente, se presentan las distintas versiones de Android (Attribution, 2013; Wikimedia Foundation, 2013):

Versión de Android	Nivel API	Nombre
4.4.3	19	KITKAT
4.4.2		
4.4.1		
4.3	18	JELLY_BEAN_MR2
4.2.2	17	JELLY_BEAN_MR1
4.2		
4.1.1	16	JELLY_BEAN
4.1		
4.0.4	15	ICE_CREAM_SANDWICH_MR1
4.0.3		
4.0.2	14	ICE_CREAM_SANDWICH
4.0.1		
4.0		
3.2	13	HONEYCOMB MR2
3.1.x	12	HONEYCOMB MR1
3.0.x	11	HONEYCOMB

2.3.4	10	GINGEBREAD_MR1
2.3.3		
2.3.2	9	GINGEBREAD
2.3.1		
2.3		
2.2.x	8	FROYO
2.1x	7	ECLAIR_MR1
2.0.1	6	ECLAIR_0_1
2.0	5	ECLAIR
1.6	4	DONUT
1.5	3	CUPCAKE
1.1	2	BASE_1_1
1.0	1	BASE

Tabla 1: Versiones de Android

El paquete de aplicaciones desarrollado en este proyecto solo es válido para los dispositivos cuya API este entre la 8 y la 20.

1.3. Ecuaciones de estado

Una ecuación de estado es una expresión matemática que explica el estado de agregación de la materia como una relación matemática entre la temperatura, la presión, el volumen, la densidad, la energía interna y otras funciones de estado asociadas con la materia. Las ecuaciones de estado son útiles para describir las propiedades de los fluidos y sus mezclas. El uso más importante de una ecuación de estado es para predecir el estado de gases (CENGEL & BOLES, 2001; Michelsen & Mollerup., 2007).

Una de las ecuaciones de estado más simples para este propósito es la ecuación de estado del gas ideal, que explica el comportamiento de los gases a bajas presiones y temperaturas mayores a la temperatura crítica. Sin

embargo, esta ecuación pierde mucha exactitud a altas presiones y bajas temperaturas, y no es capaz de predecir la condensación de gas en líquido (pues solo da una raíz). Por ello, existe una serie de ecuaciones de estado más precisas para gases y líquidos. Entre las ecuaciones de estado más empleadas sobresalen las ecuaciones cúbicas de estado. De ellas, las más conocidas y utilizadas son la ecuación de Peng-Robinson (PR) y la ecuación de Redlich-Kwong-Soave (RKS o SRK).

Hasta ahora no se ha encontrado ninguna ecuación de estado que prediga correctamente el comportamiento de todas las sustancias en todas las condiciones.

El porqué de utilizar ecuaciones cúbicas es que son la expresión polinómica más sencilla capaz de explicar el comportamiento de líquidos y gases (3 raíces. 1 real y dos imaginarias implican que existe una sola fase, 3 reales indican que existen dos fases, la de menor valor se corresponde a la liquida y la de mayor a la gas, la intermedia no tiene significado físico). Dado un diagrama P-V en el que se representa una isoterma puede comprobarse esta afirmación:

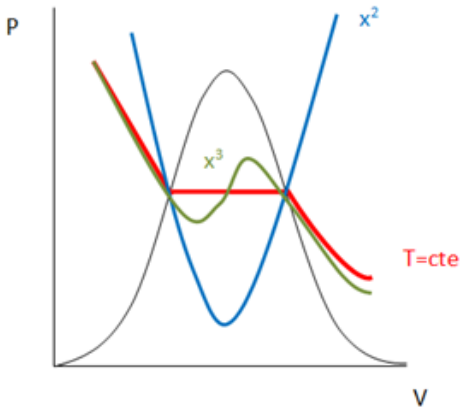


Figura 2: Por qué de las ecuaciones de tercer grado

En este trabajo se utilizarán las de ecuaciones de SRK y PR por ser las más extendidas y la de Van der Waals (VdW) por su uso académico. Todas ellas tienen una complejidad matemática aceptable para poder realizar los cálculos con programas de cálculo sencillo, e incluso manualmente en algunos casos. Esto facilitará su uso en el ámbito académico.

1. VdW:

La ecuación de Van der Waals es una ecuación de estado desarrollada en 1873 por Johannes van der Waals Diderik a partir de la ecuación de los gases ideales. Su modificación fue la eliminación de la suposición de que el tamaño de las partículas del fluido era despreciable y que no interactuaban entre sí (Bruce E. Poling, 2001; Michelsen & Mollerup., 2007).

La formulación matemática es:

$$\left(P + n^2 \cdot \frac{a}{V^2}\right) \cdot (V - n \cdot b) = n \cdot R \cdot T \qquad (1)$$

Dónde: P es la presión en atm, V es el volumen el litros del recipiente, a es el parámetro que introduce el efecto de la interacción entre partículas, b o covolumen es el volumen disponible de un mol de partículas, n es el número de moles, R es la constante de los gases ideales (0.082 $atm \cdot l \cdot mol^{-1} \cdot K^{-1}$) y T es la temperatura en K.

A pesar de estas modificaciones, la ecuación solo da resultados aceptables para gases no polares y lejos de sus condiciones de saturación (para líquidos en ningún caso). Por ello, no es válida para diseño.

2. SRK:

La ecuación SRK es una ecuación deducida en 1972 por Soave mediante la adición del término α dependiente del factor acéntrico (medida de la diferencia entre la forma de la partícula y la esfera o forma ideal) a la ecuación RK (obtenida, a su vez, mediante una mejora de la ecuación de VdW) (Bruce E. Poling, 2001; Michelsen & Mollerup., 2007).

El resultado fue una expresión capaz de representar bien la fase gas y aceptablemente la líquida, tanto en saturación como en otros estados:

$$\left(P + \alpha \cdot \frac{a}{V \cdot (V+b)}\right) \cdot (V - b) = R \cdot T \qquad (2)$$

$$a = 0.427480 \cdot R^2 \cdot T_c^2 \cdot P_c^{-1} \qquad (3)$$

$$b = 0.086640 \cdot R^2 \cdot T_c^2 \cdot P_c^{-1} \qquad (4)$$

$$\alpha = \left[1 + (0.8508 + 1.5517 \cdot w - 0.15613 \cdot w^2) \cdot \left(1 - T_r^{\frac{1}{2}}\right)\right]^2 \quad (5)$$

$$\alpha_{Hidrógeno} = 1.202 \cdot e^{-0.30288 \cdot T_r} \quad (6)$$

Dónde: P es la presión en atm, V es el volumen en litros del recipiente, R es la constante de los gases ideales (8.31 $J \cdot mol^{-1} \cdot K^{-1}$), T es la temperatura en K, Tc la temperatura crítica en K, Pc la presión crítica en atm, w el factor acéntrico y Tr la temperatura reducida.

3. PR:

La ecuación PR fue también desarrollada a partir de la de VdW (en 1976), por lo que da resultados muy parecidos a los de la ecuación SRK. La principal diferencia radica en que es más precisa para líquidos no polares, ya que fue desarrollada para la industria petroquímica (Bruce E. Poling, 2001; Michelsen & Mollerup., 2007).

La ecuación es:

$$\left(P + \alpha \cdot \frac{a}{V \cdot (V+b) + b \cdot (V-b)}\right) \cdot (V - b) = R \cdot T \quad (7)$$

$$a = 0.45724 \cdot R^2 \cdot T_c^2 \cdot P_c^{-1} \quad (8)$$

$$b = 0.07780 \cdot R \cdot T_c \cdot P_c^{-1} \quad (9)$$

$$\alpha = [1 + (0.37464 + 1.54226 \cdot w - 0.26992 \cdot w^2) \cdot (1 - T_r^{1/2})]^2 \quad (10)$$

Dónde: P es la presión en atm, V es el volumen en litros del recipiente, R es la constante de los gases ideales (8.31 $J \cdot mol^{-1} \cdot K^{-1}$), T es la temperatura en K, Tc la temperatura crítica en K, Pc la presión crítica en atm, w el factor acéntrico y Tr la temperatura reducida.

4. VTPR:

Es un método de estimación para el cálculo del equilibrio entre fases de mezclas de componentes químicos. El principal objetivo del desarrollo de este método es la capacidad de estimar las propiedades de mezclas que contengan componentes supercríticos. La ecuación de Peng-Robinson de volumen trasladado está basada en una combinación de la ecuación de estado de Peng-Robinson con una serie de ecuaciones establecidas por Péneloux en 1982. Se denomina 'de volumen trasladado' porque a la hora de calcular el factor de compresibilidad (Z), que es proporcional al volumen, se le añade un parámetro 't' que hace que el valor de dicha variable cambie, o se desplace

Las ecuaciones son:

$$P = \frac{R \cdot T}{v + t - b} - \frac{a \cdot \alpha(T)}{v^2 + 2 \cdot t \cdot v - t^2} \quad (11)$$

$$t = t_0 + (t_c - t_0) \cdot e^{(\beta \cdot |1 - T_r|)} \quad (12)$$

$$t_0 = \frac{R \cdot T_c}{p_c} \cdot (0.01447 + 0.0675 \cdot \omega - 0.08485 \cdot \omega^2 + 0.0673 \cdot \omega^3 - 0.0173 \cdot \omega^4) \quad (13)$$

$$t_c = \frac{R \cdot T_c}{p_c} \cdot (0.3074 - (0.289 - 0.0701 \cdot \omega - 0.0207 \cdot \omega^2)) \quad (13)$$

$$\beta = -10.2447 - 28.6312 \cdot \omega \quad (14)$$

Dónde: P es la presión en atm, V es el volumen en litros del recipiente, R es la constante los gases ideales (8.31 $J \cdot mol^{-1} \cdot K^{-1}$), T es la temperatura en K, Tr es la temperatura reducida, y t es el parámetro de desplazamiento incorporado.

Por otra parte, la regla de mezclas que calcula los parámetros a y b de la ecuación de estado es la misma que para las ecuaciones de Peng-Robinson.

Función alfa

Antes se ha mencionado un factor alfa introducido por Soave (y presente también en la ecuación de PR) como modificación a la ecuación de RK (o VdW).

Este factor es realmente una función empírica que depende de la temperatura y el factor acéntrico que mejora los resultados de las ecuaciones de estado en saturación. Dando valores aceptables para diseño con sustancias no polares o poco polares (Bruce E. Poling, 2001; Michelsen & Mollerup., 2007).

Reglas de mezcla y parámetros de interacción binaria

A la hora de aplicar las ecuaciones de estado a mezclas y no a componentes puros se han de aplicar unas reglas de mezcla diferente a las reglas de Kay (la propiedad de la mezcla es la suma ponderada de las propiedades de cada componente) (Bruce E. Poling, 2001; Michelsen & Mollerup., 2007):

$$a = \sum_i \sum_j x_i \cdot x_j \cdot a_{ij} \quad (15)$$

$$a_{ii} = a_i \quad (16)$$

$$a_{ij} = \sqrt{a_i \cdot a_j} \quad (17)$$

$$b = \sum_i \sum_j x_i \cdot x_j \cdot b_{ij} \quad (18)$$

$$b = b_{ji} = b_{ij} \quad (19)$$

$$b_{ij} = \frac{b_i + b_j}{b} = \sum_i x_i \cdot b_i \quad (20)$$

Otro paso más a la hora de mejorar los resultados es la introducción de parámetros de interacción binaria (k_{ij}), los cuales no son más que un añadido matemático a los coeficientes cruzados cuyo valor se obtiene experimentalmente.

$$a_{ij} = \left(1 - k_{ij}\right) \cdot \sqrt{a_i \cdot a_j} \quad (21)$$

$$a = \sum_i \sum_j \left(1 - k_{ij}\right) \cdot x_i \cdot x_j \cdot a_{ij} \quad (22)$$

Existen dos tipos de parámetros de interacción:

- 1PVDW (un parámetro ajustable): en los cuales el ajuste con los datos experimentales se hace con las ecuaciones antes expuestas.

- 2PVDW (dos parámetros ajustables): en este caso, el k_{ij} se define, a mayores, como:

$$k_{ij} = K_{ij} \cdot x_i + K_{ji} \cdot x_j \quad (23)$$

Donde los parámetros a ajustar son K_{ij} y K_{ji}.

La ventaja de los 1PVDW (que son los utilizados en este trabajo) es que son más sencillos de obtener, tienen menor dependencia con la temperatura y dan un comportamiento bueno para hidrocarburos y regular para inorgánicos. La desventajas es que en caso de grandes diferencias (alcohol + alcano, agua + alcano…) no dan resultados aceptables. Por contra, los 2PVDW son capaces de generar buenos resultados incluso para esos sistemas tan dispares, pero a cambio de una enorme dependencia con la temperatura y dificultando el proceso de ajuste.

2. Objetivos

El objeto de este proyecto consiste en realizar una aplicación Android (Calculator of Thermodynamic properties) con el fin de proporcionar una herramienta orientada a los estudiantes de ingeniería, o cualquier persona interesada en ello, que realice el cálculo de propiedades y estado físico (líquido o vapor), represente las gráficas correspondientes a las relaciones presión-temperatura de los componentes que se deseen (componentes puros), y permita al usuario acceder a una base de datos con los componentes disponibles, que él mismo puede modificar o actualizar a su antojo. Puesto que la orientación del programa es ser para uso académico se ha hecho en inglés y está sujeto a una licencia libre GNU.

De forma más específica, las capacidades que abarcará la aplicación serán:

- Cálculo de equilibrios fase líquida-fase gaseosa y de propiedades termodinámicas, empleando las ecuaciones de estado más recurrentes (Van del Waals, Peng-Robinson y Soave-Redlich-Kwong), así como la ecuación de Peng-Robinson para volúmenes trasladados, que mejora el cálculo de densidades.

- Acceso a la explicación teórica de cada uno de los cálculos realizados, tanto para las ecuaciones de estado, como para el equilibrio de fases o las propiedades energéticas.

- Conexión con una base de datos en SQL, en la que se encuentran definidas las características de 78 componentes diferentes (ver Anexo III), con los que el usuario podrá trabajar. Además, la aplicación permite la incorporación de nuevos componentes a la base de datos, solicitando para ello los valores necesarios de cada una de sus diferentes características para el correcto cálculo de propiedades.

- Obtención de gráficas 2D de los diagramas de fase para componentes puros, representando Temperatura vs Presión y Presión vs Temperatura.

- Además de todo lo nombrado, se mostrarán tanto la licencia como una breve ayuda que facilitará los primeros pasos a los usuarios que utilicen la aplicación.

3. Desarrollo

3.1. Restricciones de la aplicación

Debido a la extensión del objetivo de esta aplicación, y al amplio abanico de posibilidades que ofrece el cálculo de propiedades termodinámicas, se hacen las siguientes simplificaciones para que la aplicación sea válida para el uso académico:

- No se resuelve el equilibrio gas no condensable-líquido.
- Se asume que la fase líquida se comporta como un líquido ideal, sin considerar inmisicibilidades.
- No asegura la resolución del estado supercrítico.
- Válido sólo para mezclas de hasta 6 componentes de una base de datos inicial de 78 sustancias (ver Anexo III), ampliable a tantas como se deseen, siempre y cuando se especifiquen los datos del componente solicitados por la aplicación.

Fijadas estas restricciones se procede a explicar cómo realiza los cálculos la aplicación.

3.2. Cálculos realizados

3.2.1. Cálculo de densidades

La propiedad inmediata que puede calcularse con las ecuaciones de estado es la densidad.

Al resolver la ecuación se obtendrán tres volúmenes diferentes, el más grande corresponde al vapor y el más bajo al líquido, por lo que, la densidad, sería (Bruce E. Poling 2001; Michelsen and Mollerup. 2007):

$$\rho_{real} = \frac{1}{\mathrm{V_{real}}} \qquad (24)$$

La resolución se realiza mediante el método de Newton-Raphson (consultar anexo I) puesto que al ser un programa de orientación académica se busca que los resultados sean fácilmente contrastados por el estudiante:

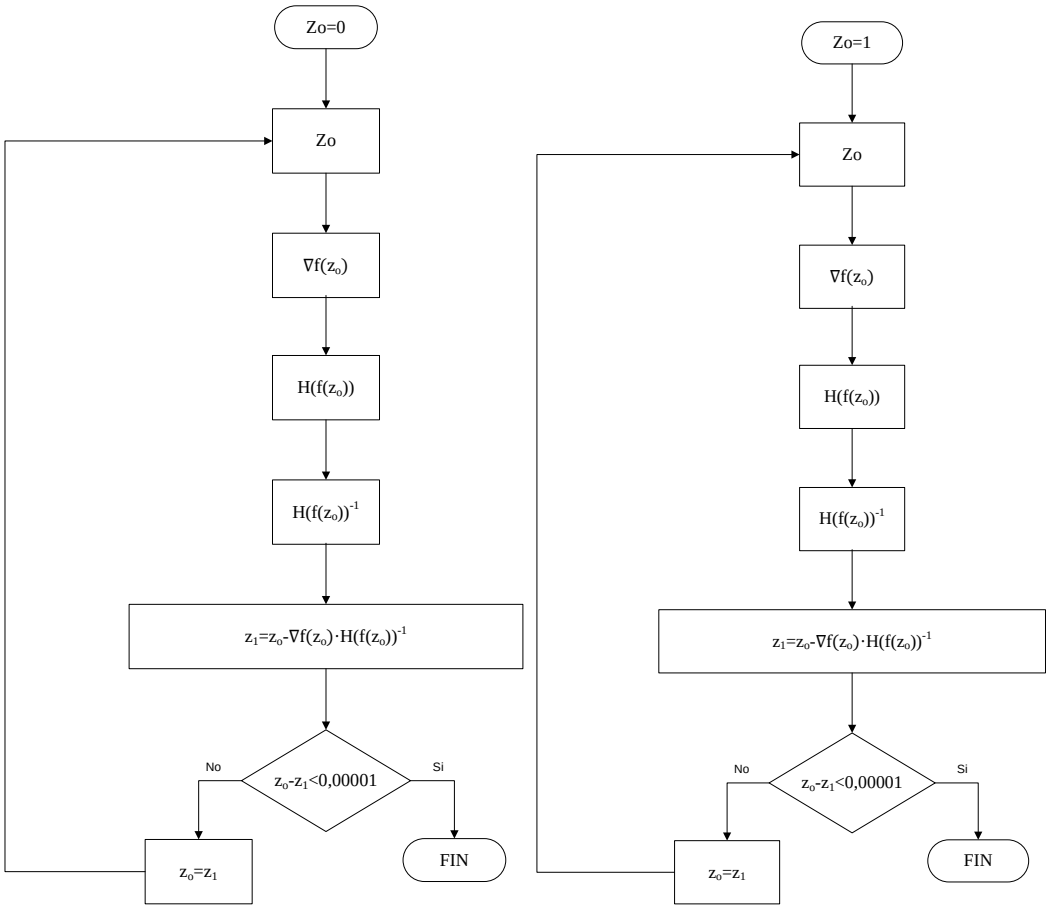


Diagrama 2: Algoritmo de obtención del Z del líquido (izquierda) y del Z del gas (derecha)

En el caso en el que las ecuaciones estén en función del coeficiente de compresibilidad (Z) se dispondrá de tres valores del coeficiente, cuya definición es:

$$Z = \frac{V_{real}}{V_{ideal}} \tag{25}$$

Y recordando que el volumen molar ideal es:

$$V_{ideal} = \frac{R \cdot T}{P} \tag{26}$$

Se llega a que la densidad molar real (ρ_{real}) se obtiene mediante la siguiente fórmula:

$$\rho_{real} = \frac{P}{R \cdot T \cdot Z} \tag{27}$$

3.2.2. Equilibrio líquido-vapor

Con lo expuesto en el apartado anterior sabríamos si una mezcla es líquido, vapor o mezcla de ambas pero, para este último caso, no conoceríamos la fracción de vapor ni la composición en cada fase. Para ello, se han de introducir dos nuevos conceptos: las constantes de equilibrio (K) y la fugacidad.

La fugacidad es una medida del potencial químico y da idea de la tendencia de una sustancia a una fase u otra. La formulación matemática se obtiene a partir de la energía libre de Gibbs, pero para este proyecto basta con saber que equivale a (Prausnitz 2000; Michelsen and Mollerup. 2007):

$$f = \varphi \cdot P \tag{28}$$

Dónde *f* es la fugacidad, φ el coeficiente de fugacidad y P la presión del sistema.

Por su parte, la constante de equilibrio es otra medida de la tendencia de una especie a estar en una fase u otra, pero, en función de la concentración:

$$K_i = \frac{y_i}{x_i} \tag{29}$$

Dónde K_i es la constante de equilibrio, x_i la fracción molar en la fase líquida del compuesto "i" e y_i la fracción molar en la fase gas.

Definidas la fugacidad y la constante de equilibrio se procede al estudio del equilibrio líquido- vapor. Dos fases están en equilibrio cuando sus fugacidades se igualan:

$$f_v = f_l \tag{30}$$

Dónde f_v es la fugacidad de la fase vapor y f_l la de la líquida.

Por ello, para cada componente se ha de cumplir también la igualdad de fugacidades en la fase vapor (f_{vi}) y en la fase líquida (f_{li}):

$$f_{v_i} = f_{l_i} \tag{31}$$

Recordando la definición de fugacidad y de la constante de equilibrio se llega a una expresión que nos permite calcular las constantes de equilibrio mediante la obtención de los coeficientes de fugacidad de cada especie (lo cual, se hará, mediante ecuaciones de estado):

$$\varphi_{v_i} \cdot P \cdot y_i = \varphi_{l_i} \cdot P \cdot x_i \quad (32)$$

$$K_i = \varphi_{l_i} / \varphi_{v_i} \quad (33)$$

Obtenidas la constantes ya puede resolverse cualquier problema de equilibrio líquido vapor. En este trabajo se plantean tres escenarios:

1. Dada una alimentación multicomponente a una cierta temperatura T y una presión P obtener su estado y la composición en cada fase.

Para este escenario basta con aplicar el método de Rachford-Rice (Treybal 1980) para el caso isotérmico con ayuda del método de Newton-Raphson (consultar anexo I):

- Se toma un estado inicial (x=y=0,5 y fracción de vapor =0,5) y junto con él y las condiciones de operación dadas se calculan unos coeficientes de fugacidad que nos dan unas constantes de equilibrio iniciales.

- Una vez conocidas esas constantes se resuelve la el método de Rachford-Rice para obtener la fracción de vapor, de tal forma que la siguiente ecuación debe ser cero:

$$\sum_{i=1}^{c} \frac{z_i \cdot (K_i - 1)}{1 + (K_i - 1) \cdot a} = 0 \quad (34)$$

  Dónde C es el número de componentes totales, "i" el componente actual, K_i la constante de equilibrio del componente "i", z_i la fracción molar de la alimentación del componente "i" y a la fracción de vapor.

- Calculada la fracción de vapor pueden obtenerse las composiciones de cada fase:

$$x_i = \frac{z_i}{1+(K_i-1)\cdot a} \quad (35)$$

$$y_i = K_i \cdot x_i \quad (36)$$

- Finalmente, se procede a comparar estas composiciones calculadas con las supuestas, si son iguales, se cierra el método y ese es el estado real de la alimentación, en caso contrario, se repite el proceso tomando como composición y fracción de vapor inicial las calculadas.

2. Dada una alimentación multicomponente como líquido saturado obtener su presión de burbuja y temperatura de burbuja.

Este segundo escenario es similar al anterior, solo que en vez de la ecuación de Rachford-Rice se utiliza la definición del punto de burbuja (Treybal 1980):

- Se toma la solución del problema ideal como estado inicial, obteniendo, de nuevo, unas constantes de equilibrio iniciales y una presión (o temperatura) de burbuja inicial.

- Una vez conocidas esas constantes aplica el método de Newton-Raphson (ver apéndice I) a la definición del punto de burbuja, el cual nos da un punto de burbuja calculado:

$$1 - \sum_{i=1}^{c} K_i \cdot z_i = 0 \quad (37)$$

Dónde C es el número de componentes totales, "i" el componente actual, K_i la constante de equilibrio del componente "i" y z_i la fracción molar de la alimentación del componente "i".

- Finalmente, se procede a comparar ambos puntos de burbuja, si coinciden, termina el método, en caso contrario se toma como nuevo punto inicial el último calculado.

3. Dada una alimentación multicomponente como vapor saturado obtener su presión de rocío y temperatura de rocío.

Esta última posibilidad es exactamente igual al anterior solo que cambiando la definición del punto de burbuja por la de rocío (Treybal 1980):

$$1 - \sum_{i=1}^{c} \frac{z_i}{K_i} = 0 \quad (38)$$

Dónde C es el número de componentes totales, "i" el componente actual, K_i la constante de equilibrio del componente "i" y z_i la fracción molar de la alimentación del componente "i".

3.2.3. Relaciones de Maxwell

Partiendo de una expresión general para las cuatro ecuaciones fundamentales (dU, dH, dG, y dA) (CENGEL and BOLES 2001; Michelsen and Mollerup. 2007):

$$dZ = M \cdot dX + N \cdot dY \quad (39)$$

Teniendo en cuenta que la diferencial de Z(f)=Z(X,Y) se puede expresar como:

$$dZ = (\frac{\partial Z}{\partial Y})_Y \cdot dX + (\frac{\partial Z}{\partial X})_X \cdot dY \quad (40)$$

Y comprando ambas ecuaciones se llega a qué:

$$M = (\frac{\partial Z}{\partial X})_Y \quad (41)$$

$$N = (\frac{\partial Z}{\partial Y})_X \quad (42)$$

Finalmente, aplicando esta deducción a las ecuaciones fundamentales y como se cumple el criterio de exactitud matemática (para un sistema simple, compresible, monofásico y en equilibrio):

$$dU = T \cdot dS - P \cdot dV \quad (43)$$

$$(\frac{\partial T}{\partial V})_S = -(\frac{\partial P}{\partial S})_V \quad (44)$$

$$dA = -S \cdot dT - P \cdot dV \quad (45)$$

$$(\frac{\partial S}{\partial V})_T = -(\frac{\partial P}{\partial T})_V \quad (46)$$

$$dH = T \cdot dS - V \cdot dP \quad (47)$$

$$(\frac{\partial T}{\partial P})_S = -(\frac{\partial V}{\partial S})_P \quad (48)$$

$$dG = -S \cdot dT + V \cdot dP \quad (49)$$

$$(\frac{\partial S}{\partial P})_T = -(\frac{\partial V}{\partial T})_P \quad (50)$$

3.2.4. Cálculo de propiedades energéticas

A partir de las relaciones de Maxwell se llega a qué (ver anexo II):

- Entalpía:

$$dH = C_p \cdot dT + \left[V - T \cdot \left(\frac{\partial V}{\partial T}\right)_P\right] \cdot dP \quad (51)$$

- Entropía:

$$dS = \frac{C_p}{T} \cdot dT - T \cdot \left(\frac{\partial V}{\partial T}\right)_P \cdot dP \quad (52)$$

- Fugacidad (energía de Gibbs):

$$R \cdot T \cdot dln(\varphi) = \left[V - R \cdot \frac{T}{P}\right] \cdot dP \quad (53)$$

Dónde φ es coeficiente de fugacidad y R la constante de los gases ideales. Solo se han desarrollado estas tres expresiones, porque a partir de las dos primeras se pueden obtener las demás y porque la tercera nos da el equilibrio líquido vapor (Bruce E. Poling 2001; Michelsen and Mollerup. 2007).

- Energía interna:

$$U = H - P \cdot V \tag{54}$$

- Energía libre de Gibbs:

$$G = H - T \cdot S \tag{55}$$

- Exergía:

$$B = H - T \cdot S - P \cdot V \tag{56}$$

Tras un primer examen de las expresiones anteriores se deduce que precisamos dos cosas: una expresión que nos permita obtener Cp (capacidad calorífica a presión constante) y otra que nos relacione la presión con el volumen y la temperatura (las ecuaciones de estado).

El problema asociado a la Cp se solventa mediante el uso de discrepancias o propiedades residuales, de tal forma que, para una propiedad M de un sistema real:

$$M = M^{id} - \Delta M' \tag{57}$$

Dónde $\mathrm{M^{id}}$ es el valor de la propiedad como gas ideal y $\Delta \mathrm{M}'$ la discrepancia con la realidad.

El proceso a seguir es el siguiente:

1) Partiendo de un estado real (1) se pasa a gas ideal (1*) mediante la discrepancia entre ambos.
2) Se procede a la variación de temperatura del proceso, llegando al estado 2* (todavía como gas ideal).
3) Finalmente, se pasa de gas ideal al estado real (2) calculando la discrepancia entre ambos.

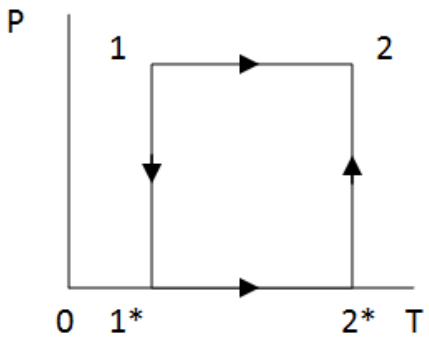


Figura 3: Proceso de cálculo de propiedades energéticas

De tal forma que, la variación de la propiedad M se calcula como:

$$M_2 - M_1^* = (M_2 - M_2^*) + (M_2^* - M_1^*) - (M_1 - M_1^*) \quad (58)$$

La utilidad de este procedimiento radica en que, generalmente, no disponemos de una expresión que nos permita calcular la Cp en el estado real en función de la temperatura pero, en cambio, si existe una correlación para el caso de gas ideal:

$$Cp = \alpha + \beta \cdot T + \gamma \cdot T^2 + \delta \cdot T^4 \quad (59)$$

Dónde los coeficientes α, β, γ y δ dependen de la sustancia.

Como puede observarse del desarrollo anterior, lo único que se podría calcular hasta ahora serían variaciones de propiedades, para poder obtener un valor absoluto se ha de fijar un estado inicial de propiedades conocidas. Para este trabajo el punto inicial tomado fue: gas ideal a P= 1atm, T=0ºC, S y H nulas (dados estos valores, el resto de propiedades se obtienen a partir de ellos).

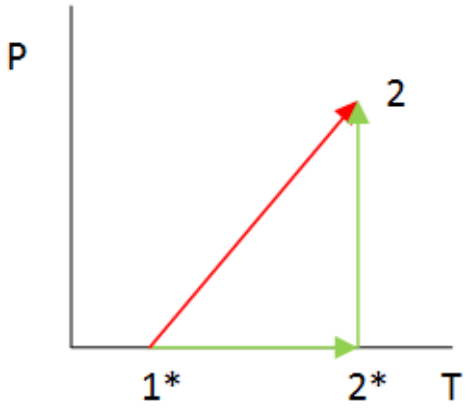


Figura 4: Proceso usado para calcular propiedades energéticas

Aplicando el desarrollo anterior a las expresiones obtenidas a partir de las relaciones de Maxwell, se llega a que la variación de entalpía entre los puntos 1 y 2 se obtiene de:

$$\Delta H = \int_{T_1}^{T_2} C_p \cdot dT - \Delta H' \tag{60}$$

Dónde $\Delta H'$ es la discrepancia definida de la siguiente manera:

$$\Delta H' = -R \cdot T \cdot [1 - Z] - T \int_{\infty}^{V} \left(\frac{\delta Z}{\delta Z}\right)_V \cdot \frac{dV}{V} \tag{61}$$

Siendo Z el coeficiente de compresibilidad.

Por otro lado, en el caso de la entropía:

$$\Delta S = \int_{T_1}^{T_2} \frac{C_p}{T} \cdot dT + R \cdot ln\left(\frac{P_1}{P_2}\right) - \Delta S' \tag{62}$$

$$\Delta S' = R \cdot T^2 \cdot \int_0^P \left[(Z-1) + T \cdot \left(\frac{\delta Z}{\delta T}\right)_P\right] \cdot \frac{dP}{P} \tag{63}$$

Finalmente, para la fugacidad no es necesario calcular discrepancias, pues es una propiedad que hace referencia al equilibrio líquido-vapor y no energética (puede observarse no hay dependencia de la Cp) (Prausnitz 2000; Michelsen and Mollerup. 2007):

$$ln(\varphi) = \int_0^P \frac{1}{RT} \cdot \left[V - R \cdot \frac{T}{P}\right] \cdot dP \quad (64)$$

Halladas estas expresiones, el siguiente paso es aplicar las ecuaciones de estado antes mencionadas para conseguir las expresiones de cálculo (Prausnitz 2000; Bruce E. Poling 2001; Michelsen and Mollerup. 2007):

1. VdW

i. Fugacidad (componentes puros):

$$ln(\varphi) = \frac{b}{V-b} - 2 \cdot \frac{a}{R \cdot T \cdot V} - ln\left[Z \cdot \left(1 - \frac{b}{V}\right)\right] \quad (65)$$

ii. Fugacidad parcial (mezclas):

$$ln(\varphi_i) = \frac{b_i}{V-b} - 2 \cdot \frac{\sqrt{a \cdot a_i}}{R \cdot T \cdot V} - ln\left[Z \cdot \left(1 - \frac{b}{V}\right)\right] \quad (66)$$

$$a = \left(\sum y_i \cdot \sqrt{a_i}\right)^2 \quad (67)$$

$$b = \sum y_i \cdot b_i \quad (68)$$

Dónde y_i es la fracción molar del componente "i" y φ_i el coeficiente de fugacidad de dicho componente.

iii. Entalpía:

$$\Delta H' = R \cdot T \cdot \left[2 \cdot \frac{a}{R \cdot T \cdot V} - \frac{b}{V-b}\right] \quad (69)$$

iv. Entropía

$$\Delta S' = -R \cdot ln\left[Z \cdot \left(1 - \frac{b}{V}\right)\right] \quad (70)$$

2. SRK

i. Fugacidad (componentes puros):

$$ln(\varphi) = Z - 1 - ln(Z - B) - \frac{A}{B} \cdot ln\left(1 + \frac{B}{Z}\right) \quad (71)$$

ii. Fugacidad parcial(mezclas):

$$ln(\varphi_i) = \frac{B_i}{B} \cdot (Z - 1) - ln(Z - B) + \frac{A}{B} \cdot \left[\frac{B_i}{B} - \frac{2}{a \cdot \alpha} \cdot \sum_j y_j \cdot (a \cdot \alpha)_{ij}\right] \cdot ln\left(1 + \frac{B}{Z}\right) \quad (72)$$

Dónde y_j es la fracción molar de los componentes diferentes a "i".

$$(a \cdot \alpha)_{ij} = \left(1 - k_{ij}\right) \cdot \sqrt{(a \cdot \alpha)_{ii} \cdot (a \cdot \alpha)_{jj}} \quad (73)$$

$$b = \sum y_i \cdot b_i \quad (74)$$

$$a \cdot \alpha = \sum \sum y_i \cdot y_j \cdot (a \cdot \alpha)_{ij} \quad (75)$$

$$A = a \cdot \alpha \cdot \frac{P}{(R \cdot T)^2} \quad (76)$$

$$B = b \cdot \frac{P}{R \cdot T} \quad (77)$$

$$B_i = b_i \cdot \frac{P}{R \cdot T} \quad (78)$$

iii. Entalpía

$$\Delta H' = R \cdot T \cdot \left[1 - Z + \frac{A}{B} \cdot \left(1 + \frac{D}{a \cdot \alpha}\right) \cdot ln\left(1 + \frac{B}{Z}\right)\right] \quad (79)$$

$$D = \sum_i \sum_j y_i \cdot y_j \cdot \left(1 - k_{ij}\right) \cdot \sqrt{a_i \cdot \alpha_i} \cdot \sqrt{a_j \cdot T_{rj}} \quad (80)$$

iv. Entropía

$$\Delta S' = -R \cdot ln(Z - B) + A \cdot \frac{D}{B \cdot a \cdot \alpha} \cdot ln\left(1 + \frac{B}{Z}\right) \quad (81)$$

3. PR

i. Fugacidad (componentes puros):

$$ln(\varphi) = Z - 1 - ln(Z - B) - \frac{A}{2 \cdot \sqrt{2} \cdot B} \cdot ln\left(\frac{Z + 2.414 \cdot B}{Z - 0.414 \cdot B}\right) \quad (82)$$

ii. Fugacidad parcial(mezclas):

$$ln(\varphi_i) = \frac{B_i}{B}(Z - 1) - ln(Z - B) - \frac{A}{2 \cdot \sqrt{2} \cdot B} \cdot \left[\frac{B_i}{B} - \frac{2}{a \cdot \alpha} \cdot \sum_j y_j \cdot (a \cdot \alpha)_{ij}\right] \cdot ln\left(\frac{Z + 2.414 \cdot B}{Z - 0.414 \cdot B}\right) \quad (83)$$

Las variables son las mimas que en SRK.

iii. Entalpía

$$\Delta H' = R \cdot T \cdot \left[1 - Z + \frac{A}{2 \cdot \sqrt{2} \cdot B} \cdot \left(1 + \frac{D}{a \cdot \alpha}\right) \cdot ln\left(\frac{Z + 2.414 \cdot B}{Z - 0.414 \cdot B}\right)\right] \quad (84)$$

iv. Entropía

$$\Delta S' = -R \cdot ln(Z - B) + B \cdot \frac{D}{2 \cdot \sqrt{2} \cdot A \cdot a \cdot \alpha} \cdot ln\left(\frac{Z + 2.414 \cdot B}{Z - 0.414 \cdot B}\right) \quad (85)$$

3.2.5. Algoritmo de resolución

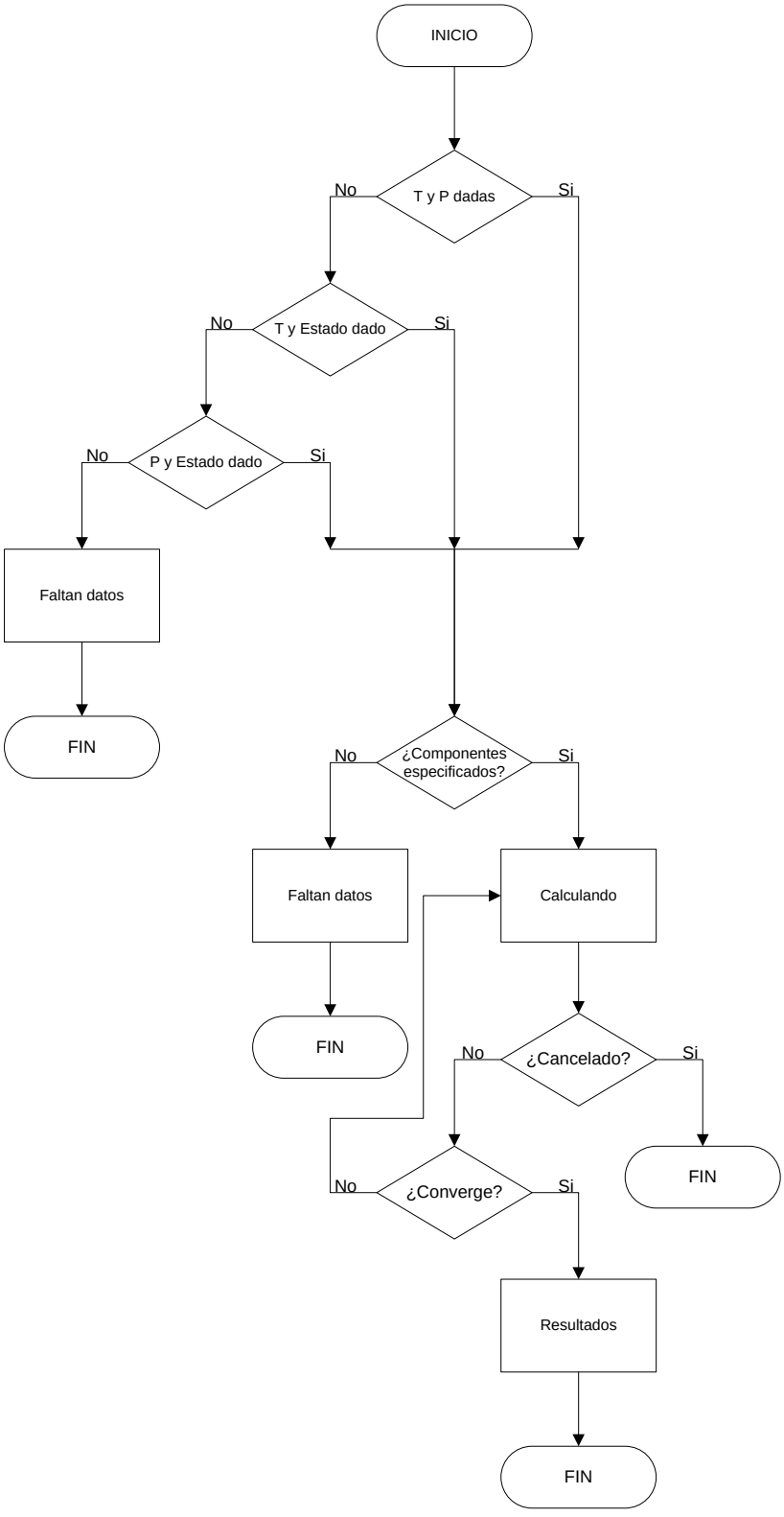


Diagrama 3: Algoritmo de resolución de "Calculator of properties"

3.3. Base de Datos

Como ya se ha indicado anteriormente, esta aplicación es una continuación de la desarrollada por Álvaro Cabeza en su Proyecto de fin de Carrera. (Cabeza Sánchez, 2012).

En un principio, los 78 componentes (ver Anexo III) con los que trabaja la aplicación estaban insertados en una matriz dentro de una clase (ver apartado 1.2.). Esto poseía la ventaja de que cuando la aplicación quería realizar algún tipo de operación, simplemente tenía que acceder a determinadas posiciones de la matriz, extraer el valor que allí se encontraba y utilizarlo.

No obstante, se podría decir que esto es un nivel muy básico de Base de Datos, ya que posee un gran nivel de rigidez al sólo permitir al usuario operar con los datos introducidos por el programador.

Uno de los principales objetivos de este TFG fue transformar dicha base de datos en otra que fuese accesible para el usuario de la aplicación, y que le permitiese tanto añadir nuevos componentes, como editar o borrar los ya existentes en la aplicación base. Para ello, se empleó el lenguaje de consulta estructurado conocido como 'SQL'.

'SQL' es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo de álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ellas (Date y Darwen, 1997)

.

3.3.1. Orígenes y evolución

Los orígenes del SQL están ligados a los de las bases de datos relacionales. En 1970 E. F. Codd. Propone el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definieron el lenguaje SEQUEL (Structured English Query Language) que más tarde fue ampliamente implementado por el sistema de gestión de bases de datos (SGBD) experimental System R, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un producto comercial.

El SEQUEL terminó siendo el predecesor de SQL, que es una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el “”SQL-86” o “SQL1”. Al año siguiente este estándar es también adoptado por la ISO.

Sin embargo, este primer estándar no cubría todas las necesidades de los desarrolladores e incluía funcionalidades de definición de almacenamiento que se consideró suprimirlas. Así que, en 1992, se lanzó un nuevo estándar ampliado y revisado del SQL llamado "SQL-92" o "SQL2".

En la actualidad el SQL es el estándar *de facto* de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

3.3.2. Características generales del SQL

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros —y no a registros individuales— permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. SQL también tiene las siguientes características:

- **Lenguaje de definición de datos**: El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.

- **Lenguaje interactivo de manipulación de datos**: El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.

- **Integridad**: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.

- **Definición de vistas**: El LDD incluye comandos para definir las vistas.

- **Control de transacciones**: SQL tiene comandos para especificar el comienzo y el final de una transacción.

- **SQL incorporado y dinámico**: Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, PHP, Cobol, Pascal y Fortran.

- **Autorización**: El LDD incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

3.3.3. SQL para Android

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias. {Oliver, #3}.

La forma típica para crear, actualizar y conectar con una base de datos SQLite será a través de una clase auxiliar "SqliteOpenHelper", o, mejor dicho, de una clase propia que derive en ella y que se debe personalizar a las necesidades concretas de la aplicación.

La clase "SQliteOpenHelper" tiene tan sólo un constructor, que normalmente no necesitaremos sobrescribir, y dos métodos abstractos, 'OnCreate()' y 'OnUpgrade()', que se deben personalizar con el código necesario para crear la base de datos y actualizar su estructura respectivamente.

Una vez definida esta clase, denominada *'clase Helper'*, la apertura desde la base de datos resulta algo sencillo. En primer lugar se creará un objeto al que se le pasará el contexto de la aplicación, el nombre de la base de datos y la última versión actualizada de la base de datos.

- Si la base de datos ya existe y su versión actual coincide con la solicitada simplemente se realizará la conexión con ella.

- Si la base de datos existe pero su versión actual es anterior a la solicitada, se llamará automáticamente al método 'onUpgrade()' para convertir la base de datos a la nueva versión y se conectará con la base de datos convertida.

- Si la base de datos no existe, se llamará automáticamente al método onCreate() para crearla y se conectará con la base de datos creada.

Una vez creada la referencia al objeto, se llamará al método 'getReadableDatabase()' o 'GetWritableDatabase()', dependiendo si sólo en necesario realizar una consulta o si es necesario realizar modificaciones, respectivamente.

De este modo, se habrá creado una base de datos básica en SQL, a la que se le añadirán los parámetros necesarios para poder cumplir con las exigencias de la aplicación (ver Anexo VI).

3.4. Resolución de los cálculos

Otro de los principales cambios realizados en la aplicación de partida, es la inclusión de tareas asíncronas a la hora de realizar los cálculos pertinentes para las mezclas seleccionadas. En la versión anterior, los cálculos se realizaban a través de 'threads' (hilos), que tienen como característica principal la capacidad de poder ejecutarse en segundo plano, pero que en caso de ser interrumpidos, fuerzan a la aplicación a bloquearse.

3.4.1. Tareas en segundo plano en Android

Todos los componentes de una aplicación Android se ejecutan en un mismo hilo de ejecución, llamado *'hilo principal'* o *'main thread'*, que también es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Es por ello, que cualquier operación larga o costosa

que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo, o mal funcionamiento en general, algo que deberíamos evitar a toda costa. Incluso puede ser peor, dado que Android monitoriza las operaciones realizadas en el hilo principal y detecta aquellas que superen los 5 segundos, en cuyo caso se muestra el famoso mensaje de "*Application Not Responding*" (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine.

Figura 5: Mensaje de error de aplicación

Para evitar la aparición de este mensaje, surgen dos opciones:

1. Crear de forma explícita un nuevo hilo para ejecutar la tarea.
2. Utilizar la clase *AsyncTask* proporcionada por Android.

En el momento en que comienza a ejecutarse un hilo, el resto de la aplicación se bloquea, incluida la actualización de la interfaz del usuario, por lo que cada vez que se quiera interactuar con la aplicación y el hilo no haya terminado su ejecución, el programa se colapsará y mostrará el mensaje de la figura 5 de forma automática.

Si se emplea la primera opción (creación de un nuevo hilo secundario), se podría solventar el problema del bloqueo de la aplicación, pero el código resultante es demasiado engorroso y complejo, llegando a ser inmanejable para códigos complejos, por lo que es más conveniente partir de la segunda opción y emplear las tareas asíncronas.

3.4.2. AsyncTask

La clase '*AsyncTask*' de Android permite realizar las mismas funciones que los hilos secundarios (ver apartado 1.2.), con la ventaja de no tener que utilizar constructores complejos y que el código generado es mucho más sencillo y legible. La forma básica de la clase '*AsyncTask*' consiste en crear una nueva clase que parta de ella y sobrescribir varios de sus métodos entre los que se repartirá la funcionalidad de la tarea. Dichos métodos son:

- onPreExecute(): Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.

- doInBackground(). Contendrá el código principal de nuestra tarea.

- onProgressUpdate(). Se ejecutará cada vez que llamemos al método publishProgress() desde el método doInBackground().

- onPostExecute(). Se ejecutará cuando finalice nuestra tarea, o dicho de otra forma, tras la finalización del método doInBackground().

- onCancelled(). Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

Estos métodos tienen una particularidad esencial para nuestros intereses (Oliver, S. G., 2014). El método doInBackground() se ejecuta en un hilo secundario (por tanto no podremos interactuar con la interfaz), pero sin embargo todos los demás se ejecutan en el hilo principal, lo que quiere decir que dentro de ellos podremos hacer referencia directa a nuestros controles de usuario para actualizar la interfaz. Por su parte, dentro de doInBackground() tendremos la posibilidad de llamar periódicamente al método publishProgress() para que automáticamente desde el método onProgressUpdate() se actualice la interfaz si es necesario. Al extender una nueva clase de AsyncTask indicaremos tres parámetros de tipo:

1. El tipo de datos que recibiremos como **entrada** de la tarea en el método doInBackground().

2. El tipo de datos con el que actualizaremos el **progreso** de la tarea, y que recibiremos como parámetro del método onProgressUpdate() y que a su vez tendremos que incluir como parámetro del método publishProgress().

3. El tipo de datos que devolveremos como **resultado** de nuestra tarea, que será el tipo de retorno del método doInBackground() y el tipo del parámetro recibido en el método onPostExecute().

Una vez definidos estos parámetros, lo único que queda por hacer es seleccionar qué partes del código corresponden a cada uno de los métodos. En el caso de nuestra aplicación:

- En el método OnPreExecute(), se declararán todas las variables que serán utilizadas y se abrirá la base de datos SQL para que la clase actual pueda acceder a ella para realizar los cálculos necesarios.

- En el método DoInBackground() se ejecutarán todos los cálculos. Esto se hará de forma independiente al resto de la aplicación, por lo que un excesivo tiempo de cálculo no bloqueará a la misma.

- En el método OnPostExecute() se mostrarán por pantalla los resultados obtenidos.

La introducción de las tareas asíncronas también permite la posibilidad de cancelar la tarea en medio de su ejecución. Para ello, en los bucles de cálculo del método DoInBackground() se hace una llamada a isCancelled(), que en caso de ser afirmativo, ejecuta el método onCancelled(), que se encarga de cancelar la ejecución de los cálculos.

3.5. Diagramas de fase

Otra de las novedades que ofrece esta segunda versión de la aplicación, es la posibilidad de obtener el diagrama de fase de los componentes puros que se encuentran en la base de datos. Se ha planteado como posibilidad para el futuro implementar la construcción de diagramas de fase de componentes binarios, algo de lo que la propia aplicación se encarga de avisar si se pretende obtener este tipo de gráficas (ver Anexo IV).

Para dibujar dichas gráficas se ha empleado la clase 'Canvas' que ofrece la programación en Android. La clase 'Canvas' representa una superficie donde se puede dibujar. Dispone de una serie de métodos que permiten al programador representar líneas, círculos, texto, o, en general, cualquier cosa en 2D que se le pueda ocurrir. Para dibujar en Canvas es necesario definir un lienzo sobre el que se pueda pintar, un pincel (al que se accederá mediante la invocación del método 'Paint') donde podrá definirse el color, grosor de trazo, transparencia, o cualquier otra característica deseada. También es posible definir una matriz de 3x3 (Matrix) que permitirá transformar coordenadas aplicando una translación, escala o rotación. Otra opción consiste en definir un área conocida como Clip, de forma que los métodos de dibujo afecten solo a esta área.

Una vez definidas las diferentes partes del lienzo, el programa se encargará de tomar los datos introducidos por el usuario y de hacer los cálculos pertinentes para la obtención de las gráficas.

3.5.1. P vs T

La primera de las tres opciones que la interfaz de la aplicación le ofrece al usuario para graficar es la posibilidad de obtener la Presión frente a la Temperatura.

Para ello, se solicita la elección de una ecuación para realizar los cálculos (SRK o PR. Se ha excluido la posibilidad de obtener los valores por medio de las ecuaciones de Van der Waals, ya que a la hora de calcular temperaturas y presiones de saturación es poco preciso), y se pide al usuario que introduzca una temperatura inicial y otra final.

Por medio de cálculos internos, la aplicación calcula la presión de saturación a dichas temperaturas, además de la misma para cuatro valores intermedios

a ambas y equidistantes entre sí. Una vez realizado el cálculo, representa sobre un eje de coordenadas los puntos generados con cada temperatura, y los une formando la curva representativa del diagrama de fase.

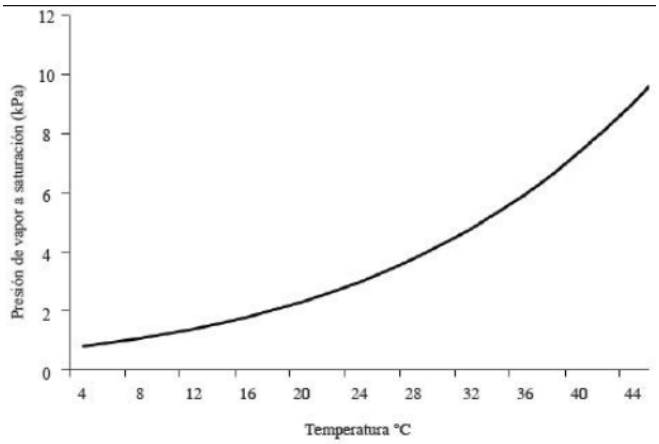


Figura 6: Diagrama representativo de Presión-Temperatura

3.5.2. T vs P

Opera del mismo modo que la representación anterior, con la salvedad de que ahora la Presión será establecida por el usuario, y la Temperatura de saturación calculada por la aplicación. Nuevamente se calcularán los cuatro puntos intermedios y procederá a la representación de la gráfica.

Como es evidente, la forma de la gráfica tenderá a ser simétrica con respecto al apartado anterior. Es decir, ante un mismo componente, si su gráfica de Presión vs Temperatura es de carácter cóncavo, la gráfica Temperatura vs Presión tendrá forma convexa.

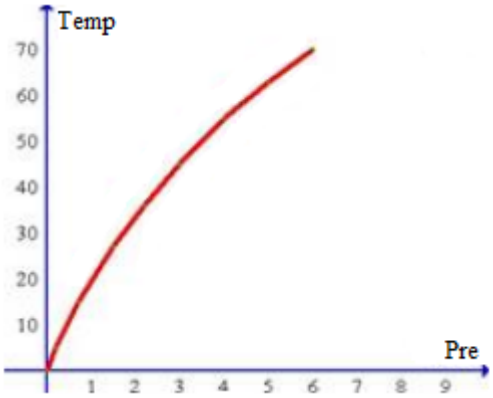


Figura 7: Diagrama representativo de Temperatura-Presión

3.5.3. Compuestos binarios

El graficado de compuestos binarios se presenta como la tercera posibilidad de obtención de diagramas, y es una mejora que se introducirá en una próxima versión de la aplicación.

La presencia de estas gráficas se caracteriza sobre la de los compuestos puros en que aparecerían dos curvas en la misma gráfica, representando al Líquido y al Vapor saturado. La superficie abarcada entre ambas curvas es una zona en la que ambos estados coexisten.

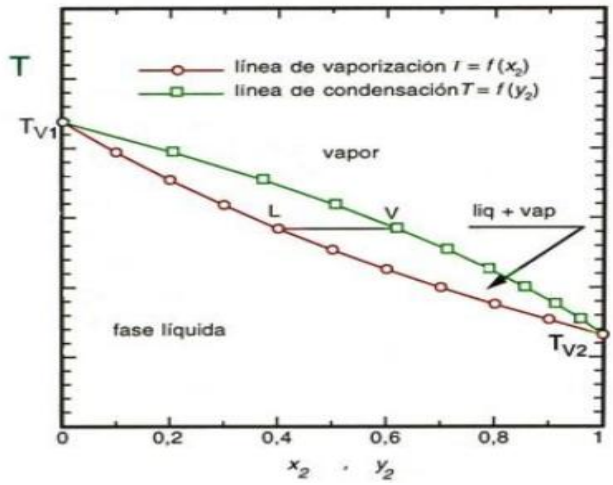


Figura 8: Diagrama de fase binario

Aparentemente, la aplicación ofrece la posibilidad de realizar estas gráficas, pero como el usuario podrá observar cuando trate de ejecutar esta opción, en la pantalla aparecerá un mensaje de aviso que indica que aún no está implementada, además de desaparecer el botón necesario para dibujar (para más información, Anexo IV).

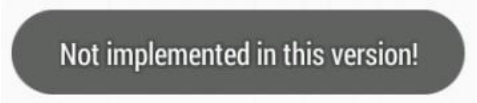


Figura 9: Mensaje de aviso en compuestos binarios

3.6. Ejemplos

Para comprobar los resultados generados por la aplicación, se comparan con valores experimentales o con los valores generados por el programa AspenONE.

Propiedades de equilibrio

Las mayores discrepancias aparecen cuando se trata de mezclar dos componentes muy diferentes entre sí, como el caso del agua y el pentano, ya que ambas son inmiscibles y la aplicación los trata como miscibles (Tabla 9). Por otra parte, cuanto más se parecen las sustancias que forman la mezcla, menor es el error producido (caso hexano-nonano, Tabla 12).

También hay que tener en cuenta la polaridad de las sustancias. Las EOS no reproducen muy bien las sustancias polares. Por tanto una mezcla por lar/apolar no la reproducirán bien enfase líquida (tabla 11).

También se puede comprobar que cuanto mayor es la presión, aumenta el error, ya que la sustancia se acerca a su estado crítico (Tablas 2 a 11). Así mismo, cuantas más especies compongan la mezcla, mayores diferencias habrá entre el valor calculado y el valor de referencia (Tablas 33 a 37).

Cómo conclusión general puede deducirse que mientras que las sustancias no difieran mucho entre sí y la mezcla se encuentre lejos de las condiciones críticas, el error es inferior al 10 %.

Propiedades energéticas

Tal y como se puede observar desde la Tabla 48 hasta la Tabla 53 los mayores errores (en torno a un 20%) se cometen en la entropía y en la densidad del líquido (Ecuaciones SRK y PR). Esto era de esperar pues una de las simplificaciones a la hora de hacer esta aplicación fue que la fase líquida se comportarse de manera ideal.

Como también puede observarse, al aplicar la ecuación de los volúmenes trasladados, las densidades sufren una tendencia general a aproximarse más al valor experimental (Tablas 48 a 51).

- Sustancias puras en saturación (Temperatura de saturación).

		SRK				PR				PRVT			
	P (atm)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error
Agua	1 atm	102,44	102,38	0,06	0,06	101,90	101,37	0,53	0,52	101,90	101,37	0,53	0,52
	5 atm	153,17	153,11	0,06	0,04	153,30	152,82	0,48	0,31	153,30	152,82	0,48	0,31
	10 atm	180,50	180,45	0,05	0,03	180,97	180,52	0,45	0,25	180,97	180,52	0,45	0,25
	50 atm	263,10	263,04	0,06	0,02	264,20	263,88	0,32	0,12	264,20	263,88	0,32	0,12
	100 atm	310,03	309,97	0,06	0,02	311,05	310,83	0,22	0,07	311,05	310,83	0,22	0,07

Tabla 2: Comparativa de la T de saturación del agua a distintas presiones.

		SRK				PR				PRVT			
	P (atm)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
NH3	1 atm	-32,61	-32,08	-0,53	1,63	-33,00	-32,62	-0,38	-1,15	-33,00	-32,62	-0,38	-1,15
	5 atm	4,71	5,34	-0,63	-13,38	4,81	5,32	-0,51	-10,60	4,81	5,32	-0,51	-10,60
	10 atm	25,28	25,97	-0,69	-2,73	25,60	26,19	-0,59	-2,30	25,60	26,19	-0,59	-2,30
	50 atm	89,06	89,91	-0,85	-0,95	89,59	90,45	-0,86	-0,96	89,59	90,45	-0,86	-0,96
	100	126,11	127,07	-0,96	-0,76	126,17	127,20	-1,03	-0,82	126,17	127,20	-1,03	-0,82

Tabla 3: Comparativa de la T de saturación del amoniaco a distintas presiones.

		SRK				PR				PRVT			
Naftaleno	P (atm)	Aspen	Programa	E. absoluto	Error	Aspen	Programa	E. absoluto	Error	Aspen	Programa	E. absoluto	Error (%)
	1 atm	218,25	218,23	0,02	0,01	217,95	217,91	0,04	0,02	217,95	217,91	0,04	0,02
	5 atm	302,73	302,70	0,03	0,01	303,43	303,38	0,05	0,02	303,43	303,38	0,05	0,02
	10 atm	350,26	350,22	0,04	0,01	351,28	351,22	0,06	0,02	351,28	351,22	0,06	0,02

Tabla 4: Comparativa de la T de saturación del Naftaleno a distintas presiones.

		SRK				PR				PRVT			
R12	P (atm)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
	1 atm	-29,49	-29,43	-0,06	0,20	-30,12	-29,80	-0,32	1,06	-30,12	-29,80	-0,32	1,06
	5 atm	15,96	16,23	-0,27	-1,69	16,37	16,45	-0,08	-0,49	16,37	16,45	-0,08	-0,49
	10 atm	41,82	42,23	-0,41	-0,98	42,74	42,66	0,08	0,19	42,74	42,66	0,08	0,19

Tabla 5: Comparativa de la T de saturación del R12 a distintas presiones.

		SRK				PR				PRVT			
Octano	P (atm)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error	Aspen	Programa	E. absoluto	Error (%)
	1 atm	125,67	125,50	0,17	0,14	125,91	125,50	0,41	0,33	125,91	125,50	0,41	0,33
	5 atm	194,66	194,63	0,03	0,02	195,54	195,31	0,23	0,12	195,54	195,31	0,23	0,12
	10 atm	233,58	233,63	-0,05	-0,02	234,49	234,39	0,10	0,04	234,49	234,39	0,10	0,04

Tabla 6: Comparativa de la T de saturación del Octano a distintas presiones.

		SRK				PR				PRVT			
	P (atm)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Metano	1 atm	-160,99	-161,01	0,02	-0,01	-161,65	-161,59	-0,06	0,04	-161,65	-161,59	-0,06	0,04
	5 atm	-137,46	-137,48	0,02	-0,01	-137,75	-137,68	-0,07	0,05	-137,75	-137,68	-0,07	0,05
	10 atm	-123,85	-123,87	0,02	-0,02	-123,95	-123,89	-0,06	0,05	-123,95	-123,89	-0,06	0,05

Tabla 7: Comparativa de la T de saturación del Metano a distintas presiones.

**Nota: Como se puede observar, los valores coinciden con PRVT y PR en todas las propiedades excepto en el caso de la densidad, así que se retomará la comparativa en las últimas tablas (48 a 53, comparativa de propiedades energéticas).*

- Mezclas binarias (Temperatura de saturación).

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Hexano (0,5) Nonano (0,5)	1	0	90,92	91,21	-0,29	-0,32%	90,95	91,27	-0,32	-0,35%
		1	129,52	129,43	0,09	0,07%	129,38	129,19	0,19	0,15%
	5	0	161,61	162,03	-0,42	-0,26%	162,29	162,69	-0,40	-0,25%
		1	194,11	194,00	0,11	0,06%	194,63	194,46	0,17	0,09%
	10	0	202,59	203,30	-0,71	-0,35%	203,34	203,99	-0,65	-0,32%
		1	229,71	229,43	0,28	0,12%	230,30	229,98	0,32	0,14%

Tabla 8: Comparativa de la T de burbuja y de la T de rocío para el sistema equimolar Hexano-Nonano a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Pentano (0,5) Agua (0,5)	1	0	31,14	30,91	0,23	0,74%	30,28	29,91	0,37	1,22%
		1	84,60	35,14	49,46	58,46%	83,82	33,42	50,40	60,13%
	5	0	84,96	84,76	0,20	0,24%	84,27	83,97	0,30	0,36%
		1	128,93	85,95	42,98	33,34%	128,74	84,83	43,91	34,11%
	10	0	114,67	114,49	0,18	0,16%	113,98	113,73	0,25	0,22%
		1	152,15	114,84	37,31	24,52%	152,27	113,94	38,33	25,17%

Tabla 9: Comparativa de la T de burbuja y de la T de rocío para el sistema equimolar Pentano-Agua a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Etanol (0,5) NH3 (0,5)	1	0	-17,81	-17,12	-0,69	3,87%	-18,17	-17,57	-0,60	3,30%
		1	62,23	62,42	-0,19	-0,31%	60,61	61,26	-0,65	-1,07%
	5	0	26,65	27,79	-1,14	-4,28%	26,78	27,87	-1,09	-4,07%
		1	103,66	103,66	0,00	0,00%	102,75	103,16	-0,41	-0,40%
	10	0	52,28	54,00	-1,72	-3,29%	52,59	54,30	-1,71	-3,25%
		1	125,68	125,45	0,23	0,18%	125,09	125,20	-0,11	-0,09%

Tabla 10: Comparativa de la T de burbuja y de la T de rocío para el sistema equimolar Etanol-Amoniaco a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Tolueno (0,5) NH3 (0,5)	1	0	-23,00	-22,18	-0,82	3,57%	-24,30	-23,37	-0,93	3,83%
		1	88,32	88,30	0,02	0,02%	88,96	87,43	1,53	1,72%
	5	0	23,54	25,91	-2,37	-10,07%	22,61	25,11	-2,50	-11,06%
		1	145,30	144,65	0,65	0,45%	147,05	144,38	2,67	1,82%
	10	0	50,99	56,39	-5,40	-10,59%	50,24	55,73	-5,49	-10,93%
		1	175,98	173,32	2,66	1,51%	178,16	173,17	4,99	2,80%

Tabla 11: Comparativa de la T de burbuja y de la T de rocío para el sistema equimolar Tolueno-Amoniaco a distintas presiones.

- Mezclas binarias (fracción de vapor y composiciones)

			SRK				PR			
	P (atm)	T (º C)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error
Hexano (0,5) Nonano (0,5)	1,0	100	0,258	0,250	0,008	3,10%	0,258	0,250	0,008	3,10%
	5,0	185	0,665	0,660	0,005	0,75%	0,649	0,650	-0,001	-0,15%
	10,0	210	0,259	0,260	-0,001	-0,39%	0,236	0,230	0,006	2,54%

Tabla 12: Comparativa de la fracción de vapor para el sistema equimolar Hexano-Nonano a distintas presiones.

Aspen								Programa							
SRK				PR				SRK				PR			
x_{Hexa}	y_{Hexa}	x_{Nona}	y_{Nona}	x_{Hexa}	y_{Hexa}	x_{Nona}	y_{Nona}	x_{Hexa}	y_{Hexa}	x_{Nona}	y_{Nona}	x_{Hexa}	y_{Hexa}	x_{Nona}	y_{Nona}
0,373	0,863	0,627	0,137	0,374	0,861	0,626	0,139	0,38	0,86	0,62	0,14	0,38	0,86	0,62	0,14
0,268	0,617	0,732	0,383	0,273	0,623	0,727	0,377	0,27	0,62	0,73	0,38	0,27	0,62	0,73	0,38
0,426	0,711	0,574	0,289	0,433	0,716	0,567	0,284	0,43	0,71	0,57	0,29	0,43	0,72	0,57	0,28

Tabla 13: Comparativa de las composiciones en cada fase para el sistema equimolar Hexano-Nonano a distintas presiones (1 , 5 y 10 atm).

Errores Composiciones															
SRK				PR				SRK				PR			
x_{Hexano}	y_{Hexano}	x_{Nonano}	y_{Nonano}	x_{Hexano}	y_{Hexano}	x_{Nonano}	y_{Nonano}	x_{Hexano}	y_{Hexano}	x_{Nonano}	y_{Nonano}	x_{Hexano}	y_{Hexano}	x_{Nonano}	y_{Nonano}
-0,007	0,003	0,007	-0,003	-0,006	0,001	0,006	-0,001	-1,88%	0,35%	1,12%	-2,19%	-1,60%	0,12%	0,96%	-0,72%
-0,002	-0,003	0,002	0,003	0,003	0,003	-0,003	-0,003	-0,75%	-0,49%	0,27%	0,78%	1,10%	0,48%	-0,41%	-0,80%
-0,004	0,001	0,004	-0,001	0,003	-0,004	-0,003	0,004	-0,94%	0,14%	0,70%	-0,35%	0,69%	-0,56%	-0,53%	1,41%

Tabla 14: Comparativa de los errores de las composiciones en cada fase para el sistema equimolar Hexano-Nonano a distintas presiones (1 , 5 y 10 atm).

			SRK				PR			
	P (atm)	T (° C)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Etanol (0,5) NH3 (0,5)	1,0	40	0,565	0,560	0,005	0,88%	0,580	0,570	0,010	1,72%
	5,0	80	0,560	0,560	0,000	0,00%	0,569	0,560	0,009	1,58%
	10,0	100	0,533	0,530	0,003	0,56%	0,540	0,530	0,010	1,85%

Tabla 15: Comparativa de la fracción de vapor para el sistema equimolar Etanol-Amoniaco a distintas presiones.

Aspen								Programa							
SRK				PR				SRK				PR			
x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}
0,939	0,163	0,061	0,837	0,94	0,181	0,06	0,819	0,94	0,16	0,06	0,84	0,94	0,17	0,06	0,83
0,878	0,203	0,122	0,797	0,879	0,213	0,121	0,787	0,88	0,2	0,12	0,8	0,88	0,21	0,12	0,79
0,829	0,211	0,171	0,789	0,83	0,219	0,17	0,781	0,83	0,21	0,17	0,79	0,83	0,21	0,17	0,79

Tabla 16: Comparativa de las composiciones en cada fase para el sistema equimolar Etanol-Amoniaco a distintas presiones (1, 5 y 10 atm).

Errores Composiciones															
SRK				PR				SRK				PR			
x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}	x_{Etanol}	y_{Etanol}	x_{NH3}	y_{NH3}
-0,001	0,003	0,001	-0,003	0	0,011	-5,55E-17	-0,011	-0,11%	1,84%	1,64%	-0,36%	0,00%	6,08%	0,00%	-1,34%
-0,002	0,003	0,002	-0,003	-0,001	0,003	0,001	-0,003	-0,23%	1,48%	1,64%	-0,38%	-0,11%	1,41%	0,83%	-0,38%
-0,001	0,001	0,001	-0,001	0	0,009	0	-0,009	-0,12%	0,47%	0,58%	-0,13%	0,00%	4,11%	0,00%	-1,15%

Tabla 17: Comparativa de los errores de las composiciones en cada fase para el sistema equimolar Etanol-Amoniaco a distintas presiones (1, 5 y 10 atm).

			SRK				PR			
	P (atm)	T (º C)	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Tolueno (0,5) NH3 (0,5)	1,0	40	0,506	0,510	-0,004	-0,79%	0,508	0,510	-0,002	-0,39%
	5,0	60	0,383	0,380	0,003	0,78%	0,387	0,380	0,007	1,81%
	10,0	100	0,407	0,400	0,007	1,72%	0,409	0,410	-0,001	-0,24%

Tabla 18: Comparativa de las fracciones de vapor para el sistema equimolar Tolueno-Amoniaco a distintas presiones.

Aspen								Programa							
SRK				PR				SRK				PR			
$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}
0,937	0,073	0,063	0,927	0,939	0,075	0,061	0,925	0,94	0,07	0,06	0,93	0,94	0,08	0,06	0,92
0,790	0,033	0,210	0,967	0,795	0,033	0,205	0,967	0,79	0,03	0,21	0,97	0,79	0,03	0,21	0,97
0,793	0,072	0,207	0,928	0,797	0,071	0,203	0,929	0,79	0,07	0,21	0,93	0,79	0,07	0,21	0,93

Tabla 19: Comparativa de las composiciones en cada fase para el sistema equimolar Tolueno-Amoniaco a distintas presiones (1, 5 y 10 atm).

Errores Composiciones															
SRK				PR				SRK				PR			
$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}	$x_{Tolueno}$	$y_{Tolueno}$	x_{NH3}	y_{NH3}
-0,003	0,003	0,003	-0,003	-0,001	-0,005	0,001	0,005	-0,32%	4,11%	4,76%	-0,32%	-0,11%	-6,67%	1,64%	0,54%
0,000	0,003	0,000	-0,003	0,005	0,003	-0,005	-0,003	0,00%	9,09%	0,00%	-0,31%	0,63%	9,09%	-2,44%	-0,31%
0,003	0,002	-0,003	-0,002	0,007	0,001	-0,007	-0,001	0,38%	2,78%	-1,45%	-0,22%	0,88%	1,41%	-3,45%	-0,11%

Tabla 20: Comparativa de los errores de las fracciones de vapor para el sistema equimolar Tolueno-Amoniaco a distintas presiones (1, 5 y 10 atm).

- Mezclas (Temperatura de saturación).

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Agua (0,33) Benceno (0,33) Butanol (0,33)	1	0	74,57	74,60	-0,03	-0,04%	73,19	72,83	0,36	0,49%
		1	95,95	95,39	0,56	0,58%	95,09	94,32	0,77	0,81%
	5	0	131,84	131,95	-0,11	-0,08%	130,89	130,70	0,19	0,15%
		1	149,49	148,96	0,53	0,35%	149,34	148,59	0,75	0,50%
	10	0	163,63	163,82	-0,19	-0,12%	162,87	162,82	0,05	0,03%
		1	179,45	179,01	0,44	0,25%	179,53	178,88	0,65	0,36%

Tabla 21: Comparativa de las T de rocío y burbuja para el sistema equimolar Agua-Benceno-Butanol a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Octano(0,25) Propanol(0,25) Amoniaco(0,25) m-Xileno(0,25)	1	0	-8,22	-6,76	-1,46	17,76%	-9,77	-8,43	-1,34	13,72%
		1	110,48	112,04	-1,56	-1,41%	110,06	111,45	-1,39	-1,26%
	5	0	50,57	57,30	-6,73	-13,31%	49,08	55,81	-6,73	-13,71%
		1	171,02	172,26	-1,24	-0,73%	171,25	172,27	-1,02	-0,60%
	10	0	87,89	105,07	-17,18	-19,55%	86,25	103,77	-17,52	-20,31%
		1	203,69	204,03	-0,34	-0,17%	204,04	204,10	-0,06	-0,03%

Tabla 22: Comparativa de las T de rocío y burbuja para el sistema equimolar Octano-Amoniaco-Propanol-m-Xileno a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
THF(0,2)	1	0	-64,27	-63,44	-0,83	1,29%	-71,20	-70,71	-0,49	0,69%
Octano(0,2)		1	81,82	85,64	-3,82	-4,67%	81,51	81,29	0,22	0,27%
Etano(0,2)	5	0	-18,10	-12,37	-5,73	31,66%	-25,43	-19,50	-5,93	23,32%
Ciclohexano(0,2)		1	133,98	136,39	-2,41	-1,80%	134,16	132,85	1,31	0,98%
HCl(0,2)	10	0	10,28	23,37	-13,09	-127,33%	2,66	13,28	-10,62	-399,25%
		1	161,60	157,93	3,67	2,27%	161,81	154,68	7,13	4,41%

Tabla 23: Comparativa de las T de rocío y burbuja para el sistema equimolar Octano-THF-Etanol-Ciclohexano-HCL a distintas presiones.

			SRK				PR			
	P (atm)	Φ	Aspen	Programa	E. absoluto	Error (%)	Aspen	Programa	E. absoluto	Error (%)
Agua(0,16)	1	0	4,39	4,82	-0,43	-9,79%	3,80	3,36	0,44	11,58%
R22(0,16)		1	155,26	154,28	0,98	0,63%	156,35	157,48	-1,13	-0,72%
Benceno(0,16)	5	0	71,34	72,04	-0,70	-0,98%	67,99	67,03	0,96	1,41%
Decano(0,16)		1	215,19	214,82	0,37	0,17%	219,07	219,44	-0,37	-0,17%
Naftaleno(0,16)	10	0	117,96	118,12	-0,16	-0,14%	107,11	106,18	0,93	0,87%
CCl4(0,16)		1	244,38	244,03	0,35	0,14%	252,16	252,34	-0,18	-0,07%

Tabla 24: Comparativa de las T de rocío y burbuja para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones.

- Mezclas (fracción de vapor y composiciones)

			SRK				PR			
	P (atm)	T (° C)	Aspen	Programa	Error absoluto	Error (%)	Aspen	Programa	Error absoluto	Error (%)
Agua (0,33)	1,0	80	0,353	0,350	0,003	0,85%	0,403	0,410	-0,007	-1,74%
Benceno	5,0	140	0,479	0,470	0,009	1,88%	0,504	0,510	-0,006	-1,19%
Butanol (0,33)	10,0	165	0,097	0,100	-0,003	-3,09%	0,141	0,160	-0,019	-13,48%

Tabla 25: Comparativa de la fracción de vapor para el sistema equimolar Agua-Benceno-Butanol a distintas presiones.

Aspen											
SRK						PR					
x_{Agua}	y_{Agua}	$x_{Benceno}$	$y_{Benceno}$	$x_{Butanol}$	$y_{Butanol}$	x_{Agua}	y_{Agua}	$x_{Benceno}$	$y_{Benceno}$	$x_{Butanol}$	$y_{Butanol}$
0,219	0,542	0,327	0,346	0,454	0,112	0,198	0,534	0,327	0,343	0,475	0,123
0,205	0,473	0,339	0,327	0,455	0,201	0,195	0,470	0,342	0,325	0,463	0,206
0,310	0,553	0,339	0,283	0,352	0,163	0,298	0,551	0,342	0,281	0,360	0,168
Programa											
SRK						PR					
x_{Agua}	y_{Agua}	$x_{Benceno}$	$y_{Benceno}$	$x_{Butanol}$	$y_{Butanol}$	x_{Agua}	y_{Agua}	$x_{Benceno}$	$y_{Benceno}$	$x_{Butanol}$	$y_{Butanol}$
0,22	0,54	0,33	0,34	0,45	0,11	0,19	0,53	0,33	0,34	0,48	0,12
0,21	0,47	0,34	0,33	0,45	0,2	0,19	0,47	0,34	0,33	0,47	0,21
0,31	0,55	0,34	0,28	0,35	0,16	0,29	0,55	0,34	0,28	0,36	0,17

Tabla 26: Comparativa de las composiciones en cada fase para el sistema equimolar Agua-Benceno-Butanol a distintas presiones (1 , 5 y 10atm).

Error Absoluto											
SRK						PR					
-0,001	0,002	-0,003	0,006	0,004	0,002	0,008	0,004	-0,003	0,003	-0,005	0,003
-0,005	0,003	-0,001	-0,003	0,005	0,001	0,005	0,000	0,002	-0,005	-0,007	-0,004
0,000	0,003	-0,001	0,003	0,002	0,003	0,008	0,001	0,002	0,001	0,000	-0,002
Error (%)											
SRK						PR					
-0,46%	0,37%	-0,92%	1,73%	0,88%	1,79%	4,04%	0,75%	-0,92%	0,87%	-1,05%	2,44%
-2,44%	0,63%	-0,29%	-0,92%	1,10%	0,50%	2,56%	0,00%	0,58%	-1,54%	-1,51%	-1,94%
0,00%	0,54%	-0,29%	1,06%	0,57%	1,84%	2,68%	0,18%	0,58%	0,36%	0,00%	-1,19%

Tabla 27: Comparativa de los errores de las composiciones en cada fase para el sistema equimolar Agua-Benceno-Butanol a distintas presiones (1 , 5 y 10atm).

			SRK				PR			
Octano(0,25)	P (atm)	T (º C)	Aspen	Programa	Error absoluto	Error (%)	Aspen	Programa	Error absoluto	Error (%)
Propanol(0,25)	1,0	50	0,235	0,230	0,005	2,13%	0,238	0,240	-0,002	-0,84%
Amoniaco(0,25)	5,0	90	0,168	0,170	-0,002	-1,19%	0,172	0,170	0,002	1,16%
m-Xileno(0,25)	10,0	120	0,135	0,130	0,005	3,70%	0,141	0,140	0,001	0,71%

Tabla 28: Comparativa de la fracción de vapor para el sistema equimolar Octano-Propanol-Benceno-Amoniaco-m-Xileno a distintas presiones.

Aspen															
SRK								PR							
x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$	x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$
0,318	0,027	0,310	0,054	0,048	0,908	0,323	0,011	0,319	0,028	0,310	0,058	0,046	0,902	0,324	0,012
0,295	0,027	0,289	0,058	0,119	0,903	0,298	0,013	0,296	0,027	0,289	0,060	0,115	0,899	0,299	0,013
0,284	0,036	0,277	0,079	0,153	0,867	0,286	0,019	0,285	0,036	0,278	0,080	0,149	0,864	0,288	0,020

Tabla 29: Composiciones en cada fase para el sistema equimolar Octano-Propanol-Benceno-Amoniaco-m-Xileno a distintas presiones (1, 5 y 10 atm) según Aspen.

Programa															
SRK								PR							
x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$	x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$
0,32	0,03	0,31	0,05	0,05	0,91	0,32	0,01	0,32	0,03	0,31	0,06	0,05	0,90	0,32	0,01
0,29	0,03	0,29	0,06	0,12	0,90	0,30	0,01	0,30	0,03	0,29	0,06	0,12	0,90	0,30	0,02
0,28	0,04	0,28	0,08	0,15	0,86	0,29	0,02	0,28	0,04	0,28	0,08	0,15	0,86	0,29	0,02

Tabla 30: Composiciones en cada fase para el sistema equimolar Octano-Propanol-Benceno-Amoniaco-m-Xileno a distintas presiones (1, 5 y 10 atm) según la aplciación.

Error Absoluto															
SRK								PR							
x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$	x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$
-0,002	-0,003	0,000	0,004	-0,002	-0,002	0,003	0,001	-0,001	-0,002	0,000	-0,002	-0,004	0,002	0,004	0,002
0,005	-0,003	-0,001	-0,002	-0,001	0,003	-0,002	0,003	-0,004	-0,003	-0,001	0,000	-0,005	-0,001	-0,001	-0,007
0,004	-0,004	-0,003	-0,001	0,003	0,007	-0,004	-0,001	0,005	-0,004	-0,002	0,000	-0,001	0,004	-0,002	0,000

Tabla 31: Comparativa de los errores de las composiciones en cada fase para el sistema equimolar Octano-Propanol-Benceno-Amoniaco-m-Xileno a distintas presiones (1, 5 y 10 atm).

Error (%)															
SRK								PR							
x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$	x_{Octano}	y_{Octano}	$x_{Propanol}$	$y_{Propanol}$	$x_{Amoniaco}$	$y_{Amoniaco}$	$x_{m-Xileno}$	$y_{m-Xileno}$
-0,63%	-11,11%	0,00%	7,41%	-4,17%	-0,22%	0,93%	9,09%	-0,31%	-7,14%	0,00%	-3,45%	-8,70%	0,22%	1,23%	16,67%
1,69%	-11,11%	-0,35%	-3,45%	-0,84%	0,33%	-0,67%	23,08%	-1,35%	-11,11%	-0,35%	0,00%	-4,35%	-0,11%	-0,33%	-53,85%
1,41%	-11,11%	-1,08%	-1,27%	1,96%	0,81%	-1,40%	-5,26%	1,75%	-11,11%	-0,72%	0,00%	-0,67%	0,46%	-0,69%	0,00%

Tabla 32: Comparativa de los errores porcentuales de las composiciones en cada fase para el sistema equimolar Octano-Propanol-Benceno-Amoniaco-m-Xileno a distintas presiones (1, 5 y 10 atm).

Octano(0,2) THF(0,2) Etano(0,2) Ciclohexano(0,2) HCl(0,2)			SRK				PR			
	P (atm)	T (° C)	Aspen	Programa	Error absoluto	Error (%)	Aspen	Programa	Error	Error (%)
	1,0	10	0,401	0,420	-0,019	-4,74%	0,411	0,440	-0,029	-7,06%
	5,0	90	0,510	0,550	-0,040	-7,84%	0,520	0,560	-0,040	-7,69%
	10,0	100	0,414	0,450	-0,036	-8,70%	0,429	0,460	-0,031	-7,23%

Tabla 33: Comparativa de la fracción de vapor para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones.

ASPEN																			
SRK										PR									
x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}	x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}
0,307	0,040	0,332	0,003	0,028	0,456	0,314	0,030	0,019	0,471	0,310	0,043	0,337	0,003	0,017	0,462	0,317	0,032	0,018	0,461
0,271	0,132	0,377	0,030	0,034	0,359	0,294	0,110	0,024	0,369	0,272	0,133	0,383	0,031	0,024	0,362	0,297	0,110	0,024	0,363
0,274	0,095	0,326	0,021	0,063	0,393	0,287	0,077	0,049	0,414	0,278	0,096	0,333	0,022	0,049	0,401	0,292	0,078	0,047	0,403

Tabla 34: Composiciones en cada fase para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones (1, 5 y 10 atm) según Aspen.

PROGRAMA																			
SRK										PR									
x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}	x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}
0,28	0,09	0,35	0,00	0,03	0,45	0,33	0,02	0,02	0,45	0,28	0,10	0,35	0,00	0,02	0,44	0,34	0,02	0,02	0,44
0,21	0,19	0,41	0,03	0,03	0,34	0,33	0,10	0,02	0,34	0,21	0,19	0,41	0,03	0,02	0,34	0,33	0,10	0,02	0,34
0,24	0,15	0,34	0,02	0,06	0,37	0,31	0,07	0,05	0,39	0,24	0,15	0,35	0,02	0,05	0,38	0,31	0,07	0,05	0,38

Tabla 35: Composiciones en cada fase para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones (1, 5 y 10 atm) según la aplicación.

Error Absoluto																			
SRK										PR									
x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohex}$	$y_{Ciclohex}$	x_{HCL}	y_{HCL}	x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohex}$	$y_{Ciclohex}$	x_{HCL}	y_{HCL}
0,027	-0,050	-0,018	0,003	-0,002	0,006	-0,016	0,010	-0,001	0,021	0,030	-0,057	-0,013	0,003	-0,003	0,022	-0,023	0,012	-0,002	0,021
0,061	-0,058	-0,033	0,000	0,004	0,019	-0,036	0,010	0,004	0,029	0,062	-0,057	-0,027	0,001	0,004	0,022	-0,033	0,010	0,004	0,023
0,034	-0,055	-0,014	0,001	0,003	0,023	-0,023	0,007	-0,001	0,024	0,038	-0,054	-0,017	0,002	-0,001	0,021	-0,018	0,008	-0,003	0,023

Tabla 36: Comparativa de los errores de las composiciones en cada fase para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones (1, 5 y 10 atm).

Error (%)									
SRK									
x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}
8,79%	-125,00%	-5,42%	100,00%	-7,14%	1,32%	-5,10%	33,33%	-5,26%	4,46%
22,51%	-43,94%	-8,75%	0,00%	11,76%	5,29%	-12,24%	9,09%	16,67%	7,86%
12,41%	-57,89%	-4,29%	4,76%	4,76%	5,85%	-8,01%	9,09%	-2,04%	5,80%

Tabla 37: Comparativa de los errores porcentuales de las composiciones en cada fase para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones (1, 5 y 10 atm) para la ecuación de SRK.

Error (%)									
PR									
x_{THF}	y_{THF}	x_{Octano}	y_{Octano}	x_{Etanol}	y_{Etanol}	$x_{Ciclohexano}$	$y_{Ciclohexano}$	x_{HCL}	y_{HCL}
9,68%	-132,56%	-3,86%	100,00%	-17,65%	4,76%	-7,26%	37,50%	-11,11%	4,56%
22,79%	-42,86%	-7,05%	3,23%	16,67%	6,08%	-11,11%	9,09%	16,67%	6,34%
13,67%	-56,25%	-5,11%	9,09%	-2,04%	5,24%	-6,16%	10,26%	-6,38%	5,71%

Tabla 38: Comparativa de los errores porcentuales de las composiciones en cada fase para el sistema equimolar THF-Ciclohexano-Octano-Etanol-HCL a distintas presiones (1, 5 y 10 atm) para la ecuación de PR.

Agua(0,16) R22(0,16) Benceno(0,16) Decano(0,16) Naftaleno(0,16) CCl4(0,16)			SRK				PR			
	P (atm)	T (° C)	Aspen	Programa	Error absoluto	Error (%)	Aspen	Programa	Error	Error (%)
	1,0	100	0,586	0,590	-0,004	-0,68%	0,594	0,590	0,004	0,67%
	5,0	170	0,613	0,610	0,003	0,49%	0,618	0,620	-0,002	-0,32%
	10,0	200	0,569	0,570	-0,001	-0,18%	0,574	0,570	0,004	0,70%

Tabla 39: Comparativa de la fracción de vapor para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones.

Aspen											
SRK											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
0,038	0,257	0,007	0,279	0,115	0,203	0,343	0,042	0,386	0,012	0,110	0,206
0,053	0,238	0,016	0,261	0,122	0,195	0,307	0,078	0,382	0,031	0,120	0,196
0,072	0,238	0,028	0,272	0,141	0,186	0,280	0,081	0,340	0,035	0,139	0,187

Tabla 40: Composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) según Aspen y en base a la ecuación de SRK.

Aspen											
PR											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
0,035	0,257	0,007	0,276	0,114	0,203	0,344	0,045	0,391	0,013	0,109	0,206
0,050	0,239	0,017	0,259	0,123	0,194	0,307	0,080	0,384	0,032	0,120	0,195
0,068	0,240	0,028	0,269	0,141	0,186	0,280	0,082	0,342	0,036	0,140	0,187

Tabla 41: Composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) según Aspen y en base a la ecuación de PR.

Programa											
SRK											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
0,04	0,26	0,01	0,28	0,12	0,20	0,34	0,04	0,36	0,01	0,11	0,21
0,05	0,24	0,02	0,26	0,12	0,16	0,31	0,05	0,38	0,03	0,12	0,20
0,07	0,24	0,03	0,27	0,14	0,19	0,28	0,08	0,34	0,04	0,14	0,19

Tabla 42: Composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) según la aplicación y en base a la ecuación de SRK.

Programa											
PR											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
0,03	0,26	0,01	0,28	0,11	0,20	0,34	0,05	0,39	0,01	0,11	0,21
0,05	0,24	0,02	0,26	0,12	0,19	0,31	0,08	0,38	0,03	0,12	0,19
0,04	0,24	0,03	0,27	0,14	0,19	0,28	0,08	0,34	0,04	0,14	0,19

Tabla 43: Composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) según la aplicación y en base a la ecuación de PR.

Error Absoluto											
SRK											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
-0,002	-0,003	-0,003	-0,001	-0,005	0,003	0,003	0,002	0,026	0,002	0,000	-0,004
0,003	-0,002	-0,004	0,001	0,002	0,035	-0,003	0,028	0,002	0,001	0,000	-0,004
0,002	-0,002	-0,002	0,002	0,001	-0,004	0,000	0,001	0,000	-0,005	-0,001	-0,003

Tabla 44: Errores cometidos en las composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) en base a la ecuación de SRK.

Error (%)											
SRK											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
-5,26%	-1,17%	-42,86%	-0,36%	-4,35%	1,48%	0,87%	4,76%	6,74%	16,67%	0,00%	-1,94%
5,66%	-0,84%	-25,00%	0,38%	1,64%	17,95%	-0,98%	35,90%	0,52%	3,23%	0,00%	-2,04%
2,78%	-0,84%	-7,14%	0,74%	0,71%	-2,15%	0,00%	1,23%	0,00%	-14,29%	-0,72%	-1,60%

Tabla 45: Errores porcentuales cometidos en las composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) en base a la ecuación de SRK.

Error Absoluto											
PR											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
0,005	-0,003	-0,003	-0,004	0,004	0,003	0,004	-0,005	0,001	0,003	-0,001	-0,004
0,000	-0,001	-0,003	-0,001	0,003	0,004	-0,003	0,000	0,004	0,002	0,000	0,005
0,028	0,000	-0,002	-0,001	0,001	-0,004	0,000	0,002	0,002	-0,004	0,000	-0,003

Tabla 46: Errores cometidos en las composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) en base a la ecuación de PR.

Error (%)											
PR											
x_{Agua}	y_{Agua}	x_{R22}	y_{R22}	$x_{Benceno}$	$y_{Benceno}$	x_{Decano}	y_{Decano}	$x_{Naftaleno}$	$y_{Naftaleno}$	x_{CCl4}	y_{CCl4}
16,67%	-1,15%	-30,00%	-1,43%	3,64%	1,50%	1,18%	-10,00%	0,26%	30,00%	-0,91%	-1,90%
0,00%	-0,42%	-15,00%	-0,38%	2,50%	2,11%	-0,97%	0,00%	1,05%	6,67%	0,00%	2,63%
70,00%	0,00%	-6,67%	-0,37%	0,71%	-2,11%	0,00%	2,50%	0,59%	-10,00%	0,00%	-1,58%

Tabla 47: Errores porcentuales cometidos en las composiciones de cada fase para el sistema equimolar Agua-R22-Benceno-Decano-Naftaleno-CCl4 a distintas presiones (1, 5 y 10 atm) en base a la ecuación de PR.

- Propiedades energéticas.

Agua	SRK				PR			
1 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	2165	2257	-92	-4,10%	2047	2257	-210	-9,30%
ΔSv(kJ/kgK)	4,11	6,00	-1,89	-31,53%	4,20	6,00	-1,80	-30,00%
ΔUv(kJ/kg)	2163	2088	75	3,61%	2045	2088	-43	-2,04%
ρ(kg/m³)	710	958	-248	-25,87%	801	958	-157	-16,37%

Agua	PRVT			
1 atm	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	895	958	-63	-6,57%

Tabla 48: Comparativa de la entalpía, entropía, energía interna de vaporización y densidad (líquido) para el agua a 1 atm.

Agua	SRK				PR			
10 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	1835	2015	-180	-8,93%	1758	2015	-257	-12,76%
ΔSv(kJ/kgK)	2,90	4,45	-1,55	-34,83%	2,98	4,45	-1,48	-33,15%
ΔUv(kJ/kg)	1833	1821	12	0,66%	1756	1821	-65	-3,57%
ρ(kg/m³)	647	887	-240	-27,09%	714	887	-173	-19,50%

Agua	PRVT			
10 atm	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	868	887	-19	-2,14%

Tabla 49: Comparativa de la entalpía, entropía, energía interna de vaporización y densidad (líquido) para el agua a 10 atm.

Amoniaco	SRK				PR			
1 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	1330	1366	-36	-2,65%	1263	1366	-103	-7,51%
ΔSv(kJ/kgK)	4,00	5,70	-1,70	-29,82%	4,05	5,70	-1,65	-28,88%
ρ(kg/m³)	533	680	-147	-21,65%	602	680	-79	-11,55%

	PRVT			
	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	657	680	-23	-3,38%

Tabla 50: Comparativa de la entalpía, entropía de vaporización y densidad (líquido) para el amoniaco a 1 atm.

Amoniaco	SRK				PR			
10 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	1094	1162	-68	-5,83%	1055	1162	-107	-9,21%
ΔSv(kJ/kgK)	2,65	3,93	-1,28	-32,61%	2,72	3,93	-1,21	-30,70%
ρ(kg/m³)	470	601	-131	-21,76%	533	601	-68	-11,39%

	PRVT			
	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	634	601	-33	5,49%

Tabla 51: Comparativa de la entalpía, entropía de vaporización y densidad (líquido) para el amoniaco a 10 atm.

R-12	SRK				PR			
10 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	119	121	-3	-2,16%	116	121	-6	-4,74%
ΔSv(kJ/kgK)	0,28	0,38	-0,10	-26,69%	0,28	0,38	-0,09	-24,23%
ρ(kg/m³))	1115	1211	-96	-7,92%	1262	1211	52	4,28%

PRVT				
	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	1633	1211	-422	-34,84%

Tabla 52: Comparativa de la entalpía, entropía de vaporización y densidad (líquido) para el R12 a 10 atm.

R-12	SRK				PR			
1 atm	Programa	Experimental	Error Absoluto	Error (%)	Programa	Experimental	Error Absoluto	Error (%)
ΔHv(kJ/kg)	161	165	-4	-2,26%	154	165	-11	-6,59%
ΔSv(kJ/kgK)	0,48	0,68	-0,20	-29,26%	0,49	0,68	-0,19	-27,57%
ρ(kg/m³)	1376	1488	-113	-7,56%	1556	1488	67	4,53%

PRVT				
	Programa	Experimental	Error Absoluto	Error (%)
ρ(kg/m³)	1687	1488	-199	-13,37%

Tabla 53: Comparativa de la entalpía, entropía de vaporización y densidad (líquido) para el R12 a 1 atm.

3.7. Arquitectura

Una aplicación está formada por ficheros de programación denominados 'clases' (class), que poseen un archivo de extensión '.xml' asociado en caso de que tengan que realizar algún tipo de interacción con el usuario.

En el siguiente diagrama (Diagrama 3) se muestra la relación entre las distintas clases con líneas continuas si poseen un fichero '.xml' asociado o discontinuas si son de apoyo para las anteriores.

La clase principal, a la que también se puede denominar como 'padre', es la "InicialLabel", un splash que aparece nada más ejecutar la aplicación, y que tras 3 segundos de demora, da paso a la "MainActivity", a partir de la cual se pueden acceder a varias clases hijas: licencia ("license"), base de datos ("listcomp"), fundamento teórico ("tabla"), datos de operación ("datos"), gráficas ("graphic") y clausura de la aplicación ("closer").

- La clase "license" muestra la licencia de la aplicación, y permite regresar a la clase "MainActivity".

- La clase "closer" finaliza la ejecución de la aplicación, y deriva en un splash de despedida antes de volver a la pantalla de inicio del dispositivo Android empleado.

Las clases "tabla", "datos", "graphic" y "listcomp", son a su vez, padres de otras clases.

- La clase "tabla" es padre de "eoshist" (dónde se expone la teoría referente a las EOS), "epchist" (donde queda recogida la base teórica del cálculo de las propiedades energéticas), y "lvehist" (dónde se explica cómo se ha procedido para el cálculo de equilibrio líquido vapor).

- La clase "datos" es padre de "componentes", donde se introducirán los componentes que forman la mezcla a resolver. A su vez, la clase "componentes" es padre de la clase "resultados", donde el usuario podrá consultar los valores que genera el programa para la mezcla introducida con las condiciones especificadas.

- La clase “listcomp” muestra la base de datos empleada, y a su vez es padre de “carcomp”, clase en la que se pueden ver las características del componente seleccionado. La clase “carcomp” puede derivar en otras 2 clases: “nuevo” (activity que solicita los datos necesarios para añadir un nuevo componente a la base de datos) y “edit” (permite la edición del componente seleccionado). También permite retroceder a la clase “listcomp” y eliminar un registro.

- La clase “graphic” permite seleccionar el tipo de ecuación que se empleará para graficar un componente puro, y es padre de la clase “drawpure”, en la que se podrá seleccionar el componente a graficar y el tipo de gráfica deseada, así como el rango de su eje de abscisas.

Por último, las clases auxiliares son: “kij” (coeficientes de interacción binaria), “EOS_Utils” (normalizador), “EOS” (expresiones matemáticas de cálculo), “SQLiteHelper” (se definen los parámetros para crear una base de datos SQL), “SQLite” (genera las funciones llamadas a lo largo del programa cuando se quiera utilizar la base de datos), “Lienzop1” y “Lienzop3” (superficies de dibujo para canvas, que permiten el uso de las gráficas).

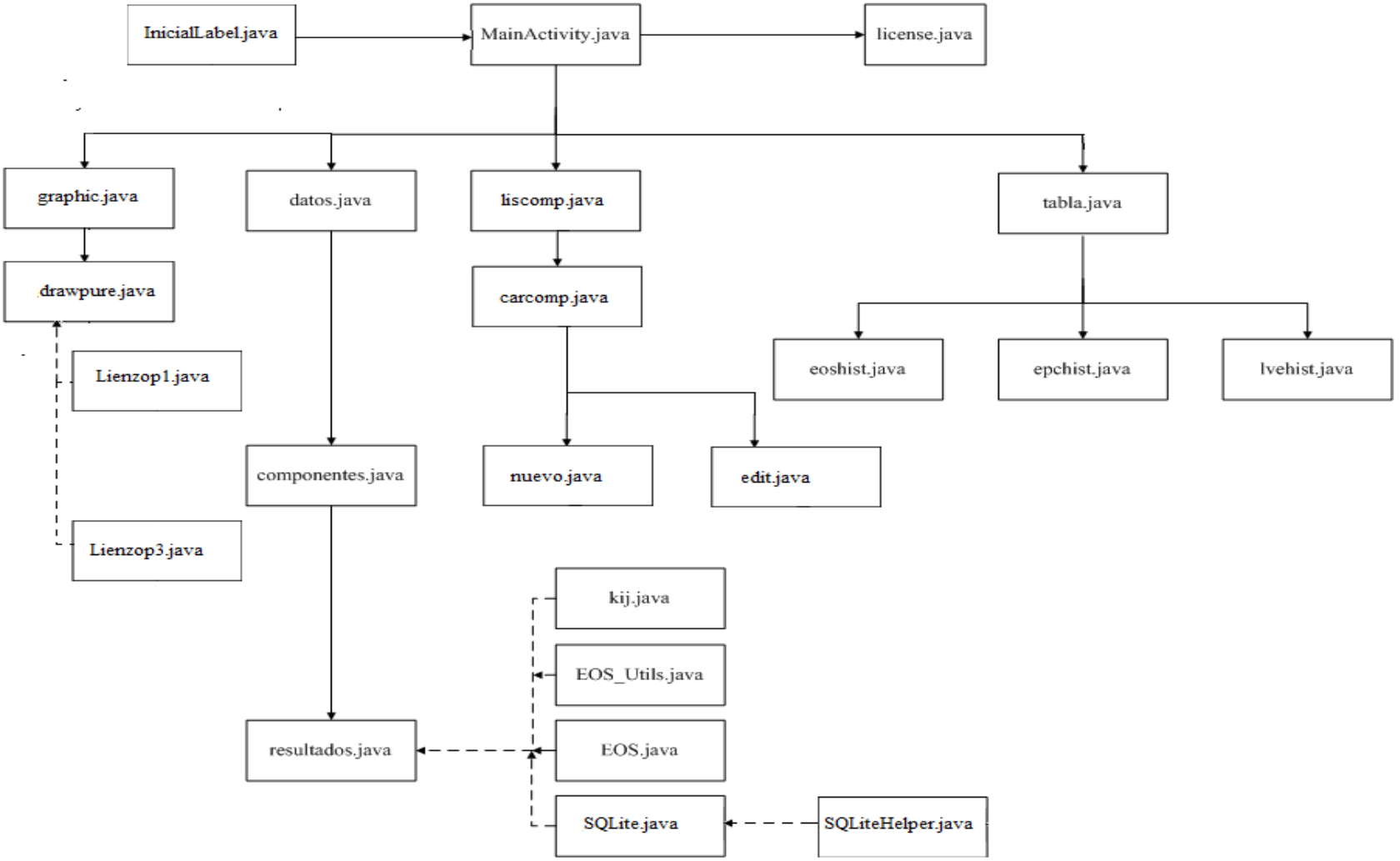


Diagrama 4: arquitectura de Calculator of Thermodynamic Properties.

4. Simbología

4.1. Calculator of thermodynamic propierties

-P es la presión en atm.

-V es el volumen el litros del recipiente.

-a es el parámetro de la interacción entre partículas en atm· L^2/mol^2.

-b o covolumen es el volumen disponible de un mol de partículas en L/mol.

-n es el número de moles.

-R es la constante de los gases ideales.

-T es la temperatura en K.

-Tc es la temperatura crítica en K

-Pc es la presión crítica en atm

-w es el factor acéntrico (adim.).

-Tr la temperatura reducida (adim.).

-Pr es la presión reducida (adim.).

-k_{ij} son los parámetros de interacción binaria (adim.).

-ρ es la densidad molar en mol/L.

-Z es el factor de compresibilidad (adim.).

- *f* es la fugacidad en atm.

-φ es el coeficiente de fugacidad (adim.).

-K es la constante de equilibrio (adim.).

-x, y es la fracción molar en la fase líquida y gaseosa (adim).

-H es la entalpía en J/mol.

-S es la entropía en J/molK.

-U es la energía interna en J/mol.

-G es la energía libre de Gibss en J/mol.

-B es la Exergía en J/mol.

-t es el traslado que sufre el factor de compresibilidad en PRVT.

5. Conclusiones

A través de la elaboración del presente Trabajo de Fin de Grado se ha pretendido elaborar una aplicación para dispositivos Android capaz de calcular las propiedades termodinámicas de diversos componentes, así como de implementar sus diagramas de fase. Dicha aplicación cuenta con una serie de herramientas con diversas utilidades, desde almacenamiento en una base de datos externa hasta explicación teórica de los procesos empleados en la misma. Para ello se ha seguido un procedimiento que se ha ido detallando a lo largo de la elaboración del presente documento.

Las características de partida de la aplicación exigen llevar a cabo un estudio cuidadoso de las alternativas existentes como solución para su implementación, teniendo en cuenta los requisitos que precisan este tipo de aplicaciones. Para abordar dicho estudio, se han analizado detalladamente las diferentes ecuaciones que permiten realizar el cálculo de las propiedades termodinámicas y seleccionado las que mejor se adaptan a las necesidades de los posibles usuarios.

En cuanto a la implementación de las ecuaciones, hay que destacar la importancia de que todos los términos empleados deben tener unas dimensiones apropiadas, para obtener así resultados que sean concluyentes.

Respecto al uso de la base de datos SQL hay que destacar la posibilidad que ofrecen al usuario de modificar la misma a su antojo, que es la razón principal por la que se ha decidido realizar la base en dicho lenguaje. Es importante destacar que la programación en SQL facilita en gran medida la interacción entre el usuario y el programador.

El empleo del elemento Canvas para la generación de las gráficas ha sido decidido por la simplicidad de la implementación del mismo. Si bien es cierto que podrían existir otros métodos para generar gráficas (como el uso de OpenGL), el elemento Canvas posee mayor claridad para el programador y es más sencillo de comprender para el usuario debido a la simplicidad de los diagramas generados.

Las soluciones escogidas tras el proceso de estudio realizado, han sido referentes en un primer lugar al lenguaje de programación y en un segundo a las versiones de Android de las que se ha dispuesto. En cuanto al lenguaje, la interfaz de programación ‘Eclipse IDE’, usualmente empleada para la creación de aplicaciones móviles, da al usuario la posibilidad de emplear el lenguaje Java para la programación de los eventos que se suceden en la aplicación, y el lenguaje XML para definir la estructura de todas las interfaces utilizadas a lo largo de la aplicación. A lo largo de toda la programación, se ha decidido excluir ciertos elementos que pudiesen generar algún tipo de problema para

usuarios con versiones de Android ya obsoletas (por debajo de la 3.0), pero siempre tratando de crear una aplicación lo más genérica posible, que no supusiera un problema para cualquier persona que decida proceder al uso de la misma.

El método empleado para comprobar el funcionamiento de cada una de las herramientas se ha basado en un mecanismo de "prueba y error". A partir de los requisitos de cada herramienta se ha ido desarrollando el código que se consideraba adecuado, y al finalizar su concepción se comprobaba el funcionamiento. En caso de no obtener los resultados deseados, se procedió a realizar las modificaciones oportunas, tratando de que fueran las menores posibles, para así no alejarse demasiado del concepto obtenido inicialmente.

Finalmente, tras llevar a cabo el análisis y desarrollo de la aplicación en cuestión, se puede concluir que las aplicaciones basadas en el sistema Android ofrecen un gran abanico de posibilidades, como se puede comprobar en las aplicaciones móviles actuales. En el presente a través de los dispositivos existentes se pueden observar multitud de aplicaciones con diversas utilidades, que ofrecen a los usuarios prácticamente cualquier cosa que deseen. Es por ello que este tipo de proyectos están a la orden del día y son muy adecuados a lo que la sociedad hoy en día puede demandar y un Ingeniero Electrónico puede ofrecer.

Por tanto, se ha conseguido implementar una aplicación capaz de calcular las propiedades energéticas y de equilibrio de mezclas de hasta 6 componentes a libre elección del usuario, habilitando la posibilidad de la obtención de los diagramas de fase pertinentes, y acceso al procedimiento teórico llevado a cabo.

6. Líneas futuras de trabajo

Al tratarse de una aplicación para dispositivos Android se pueden intensificar una gran cantidad de líneas de trabajo que podrían continuar en el futuro el trabajo desarrollado en este Proyecto de Fin de Carrera, ya sea desde un punto de vista nuevamente académico o empresarial.

Para empezar, habría que centrarse en pequeños aspectos de la aplicación tal y como se entrega que podrían ser mejorables en sucesivas ediciones. Por ejemplo, como se puede observar en el código adjunto en el CD, existen algunos archivos Java que no son empleados en ningún momento. Estos archivos serían los destinados a la consecución de gráficas de compuestos binarios, que no está implementado en esta versión. En el futuro se podría desarrollar la programación oportuna para la obtención de dichas gráficas, lo cual aportaría mayor información de la actual al usuario.

Otro punto que queda en el aire es la posibilidad de que, al pulsar sobre las gráficas obtenidas, se muestre por pantalla el valor exacto de Presión-Temperatura en dicho lugar. Para lograrlo, sería necesario generar gran cantidad de puntos para conseguir aumentar la precisión de las líneas que los unen, lo cual actualmente supondría una gran cantidad de trabajo para la aplicación, y podría demorar varios segundos en obtener resultados. No obstante, es posible que en un futuro cercano las velocidades de procesamiento de los dispositivos Android mejoren sustancialmente, por lo que esto dejaría de suponer un problema para la aplicación.

Un pequeño retoque adicional que se puede aconsejar es la inclusión de una tabla con los datos representados en una pestaña anexa y de un botón que permita al usuario exportar los datos obtenidos, tanto en la calculadora como en las gráficas, a un archivo de texto, que pueda ser almacenado en el dispositivo y empleado por el usuario para lo que considere oportuno, o cargado en la aplicación Dropbox del mismo, puesto que existiría esa posibilidad

En lo que se refiere a términos estrictos de programación, se podría tratar de generar las gráficas mediante el uso de OpenGL en lugar de Canvas. Esto supondría mayor complejidad de programación, pero permitiría al usuario la posibilidad de obtener gráficas mucho más desarrolladas, incluyendo diagramas tridimensionales.

En lo referente al aspecto termodinámico de la aplicación, es posible que dentro de unos años aparezcan ecuaciones que consigan simplificar los cálculos actualmente implementados, pero con lo que se conoce hasta ahora, hay varios aspectos que cabría la posibilidad de mejorar.

Por ejemplo, en ningún momento se tienen en cuenta las inmiscibilidades de los líquidos, lo cual puede suponer una gran variación de los datos obtenidos, al suponer a todos los compuestos como ideales. Cabría la posibilidad de tener en cuenta este aspecto y reflejarlo en los componentes de la aplicación. Para ello, se podría emplear un modelo basado en componentes de actividad (como el NRTL).

Otra posibilidad sería modificar las rutinas de cálculo para poder introducir el equilibrio líquido-gas no condensable, y permitir al usuario elegir la fracción de vapor empleada (actualmente, sólo permite valores de 0 ó 1).

Además, podrían introducirse otro tipo de mejoras que ya no estarían tan al alcance del programador como las anteriores, como podría ser la introducción de ecuaciones, reglas y factores específicos para condiciones supercríticas, la introducción de métodos de contribución de grupos para la estimación de propiedades, o la introducción de métodos de cálculo para propiedades de transporte (viscosidades, conductividades y difusividades).

Por último, también se podría considerar la posibilidad de traducir la aplicación a otros lenguajes, como Windows o Apple, para lograr así 'universalizar' la aplicación.

7. Bibliografía

- Ableson, W. F., R. Sen, et al. (2012). Android in Action. , Manning Publications Co.

- ASPEYO (2012). PRINCIPIOS BÁSICOS DE SEGURIDAD CONTRA INCENDIOS, ASPEYO.

- Attribution, C. C. (2013). "uses-sdk." from http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

- BlueSens (2013). "Productos > Análisis Fisicoquímico > Sensor de conductividad > BlueSens gas sensor." Retrieved 25/2/2013, 2013, from http://www.directindustry.es/prod/bluesens-gas-sensor/sensores-de-gas-per-conductividad-termica-hidrogeno-h2-32049-472738.html.

- Bruce E. Poling, J. M. P., John P. O'Connell (2001). The properties of gases and liquids, McGraw-Hill.

- CENGEL, Y. A. and M. A. BOLES (2001). THERMODYNAMICS. AN ENGINEERING APPROACH, McGraw-Hill.

- DropboxInc. (2013). "Dropbox." from www.DROPBOX.com.

- Dutta, B. K. (2007). Principles Of Mass Transfer And Separation Process, Prentice-Hall of India Pvt.Ltd.

- Esparza, F. (2012). El fuego o Combustión, Bomberos de Navarra.

- Fogler, H. S. (2006). Elements of Chemical Reaction Engineering, Pretince Hall

- Gilbert F. Froment, K. B. B., Juray De Wilde (2010). Chemical Reactor Analysis and Design, Wiley.

- IPICYT, I. P. d. I. C. y. T. (2012). "Laboratorio Analítico." Retrieved 25/2/2013,2013,from

http://www.ipicyt.edu.mx/Ciencias_Ambientales/areas_ciencias_ambientales_lab_analitico.php.

- Lequerica, J. R. (2011). Desarrollo de Aplicaciones para Android, Anaya.

- Levenspiel, O. (1998). Chemical Reaction Engineering, John Wiley and Sons (WIE).

- Meier, R. (2009). Professional Android™ Application Development, Wiley Publishing, Inc.

- Michelsen, M. L. and J. Mollerup. (2007). Thermodynamic Models: Fundamentals and Computational Aspects, Tie-Line Publications.

- Pablo Basanta Val (2012). "Arquitectura Android." Software de Comunicaciones. Retrieved 16/2/2013, 2013, from https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android.

- Pérez, F. P. and D. P. Bendito (1983). Análisis de elementos-traza por espectrofotometría de absorción molecular ultravioleta-visible, Universidad de Sevilla

- Piqueras, C. M., J. García-Serna, et al. (2010). "Estimation of lower flammability limits in high-pressure systems. Application to the direct synthesis of hydrogen peroxide using supercritical and near-critical CO2 and air as diluents." The Journal of Supercritical Fluids.

- Prausnitz, J. M., Lichtenthaler, R.N. y Gomes de Azevedo, (2000). Termodinámica molecular de los equilibrios de fases, Prentice-Hall.

- Quarteroni, A., R. Sacco, et al. (2000). Numerical Mathematics, 3.

- Santamaría, J. M., J. Herguido, et al. (2002). Ingeniería de reactores, SINTESIS.

- Solé, A. C. (2011). Instrumentación Industrial, marcombo.

- Southeastern Automation, I. (2013). "ROSEMOUNT ANALYTICALMEASUREMENT DIVISION." from http://southeastern-automation.com/Files/Emerson/Measurement/measurementprod.html.

- Tapio O. Salmi, J.-P. M., Johan P. Warna (2011). Chemical Reaction Engineering and Reactor Technology, CRC Press.

- Técnicos, O. (2013). "Documentos Técnicos: Instrumentación Industrial. Sesnores de nivel." from http://www.tecnoficio.com/docs/doc60.php.

- Treybal, R. E. (1980). Operaciones de transferencia de masa, Mcgraw Hill.

- VirtualExpo, G. (2013). "Productos > Mediciónes de la Temperatura y Humedad > Sonda de temperatura con termopar > Dalian Bocon Science & Technology Co. Ltd." Retrieved 3/3/2013, 2013, from http://www.directindustry.es/prod/dalian-bocon-science-technology-co-ltd/sondas-de-temperatura-termopar-tipo-j-78582-791985.html.

- Wikimedia Foundation, I. (2013). "Android version history." from http://en.wikipedia.org/wiki/Android_version_history.

- Cabeza Sánchez, Á. (2012). *Cálculo de propiedades termodinámicas y simulación de reactores en tiempo real en el entorno de programación Android.* Universidad de Valladolid.

- Oliver, S. G. Bases de Datos en Android: Primeros pasos [en linea]. Retrieved 12 Agosto, 2014, from http://www.sgoliver.net/blog/?p=1611

- C. J. Date y Hugh Darwen: *A Guide to the SQL standard : a users guide to the standard database language SQL, 4th ed.*, Addison Wesley, USA 1997,

8. Anexos

8.1. Anexo I. Métodos numéricos Utilizados

8.1.1. Resolución de sistemas de ecuaciones no lineales

8.1.1.1. Método de Newton-Raphson

El método de Newton-Raphson es un algoritmo que permite obtener las raíces de una función real o hallar un mínimo de la misma. Es un proceso de optimización iterativo que se basa en aproximar la función a optimizar por medio de la serie de Taylor hasta llegar a orden 2 (Quarteroni, Sacco et al. 2000).

Si se desea obtener un mínimo de la función f(x) de n variables, esta puede aproximarse de la siguiente manera:

$$f(x) \sim m(x) = f(x_o) + (x - x_o) \cdot \nabla f(x_o) + \frac{1}{2} \cdot (x - x_o) \cdot H_{f(x_o)} \cdot (x - x_o) \quad (86)$$

De tal forma que un mínimo relativo de la función m(x) también lo será de la función f(x). Asumiendo que este mínimo es x_1 y, por tanto, $\nabla f(x_1) = \nabla m(x_1) = 0$:

$$\nabla f(x_o) + H_{f(x_o)} \cdot (x_1 - x_o) = 0 \quad (87)$$

Y si el Hessiano tiene inversa se llega a una expresión que se usa como una ecuación de recurrencia para dado un punto inicial generar una sucesión de puntos que deben converger al mínimo local de f(x):

$$x_1 = x_o - \nabla f(x_o) \cdot H_{f(x_o)}^{-1} \quad (88)$$

La principal ventaja de este método es que implica menos iteraciones que sus homólogos pero, cómo desventajas tiene que: precisa un punto inicial cercano a la solución, requiere el cálculo del gradiente, requiere el cálculo del Hessiano y se ha de invertir el Hessiano. Por otro lado, no asegura la convergencia, ya que el Hessiano no tiene por qué ser positivo definido.

- Algoritmo de resolución

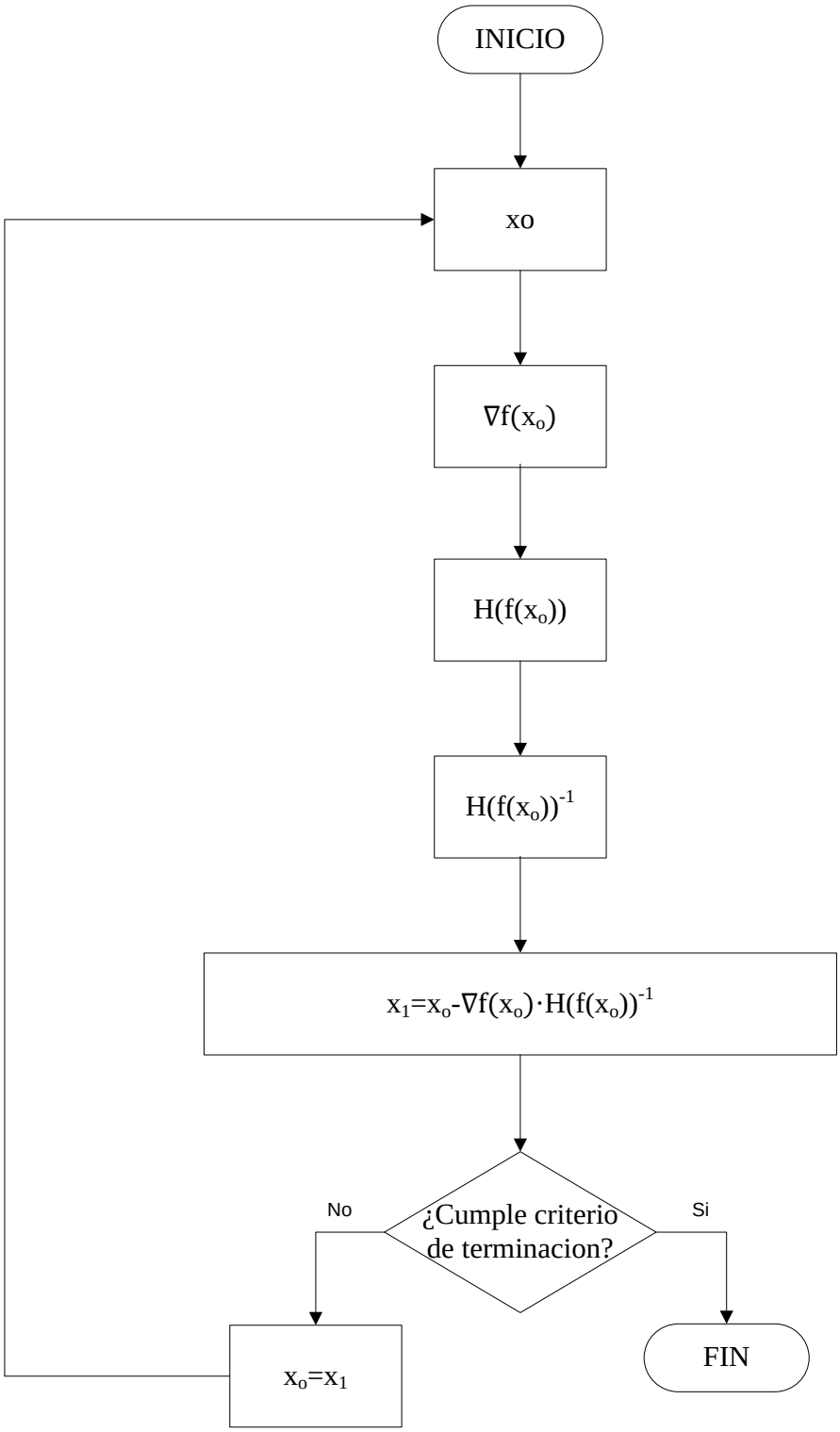


Diagrama 5: algoritmo de resolución del método de Newton-Raphson.

8.2. Anexo II. Obtención de expresiones para el cálculo de propiedades energéticas

La obtención de las expresiones desarrolladas a continuación se hace siguiendo los pasos marcados por la bibliografía aquí especificada (Prausnitz 2000; Bruce E. Poling 2001; CENGEL and BOLES 2001; Michelsen and Mollerup. 2007).

8.2.1. Criterio de exactitud matemática

Dada la expresión:

$$dZ = M \cdot dX + N \cdot dY \tag{89}$$

Si X e Y son funciones de punto (sólo dependen del estado y no de la trayectoria) las diferencias son exactas y se cumple la igualdad:

$$\left(\frac{\partial M}{\partial Y}\right)_X = \left(\frac{\partial N}{\partial X}\right)_Y \tag{90}$$

Lo que puede demostrarse de la siguiente manera:

Recordando que:

$$M = \left(\frac{\partial Z}{\partial X}\right)_Y \tag{91}$$

$$N = \left(\frac{\partial Z}{\partial Y}\right)_X \tag{92}$$

Se llega a qué:

$$\left(\frac{\partial M}{\partial Y}\right)_X = \frac{\partial^2 Z}{\partial X \cdot \partial Y} \tag{93}$$

$$\left(\frac{\partial N}{\partial X}\right)_Y = \frac{\partial^2 Z}{\partial Y \cdot \partial X} \tag{94}$$

Y cómo el orden derivación no importa para funciones de punto:

$$\left(\frac{\partial M}{\partial Y}\right)_X = \frac{\partial^2 Z}{\partial X \cdot \partial Y} = \frac{\partial^2 Z}{\partial Y \cdot \partial X} = \left(\frac{\partial N}{\partial X}\right)_Y \quad (95)$$

8.2.2. Relaciones termodinámicas

Para una sustancia pura se cumple que:

$$H = f(T, P) \quad (96)$$

Por lo tanto:

$$dH = \left(\frac{\partial H}{\partial T}\right)_P \cdot dT + \left(\frac{\partial H}{\partial P}\right)_T \cdot dP \quad (97)$$

Y como:

$$dH = T \cdot dS - v \cdot dP \quad (98)$$

Se deduce la siguiente relación:

$$\left(\frac{\partial H}{\partial T}\right)_P = v + T \cdot \left(\frac{\partial S}{\partial P}\right)_T \quad (99)$$

Aplicando ahora la siguiente relación de Maxwell:

$$\left(\frac{\partial S}{\partial P}\right)_T = -\left(\frac{\partial v}{\partial T}\right)_P \quad (100)$$

Y sabiendo que la capacidad calorífica a presión constante se define como:

$$C_p = \left(\frac{\partial H}{\partial T}\right)_P = T \cdot \left(\frac{\partial S}{\partial T}\right)_P \quad (101)$$

Resultando que la variación de entalpía se define como:

$$dH = C_p \cdot dT + \left[v - T \cdot \left(\frac{\partial v}{\partial T}\right)_P\right] \cdot dP \quad (102)$$

Por otro lado, para el caso de la entropía:

$$S = f(T, P) \quad (103)$$

$$dS = \left(\frac{\partial S}{\partial T}\right)_P \cdot dT + \left(\frac{\partial S}{\partial P}\right)_T \cdot dP \quad (104)$$

Y aplicando el mismo procedimiento, la variación de entropía resulta:

$$dS = \frac{C_p}{T} \cdot dT - T \cdot \left(\frac{\partial v}{\partial T}\right)_P \cdot dP \quad (105)$$

8.2.3. Cálculo del coeficiente de fugacidad

A temperatura constante, la variación de la energía de Gibbs de una sustancia pura con la presión viene dada por:

$$dG = VdP \quad (106)$$

Sabiendo que el potencial químico de una sustancia pura se define como $\mu = G / n$, la ecuación se convierte en:

$$d\mu = vdP \quad (107)$$

Si el sistema consiste en un gas ideal, existe una expresión analítica para el volumen molar y la ecuación (2) puede integrarse para dar:

$$\mu_{id}(T, P) = \mu_o(T) + R \cdot T \cdot ln\frac{P}{P_0} \quad (108)$$

Donde μ (el potencial químico a P_0) es una constante de integración que depende de las variables que se han mantenido constantes (T).

Ahora bien, esta expresión sólo tiene validez para gases ideales, para poder generalizarla para el caso real se ha de sustituir la presión por la fugacidad.

$$\mu_{id}(T,P) = \mu_o(T) + R \cdot T \cdot ln\frac{f(T,P)}{P_0} \quad (109)$$

$$f(T,P) = \varphi(T,P) \cdot P \quad (110)$$

Diferenciando esta ecuación:

$$d\mu = R \cdot T \cdot dlnf \quad (111)$$

Sustituyendo la definición de *dμ*:

$$dlnf = \frac{v}{R \cdot T} \cdot dP \quad (112)$$

Restando *lnP* en ambos miembros de la ecuación anterior:

$$dln\left(\frac{f}{P}\right) = \left(\frac{v}{R \cdot T} - \frac{1}{P}\right) \cdot dP \quad (113)$$

E introduciendo la definición de la fugacidad se llega a:

$$R \cdot T \cdot dln(\varphi) = \left[v - R \cdot \frac{T}{P}\right] \cdot dP \quad (114)$$

8.3. Anexo III. Componentes presentes en la aplicación

Los componentes disponibles en la aplicación Calculator of Energy Properties quedan recogidos en la Tabla 54. En total son 78 sustancias, seleccionadas por ser representativas de todas las que puedan tener interés para los estudiantes de termodinámica (orgánicos polares y apolares, inorgánicos, agua, gases no condensables…). Los valores de las propiedades han sido obtenidos de la base de datos suministrada en la asignatura de "Introducción a la ingeniería química".

1	METHANE	21	C2H4	41	C3H8O-01	61	N2O
2	ETHANE	22	C3H6--01	42	C3H8O-02	62	O2S
3	PROPANE	23	C4H8--01	43	C4H10-02	63	O3S
4	BUTANE	24	C4H8--02	44	C4H10-03	64	O2-02
5	C4H10-01	25	C5H10-02	45	C4H8O-01	65	H2
6	PENTANE	26	C2H2	46	C4H10-04	66	N2-02
7	C5H12-01	27	C4H6--01	47	C2H4O-01	67	CO
8	C5H12-02	28	C5H8--01	48	C4H8O-02	68	CO2
9	HEXANE	29	C6H6	49	CCL2F-01	69	H2S
10	HEPTANE	30	C7H8	50	CHCLF-01	70	CS2
11	OCTANE	31	C8H10-01	51	CCL3F	71	H4N2
12	NONANE	32	C8H10-02	52	C2CL3-01	72	HCL
13	DECANE	33	C8H10-03	53	CH3CL	73	CHN
14	DODECANE	34	C8H10-04	54	CHCL3	74	C3H6O-01
15	C14H3-01	35	C9H12-01	55	CCL4	75	C2H3N
16	C16H3-01	36	C12H1-01	56	C6H5C-01	76	C2H4O-02
17	C5H10-01	37	C10H8	57	AR	77	H3N
18	C6H12-01	38	C11H1-01	58	BR2	78	H2O
19	C6H12-02	39	CH4O	59	CL2		
20	C7H14-01	40	C2H6O-01	60	NO-02		

Tabla 54: componentes presentes en Calculator of Thermodynamic Properties.

8.4. Anexo IV. Manual de uso de la app 'Calculator of Thermodynamic Propierties'

A continuación se mostrarán paso por paso todos los aspectos que ofrece la versión de la aplicación desarrollada, haciendo especial hincapié en las nuevas características implementadas, como son la base de datos, las gráficas, y el funcionamiento de la tarea asíncrona.

En primer lugar, nada más ejecutar la aplicación, aparecerá durante unos segundos el logo de la Escuela de Ingenieros Industriales (Figura 11), que dará paso a la pantalla principal de la aplicación (MainActivity, Figura 12).

Figura 10: Primera pantalla. Logo de la Escuela

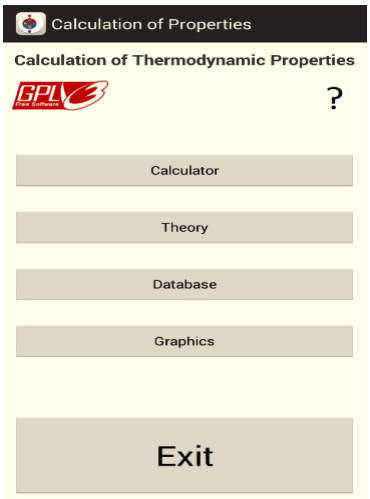


Figura 11: Pantalla Principal

En ella podemos ver 4 botones principales y el botón de salida, que se comentarán en unos instantes.

En la parte superior de la aplicación hay dos imágenes: una interrogación y el logo de GPL, que es la licencia pública del software GNU.

Pulsando sobre la primera imagen, se desplegará una ventana de ayuda, que indicará al usuario qué es lo que encontrará pulsando los diferentes botones (Figura 13). Por otro lado, si se pulsa la segunda imagen, aparecerá la licencia del programa, y un botón que llevará al usuario a una pestaña donde se muestran todas las condiciones de la GNU GPL (Figura 14).

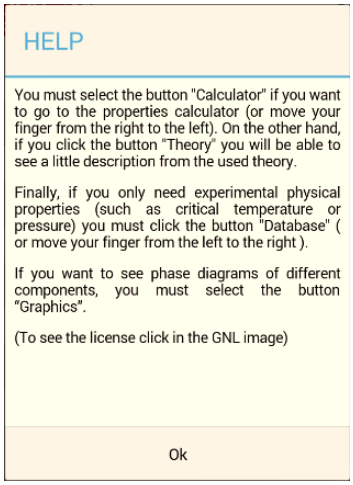
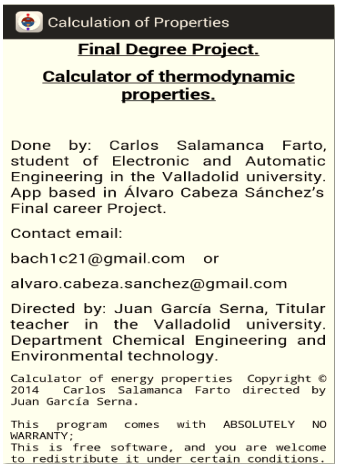


Figura 12: Ayuda de la Pantalla Principal



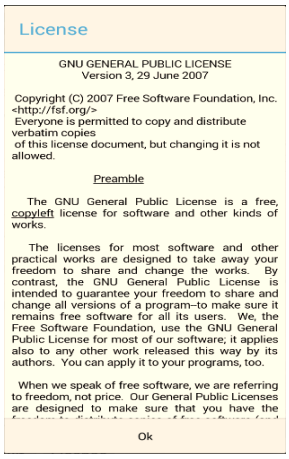


Figura 13: Licencia de la aplicación y GNU GPL

Ahora, es el momento de comenzar a 'jugar' con la aplicación.

8.4.1. Calculadora

Al pulsar sobre el botón 'Calculator', aparecerá una nueva pantalla, en la que se le solicita al usuario que introduzca una presión (en atmósferas), una temperatura (en grados Celsius), y una ecuación para realizar los cálculos (Van der Waals, Peng-Robinson, Soave-Redlich-Kwong, o volúmenes trasladados). También permite la posibilidad de realizar los cálculos con la temperatura y la presión de saturación, y de utilizar los parámetros de iteración binaria. La aplicación no permite elegir de manera simultánea presión y temperatura de saturación, ya que los cálculos no serían posibles. (Figuras 15, 16, 17 y 18).

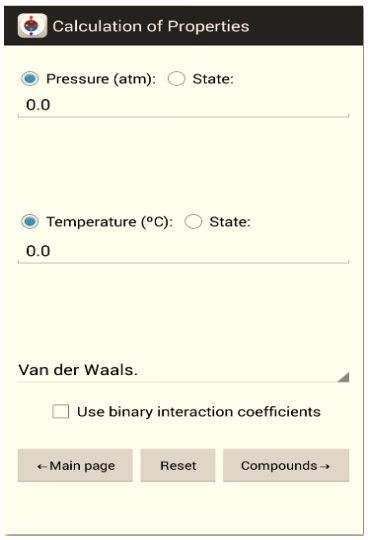


Figura 14: Selección de características (I)

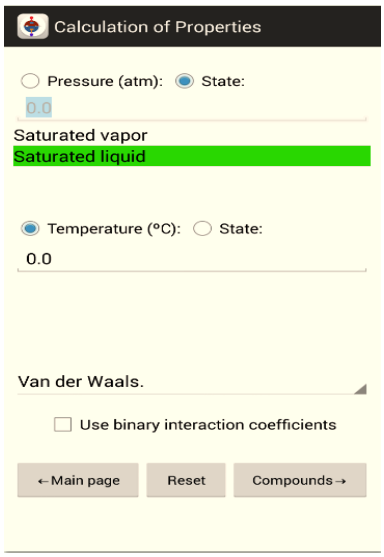


Figura 15: Selección de características (II)

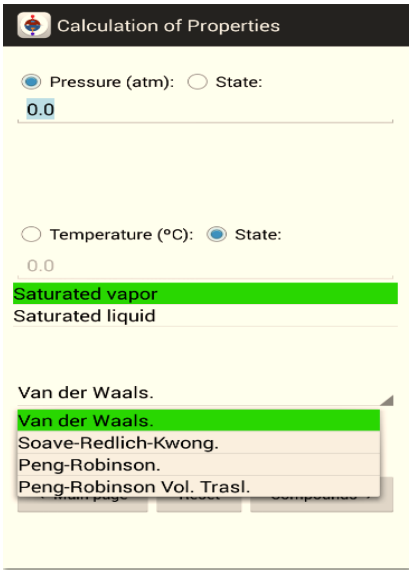


Figura 16: Selección de características (III)

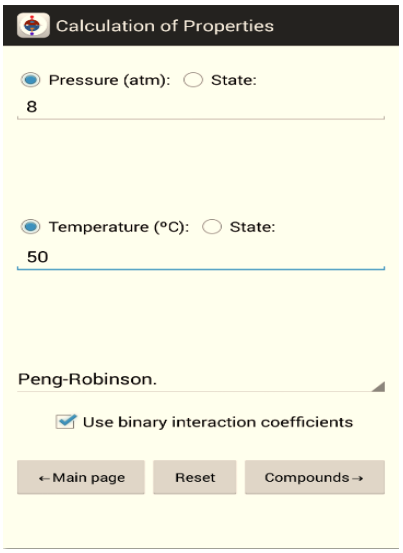


Figura 17: Selección de características (IV)

En esta pantalla hay 3 botones. En botón 'Main Page' devuelve al usuario a la pantalla principal, el botón 'Reset' devuelve los valores de la pantalla a su estado inicial, y el botón 'Compounds' lleva al usuario a una nueva pantalla, donde se seleccionarán los componentes con los que queremos realizar los cálculos (Figura 19).

En dicha pantalla, podemos encontrar de nuevo una interrogación, que nos dice cómo operar adecuadamente (Figura 20). La aplicación permite al usuario seleccionar hasta 6 componentes distintos para realizar las mezclas,

solicitando el nombre y el tanto por 1 de composición de la misma (Figuras 22 y 23). En caso de que la suma total no sea igual a la unidad, la propia aplicación realizará los reajustes pertinentes para poder realizar los cálculos (Figura 24).

El cajetín donde se seleccionan los componentes se autocompleta, mostrando al usuario las posibilidades existentes según va escribiendo el nombre. (Figura 21).

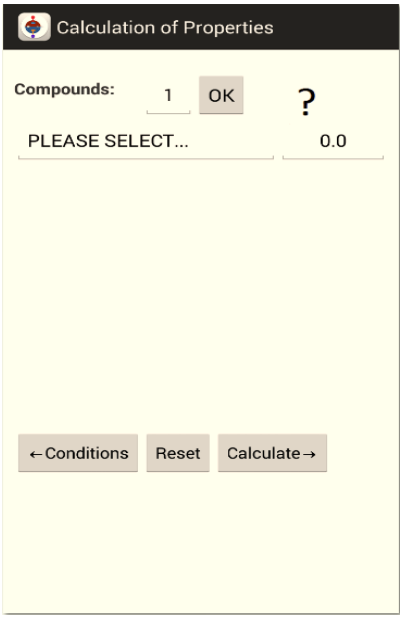


Figura 18: Pantalla de componentes (vacía)

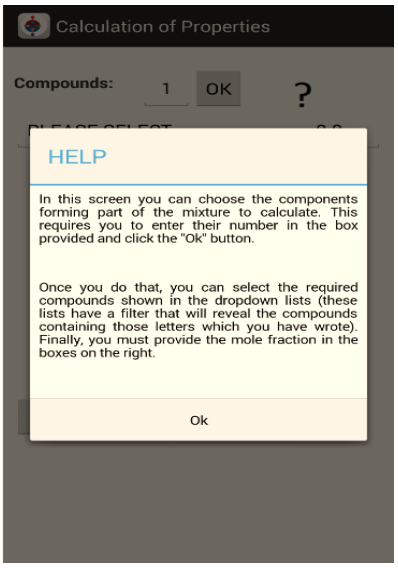


Figura 19: Ayuda de la pantalla de componentes

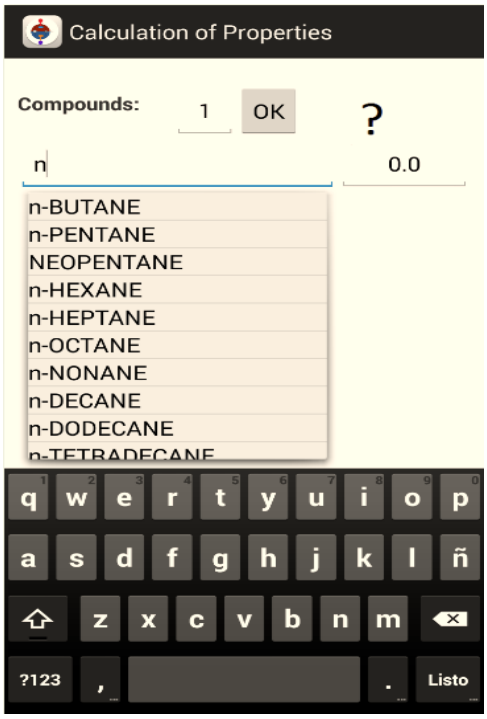


Figura 20: Desplegable de componentes

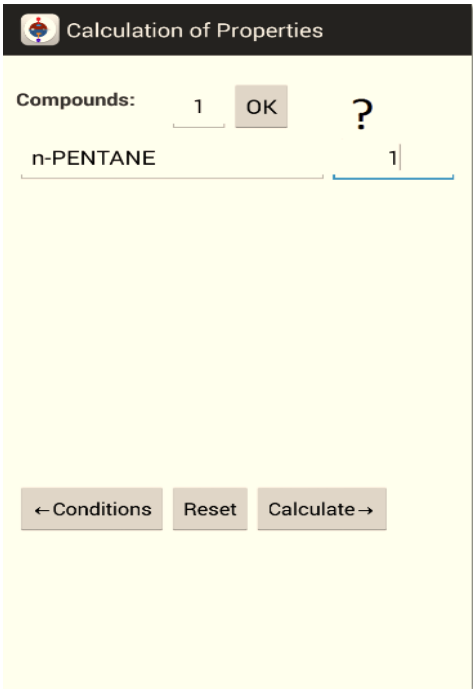


Figura 21: Pantalla de componentes (1 componente)

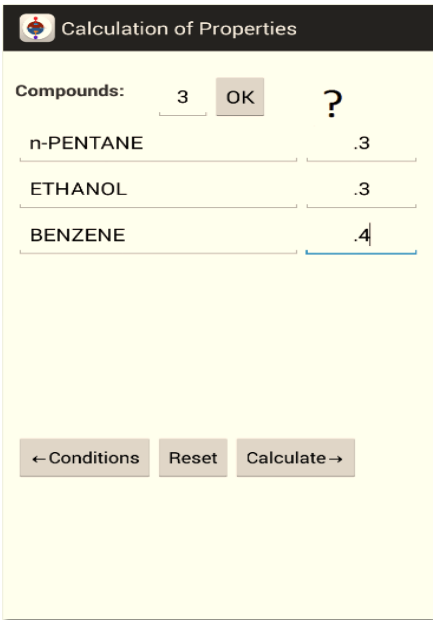


Figura 22: Pantalla de componentes (3 componentes)

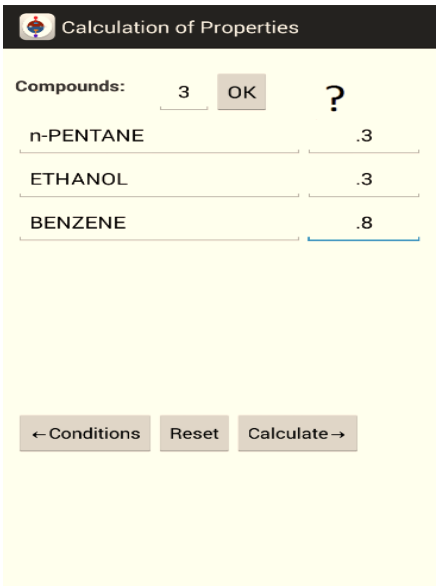


Figura 23: Pantalla de componentes (3 componentes II)

En la parte inferior de la pantalla aparecen de nuevo 3 botones. El botón 'Conditions' lleva al usuario a la pantalla anterior, el botón 'Reset' devuelve los valores iniciales a la pantalla, y el botón 'Calculate' lleva al usuario a la siguiente pantalla, donde se le mostrarán los resultados obtenidos (Figura 25).

En esta pantalla se mostrarán diferentes resultados en función de los parámetros escogidos en las pantallas anteriores. Si no se ha seleccionado ningún componente, o falta algún valor de presión o temperatura por seleccionar, la aplicación señalará que no posee datos suficientes para realizar los cálculos (Figura 26). Si los datos introducidos son correctos, se mostrarán todos los valores obtenidos por medio de los cálculos internos de la aplicación, como la entalpía, la entropía, la energía interna, la exergía o la energía libre de Gibbs (Figuras 25 y 27), y en caso de que la suma de todos los compuestos supere la unidad, mostrará un mensaje en el que indica que la aplicación ha hecho un reajuste de la composición, mostrando también por pantalla las nuevas composiciones (Figura 28).

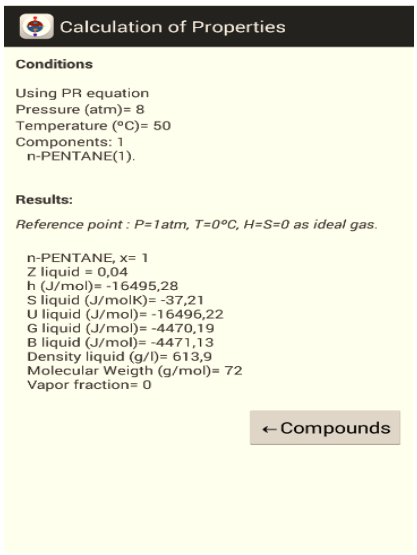


Figura 24: Pantalla Componentes (I)

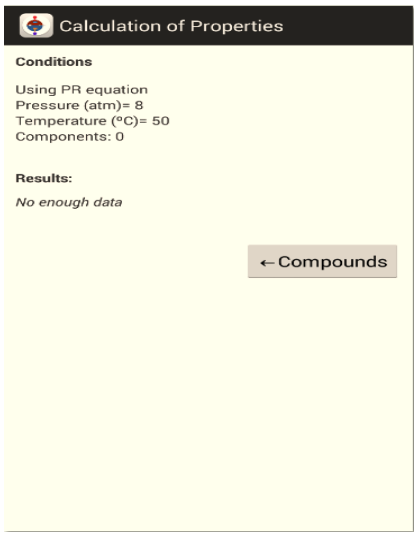


Figura 25: Pantalla Componentes (II)

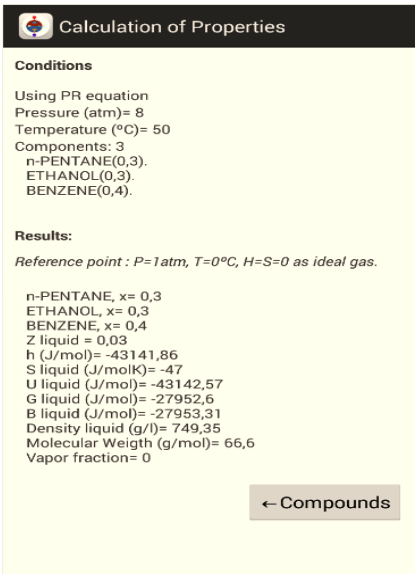


Figura 26: Pantalla Componentes (III)

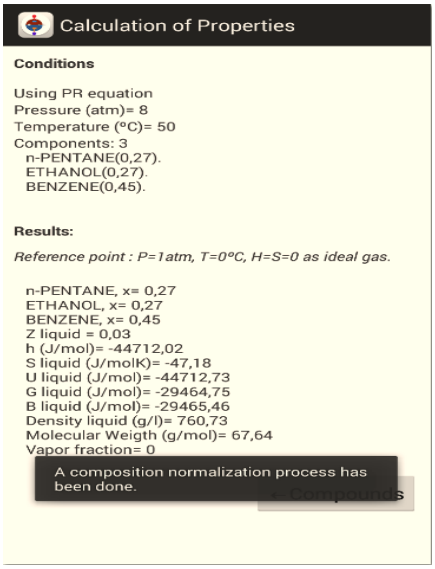


Figura 27: Pantalla Componentes (IV)

Una de las principales ventajas que presenta esta aplicación con respecto a su anterior versión, es que en caso de que el cálculo tarde demasiado en producirse, el usuario puede seguir interactuando con la misma sin que esta quede bloqueada (ver 3.4.2. AsyncTask).

En la parte inferior de la pantalla, aparecerá un botón que permite al usuario volver a la pantalla anterior de 'Componentes', para cambiar los parámetros en caso de querer realizar un nuevo cálculo.

De este modo, quedan definidas todas las posibilidades que ofrece la calculadora de la aplicación.

8.4.2. Teoría

Al pulsar sobre el botón 'Theory' de la pantalla principal, la aplicación mostrará una nueva pantalla con tres posibles pestañas, en las que está explicada toda la introducción teórica necesaria para conocer los entresijos de los cálculos que la misma realiza. Dependiendo de la pestaña seleccionada, se puede leer la teoría sobre las ecuaciones de estado (Pestaña 'EOS', Figura 29), sobre el equilibrio líquido-vapor (Pestaña 'LVE', Figura 30), y sobre el cálculo de propiedades energéticas (Pestaña EPC, Figura 31).

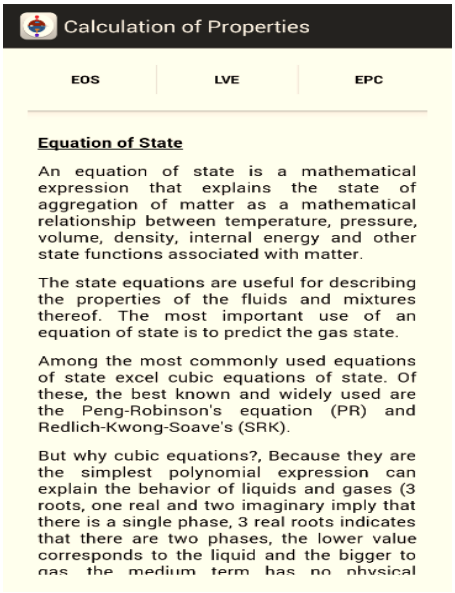


Figura 28: Teoría de las Ecuaciones de Estado

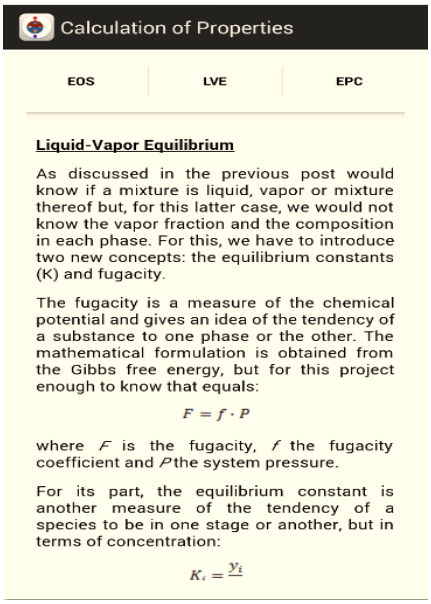


Figura 29: Teoría del Equilibrio Líquido-Vapor

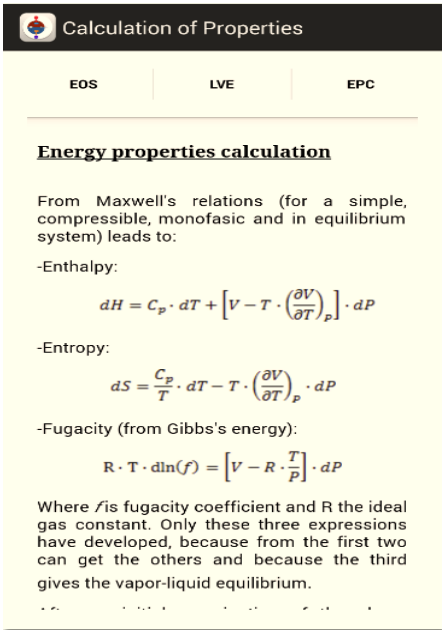


Figura 30: Teoría del Cálculo de Propiedades Energéticas

8.4.3. Base de Datos

En la pantalla a la que se puede acceder al pulsar sobre el botón 'Database', se encuentran los 78 componentes básicos establecidos por el programador (Figura 32).

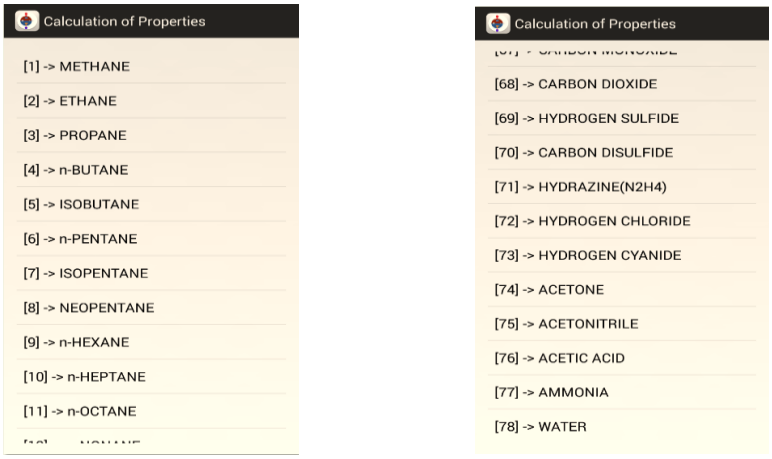


Figura 31: Lista de componentes

Al pulsar sobre uno de ellos, como por ejemplo, el agua, surgirá una nueva pantalla en la que aparecerán las principales características del componente, como su temperatura y presión crítica, el factor acéntrico, o el factor de compresibilidad crítica (Figura 33). Además, en la parte inferior se encuentran 4 botones, que permiten al usuario interactuar con la base de datos como él desee.

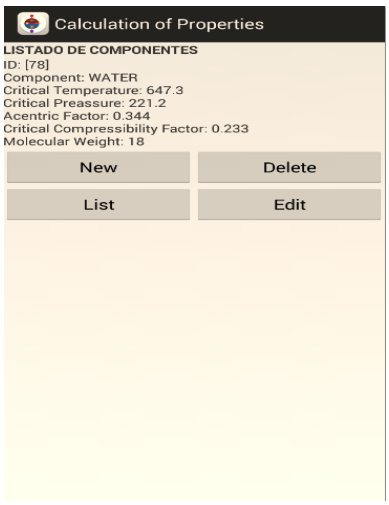


Figura 32: Características del Agua

El botón ‘List’ devuelve al usuario a la lista de componentes de la pantalla anterior (Figura 32). El botón ‘New’ lleva al usuario a una nueva pantalla, donde podrá añadir tantos componentes como desee (Figura 34). Para ello, solicitará todos los datos necesarios para poder realizar cálculos con él posteriormente.

Crearemos un componente ficticio llamado ‘TFG’, al que le daremos valores falsos aleatorios (Figura 35). En esta pantalla hay 3 botones. Pulsando sobre ‘Cancel’, se regresa a la pantalla anterior, que en nuestro caso son las características del agua (Figura 33).

Con el botón ‘Components’, se regresa a la lista inicial de componentes (Figura 32), y pulsando sobre ‘Save’, el nuevo componente queda guardado en la lista (Figura 36).

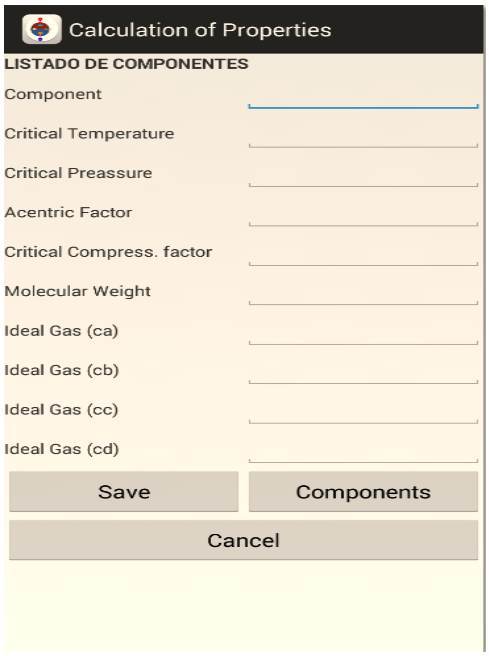


Figura 33: Nuevo componente I

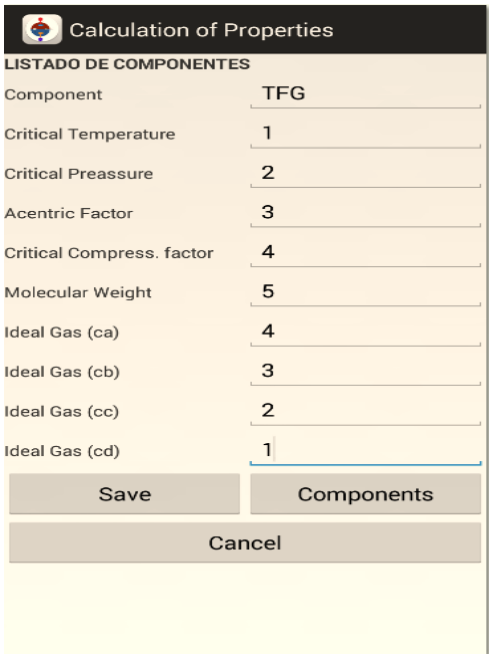


Figura 34: Nuevo componente II

Figura 35: Características nuevo componente

Ahora la aplicación ha llevado al usuario a la pantalla que muestra las características del nuevo componente creado. Al pulsar sobre el botón 'List', se puede ver cómo el nuevo componente aparece el último en la Base de Datos, y está listo para trabajar con él (Figura 37).

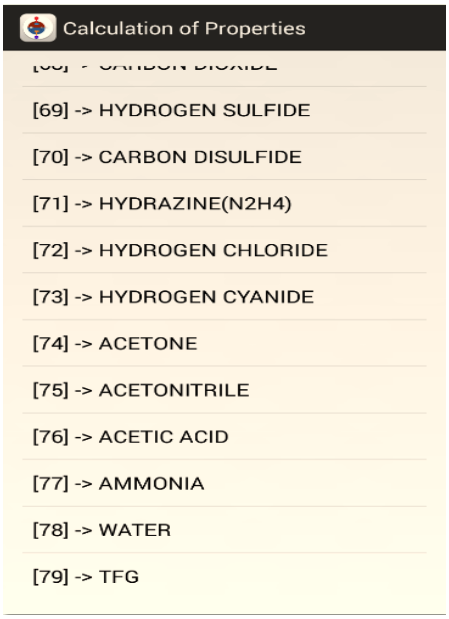


Figura 36: Nueva lista de componentes

Volviendo a la pantalla de características del nuevo componente, quedan dos botones por comprobar. El botón 'Edit' permite al usuario cambiar cualquiera de los parámetros de cualquiera de los componentes. Al editar el nuevo componente introducido, volverá a solicitar todos los datos necesarios para realizar cálculos, y al guardar dichos cambios, muestra las nuevas características por pantalla (Figura 37). Al volver a la lista, se puede ver cómo el anterior compuesto ha desaparecido y ya sólo aparece el editado (Figura 38).

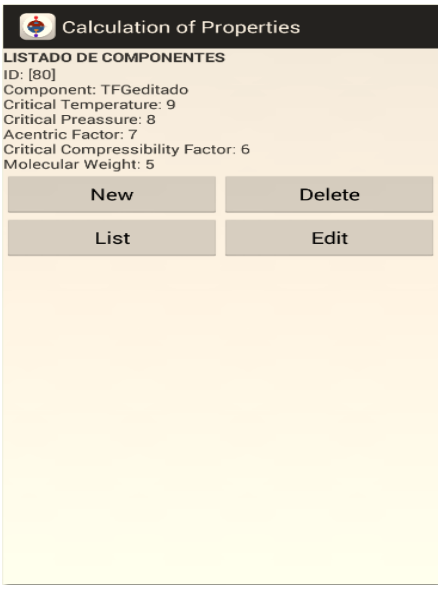


Figura 37: Características editadas

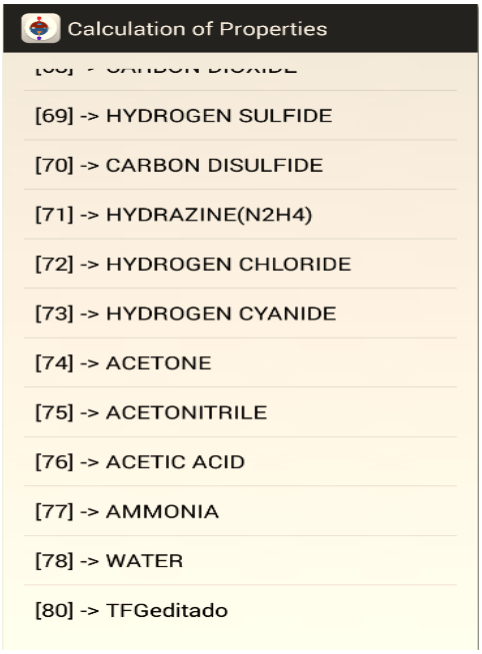


Figura 38: Nueva Lista de componentes II

Por último, si se pulsa el botón 'Delete', la aplicación le pregunta al usuario si realmente desea eliminarlo (Figura 39). En caso afirmativo, la aplicación volverá a la lista de componentes, en la que ya no aparecerá el componente creado anteriormente. También se borrará en agua y el amoniaco para mostrar el funcionamiento de la misma (Figura 40).

Figura 39: Mensaje de aviso

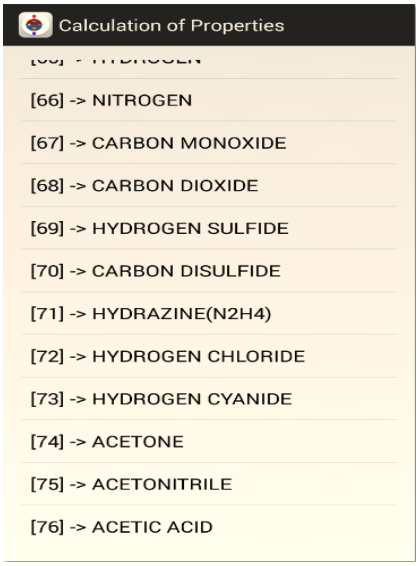


Figura 40: Lista de componentes con eliminaciones

8.4.4. Gráficas

Al pulsar sobre el botón 'Graphics', se mostrará una pantalla en la que el usuario podrá elegir la gráfica que desea hacer y la ecuación de la que quiere partir (Figura 41). Como ya se ha comentado a lo largo del informe, en caso de seleccionar una gráfica binaria, la aplicación mostrará un mensaje de aviso diciendo que esta posibilidad aún no está implementada (Figura 42). Una vez seleccionada la ecuación que se quiere utilizar, el usuario podrá volver a la página principal (Botón 'Main Page') o proceder a dibujar la gráfica (Botón 'Draw').

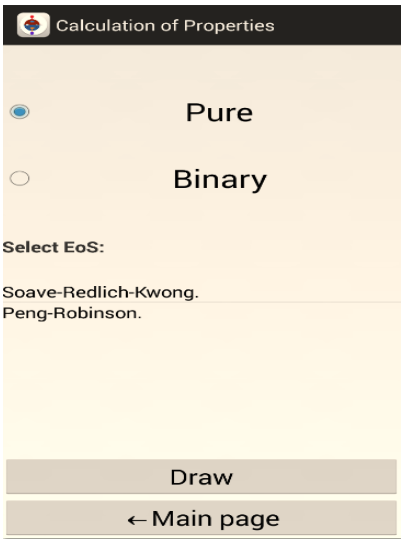


Figura 41: Grafica Pura

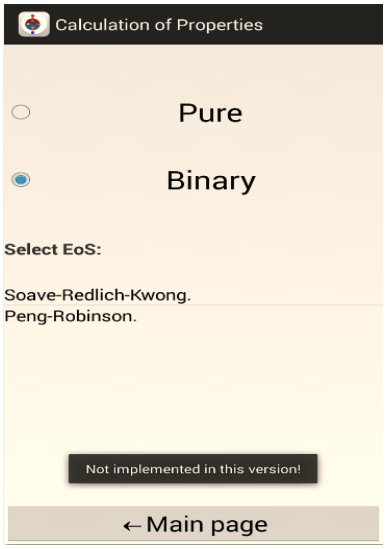


Figura 42: Gráfica Binaria

Al pulsar sobre 'Draw', la aplicación solicitará la elección de un componente para graficar, y de un tipo de diagrama a representar, a saber: Presiones saturadas frente a temperatura o temperaturas saturadas frente a presión.

En el primer caso, habrá que introducir una temperatura inicial y final para la representación (Figura 43), y en el segundo, una presión inicial y final (Figura 44).

La selección del componente, al igual que en la calculadora, se autocompleta en función del texto introducido.

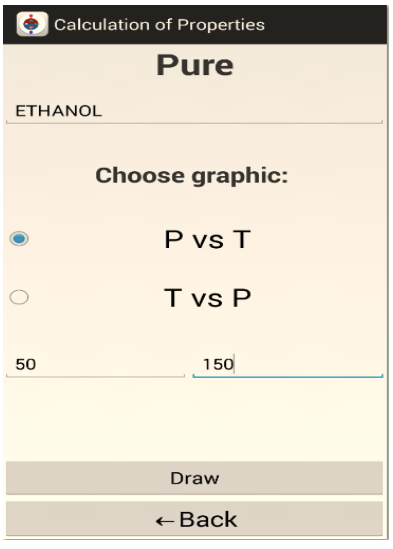


Figura 43: Selección P vs T

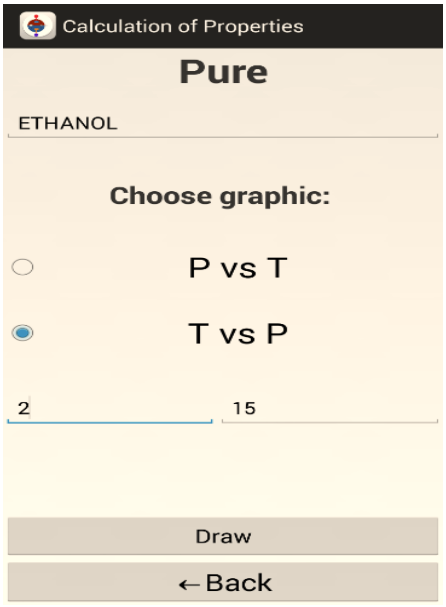


Figura 44: Selección T vs P

Una vez elegida la gráfica, al pulsar sobre 'Draw' se mostrará la gráfica formada por 6 puntos equidistantes en el eje de abscisas (Figuras 45 y 46).

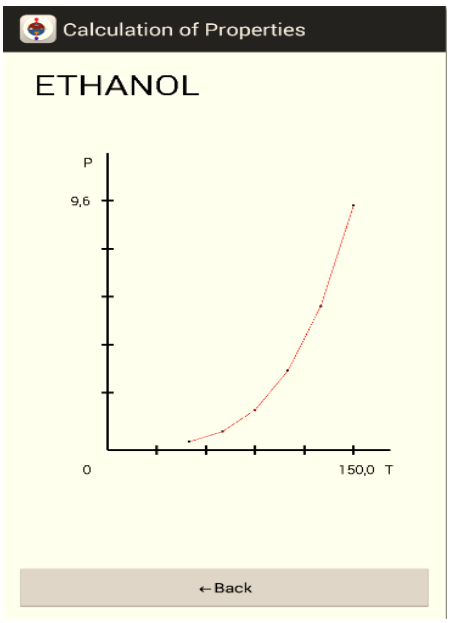


Figura 45: Grafica P vs T del Etanol

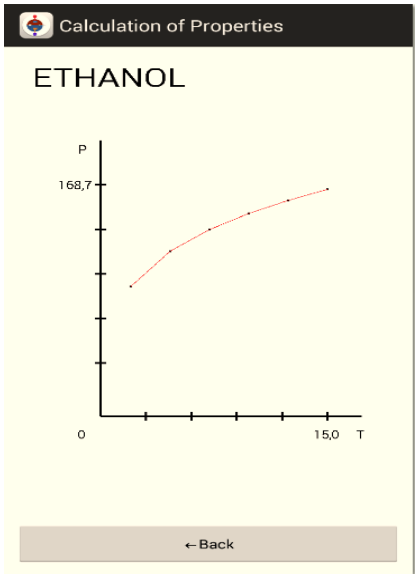


Figura 46: Gráfica T vs P del Etanol

Al pulsar sobre el botón 'Back', se puede retroceder hasta las pantallas anteriores hasta llegar a la pantalla inicial.

8.4.5. Exit

Por último, una vez visitados todos los aspectos de la aplicación, pulsaremos el botón 'Exit' para salir, y la aplicación mostrará un mensaje de despedida antes de regresar a la pantalla de inicio del dispositivo empleado (Figura 47).

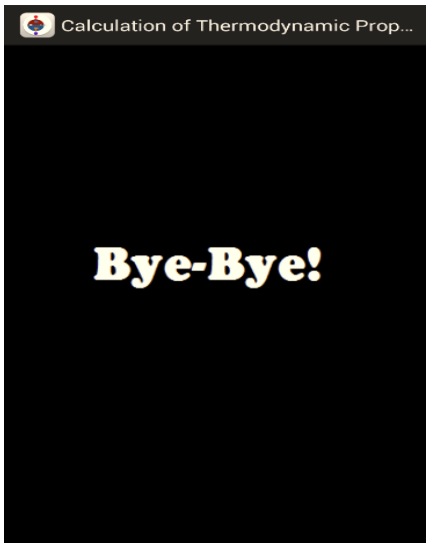


Figura 47: Mensaje de despedida

8.5. Anexo V. Licencia

Todos los programas desarrollados en este proyecto están protegidos por una licencia libre GNU asegurando su disponibilidad de forma completamente libre, pudiendo ser copiado o redistribuido según las condiciones fijadas por la licencia:

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Para más información consultar la siguiente página Web:

http://www.gnu.org/licenses.

8.6. Anexo VI. Explicación del código

Para facilitar la compresión del código utilizado, a pesar de que está debidamente comentado, se incluye este anexo en el que se explica paso a paso la intención de las nuevas funciones implementadas y la causa de su estructura.

InicialLabel.java

```
public class InicialLabel extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.iniciallabel);

        //Vamos a declarar un nuevo thread
        Thread timer = new Thread(){

            //El nuevo Thread exige el metodo run
            public void run(){

                //Mantiene la pantalla activa durante 2.5 segundos
                try{

                    sleep(2500);
                }catch(InterruptedException e){
                    //Si no puedo ejecutar el sleep muestro el error
                    e.printStackTrace();
                }finally{

                    //Tras los 2.5 segundos, se ejecuta una nueva actividad (MainActivity)
                    Intent Starter = new Intent(InicialLabel.this, MainActivity.class);
                    finish();
                    startActivity(Starter);

                }
            }
        };

        timer.start();
    }
}
```

La clase InicialLabel.java ejecuta un 'splash' inicial con el logo de la Escuela. Este logo se mantendrá en pantalla durante el tiempo estipulado, y después dejará paso a la que consideraremos la pantalla principal, MainActivity.java.

En caso de que el bloqueo de la pantalla no pudiera ejecutarse en el dispositivo, el programa mostrará un mensaje de error alertando de ello.

Closer.java

```
public class closer extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.closer);

        //Vamos a declarar un nuevo thread
        Thread leaving = new Thread(){
            //El nuevo Thread exige el metodo run
            public void run(){
                try{
                    //Mantiene la pantalla activa durante 1 segundo
                    sleep(1000);
                }catch(InterruptedException e){
                    //Si no puedo ejecutar el sleep muestro el error
                    e.printStackTrace();
                }finally{
                    //Transcurrido el segundo de espera, cierra todos los procesos operativos
                    finish();
                    System.exit(0);
                }
            }
        };
        //ejecuto el thread
        leaving.start();
    }

}
```

De igual modo que en la clase InicialLabel.java, la clase Closer.java esperará un tiempo determinado mientras muestra el mensaje de despedida, para luego cerrar todos los procesos activos y regresar a la pantalla inicial del dispositivo empleado.

MainActivity.java

Esta clase se mantiene igual que en la anterior versión, con la salvedad de que se han modificado algunos aspectos en la interfaz de la misma, y añadido el botón de 'Graphics'.

```
button05.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Intent draw=new Intent(MainActivity.this, graphic.class);
        draw.putExtras(bundle);
        startActivity(draw);
        overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);

    }
});
```

Datos.java

Se mantiene tal y como aparecía en la versión anterior, con el añadido de la nueva ecuación para los volúmenes trasladados.

Componentes.java

Ídem a Datos.java

Resultados.java

Se ha realizado una reestructuración del código para que funcione por medio de tareas asíncronas. Esto hace que la ejecución de los cálculos sea independiente del resto de la aplicación, y por tanto evita el bloque de la misma.

Para ello, la estructura seguida en el código es la siguiente:

```
//Antes de comenzar a realizar operaciones
protected void onPreExecute() {
    /* Se obtiene la información necesaria */
                                        ...
                                        ...
                                        ...
}
```

Antes de comenzar a realizar operaciones, se definen todos los parámetros necesarios para la correcta implementación de éstas. Una vez definidas las variables, se procede a calcular.

```
//Operaciones llevadas a cabo por 'debajo' de la aplicación
protected Boolean doInBackground(Void... params) {
    /* Mathematical operations */

                                        ...
                                        ...
                                        ...
    return true;
}
```

En el proceso 'doInBackground' se realizarán todas las operaciones que se le hayan solicitado al programa. Al estar definido como un booleano, cuando termine de ejecutarse devolverá el valor 'true', que permitirá la ejecución del siguiente proceso.

```
@Override
protected void onPostExecute(Boolean result) {
    if (result) {
                                    ...
                                    ...
                                    ...
    }
}
```

En último lugar, se procede a mostrar por pantalla todos los cálculos obtenidos. Para ello, el método 'OnPostExecute' espera a recibir el valor 'true' de 'doInBackground' y, cuando lo obtiene, imprime en la pantalla de la aplicación los resultados obtenidos.

EOS.java

En esta clase simplemente se han implementado las nuevas ecuaciones necesarias para ejecutar la ecuación de los volúmenes trasladados, ya definidas en la introducción, con el siguiente resultado:

```
//Vol trasl

double t0(double Tc, double Pc, double w){

    double to;
    if (w != 0){

        to = Rg*Tc*(-0.014471 + 0.067498*w - 0.084852*w*w
                +0.067298*w*w*w-0.017366*w*w*w*w)/Pc;
    }
    else{

        to=0;
    }

    return to;
}


double zc(double w){

    double z = 0;
    if (w != 0){

        z = 0.289 - 0.0701*w - 0.0207*w*w;
    }
    else{

        z = 0;
    }

    return z;
}
```

```
double tc(double Tc, double Pc, double w){

    return (Rg*Tc*(0.3074-zc(w))/Pc);
}

double beta(double w){

    double b;
    if(w != 0){

        b = -10.2427-28.6312*w;
    }
    else{

        b=0;
    }
    return b;
}

double ti(double Tc, double Pc, double w){

    if(Pc==0){Pc=0.000001;}

    return (t0(Tc,Pc,w) + (tc(Tc,Pc,w)-t0(Tc,Pc,w))
            *Math.exp(beta(w)*Math.abs(1-Tri(Tc))));

}

 double t(){

     double tm=0;
     if (mr_type==0){
         for (int i=0; i<=F.length-1;i++){

             tm=tm+F[i]*ti(Tcd[i],Pcd[i],wd[i]);
         }
     }
     return tm;


 }

double  C(){
    return t()*P/(Rg*T);
}
```

Empleando todos los parámetros anteriores para calcular el parámetro Z del componente o de la mezcla, se obtienen los resultados deseados.

La ecuación resuelta es la forma polinómica en función del coeficiente de compresibilidad de las EOS introducidas en el apartado 1.3.

SQLiteHelper.java

La clase SQLiteHelper establece las funciones necesarias para la creación de la base de datos inicial. Simplemente se definirán el nombre y la versión de la tabla, los campos a rellenar y el tipo de valor que solicitará cada campo (numérico, texto, etc). Además, se emplearán las funciones 'OnCreate' y 'OnUpgrade', encargadas de generar la base de datos si es el primer acceso, o de actualizarla si fuera necesario, respectivamente.

```
public class SQLiteHelper extends SQLiteOpenHelper{
    private static final String __DATABASE = "compprueba";
    //versión de la base de datos
    private static final int __VERSION = 1;
    //nombre tabla y campos de tabla
    public final String __comp = "Componente";
    public final String __campo_id = "_id";
    public final String __campo_nombre = "Nombre";
    public final String __campo_tempCri = "CT";
    public final String __campo_presCri = "CP";
    public final String __campo_facAc = "AF";
    public final String __campo_facCom = "ZC";
    public final String __campo_pesoMol = "MW";
    public final String __campo_ca = "ca";
    public final String __campo_cb = "cb";
    public final String __campo_cc = "cc";
    public final String __campo_cd = "cd";


    private final String sql = " CREATE TABLE " + __comp + " ( " +
            __campo_id + " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, " +
            __campo_nombre + " TEXT, " + __campo_tempCri + " NUMERIC, "
            + __campo_presCri + " NUMERIC, " + __campo_facAc + " NUMERIC, " +
            __campo_facCom + " NUMERIC, " + __campo_pesoMol + " NUMERIC, " +
            __campo_ca + " NUMERIC, " +__campo_cb + " NUMERIC, " + __campo_cc +
            " NUMERIC, " + __campo_cd + " NUMERIC " + " ) ";

    public SQLiteHelper(Context context) {
        super( context, __DATABASE, null, __VERSION );

    }

    //Creación de la tabla
    @Override
    public void onCreate(SQLiteDatabase db){

         db.execSQL( sql );
    }

    //Actualización de la tabla
    public void onUpgrade( SQLiteDatabase db,  int oldVersion, int newVersion ) {
        if ( newVersion > oldVersion )
        {
            //elimina tabla
            db.execSQL( " DROP TABLE IF EXISTS " + __comp );
            //y luego creamos la nueva tabla
            db.execSQL( sql );
        }
    }


}
```

SQLite.java

Una vez definidos los parámetros esenciales de la base de datos en la clase 'SQLiteHelper', se crean las funciones que permitirán a la aplicación acceder a los diferentes campos de la misma.

En primer lugar, se ha generado una base de datos inicial a través del programa 'Valentina Studio', que posteriormente podrá ser modificada por el usuario de la aplicación. El aspecto que presenta dicha base de datos es el siguiente:

_id	Nombre	CT	CP	AF	ZC	MW	ca	cb	cc	cd
1	METHANE	190.6	46.04	0.011	0.288	16	19.25	0.05	1.2e-05	-1.13e-08
2	ETHANE	305.4	48.8	0.099	0.284	30	5.4	0.178	-6.937e-05	8.71e-09
3	PROPANE	369.8	42.49	0.152	0.281	44	-4.224	0.3062	-0.0001586	3.21e-08
4	n-BUTANE	425.2	37.97	0.193	0.274	58	9.487	0.331	-0.00011	-2.82e-09
5	ISOBUTANE	408.1	36.48	0.177	0.282	58	-1.39	0.3847	-0.000184	-2.89e-09
6	n-PENTANE	469.7	33.69	0.249	0.269	72	-3.623	0.4873	-0.000258	5.3e-08
7	ISOPENTANE	460.4	33.81	0.228	0.27	72	-9.525	0.5066	-0.000273	5.72e-08
8	NEOPENTANE	433.8	31.99	0.196	0.269	72	-16.592	0.555	-0.00033	7.63e-08
9	n-HEXANE	507.4	30.12	0.305	0.264	86	-4.41	0.582	-0.000311	6.49e-08
10	n-HEPTANE	540.3	27.36	0.349	0.263	100	-5.146	0.676	-0.000365	7.66e-08
11	n-OCTANE	568.8	24.86	0.396	0.259	114	-6.096	0.771	-0.00042	8.85e-08
12	n-NONANE	595.7	23.06	0.437	0.255	128	3.144	0.677	-0.000193	-2.98e-08
13	n-DECANE	618.5	21.23	0.484	0.249	142	-7.913	0.961	-0.000529	1.3e-07
14	n-DODECANE	658.2	18.24	0.575	0.238	170	-58.979	1,005	-0.0006594	1.6e-07
15	n-TETRADECANE	696.9	14.38	0.57	0.203	198	-10.98	1.337	-0.000742	1.6e-07
16	n-HEXADECANE	720.6	14.19	0.747	0.22	226	-13.017	1.53	-0.000854	1.85e-07
17	CYCLOPENTANE	511.8	45.02	0.194	0.273	70	-53.63	0.543	-0.000303	6.48e-08
18	METHYLCYCLOPENTANE	532.8	37.85	0.23	0.272	84	-50.108	0.638	-0.000364	8.013e-08
19	CYCLOHEXANE	553.5	40.75	0.215	0.273	84	-54.541	0.611	-0.000252	1.32e-08
20	METHYLCYCLOHEXANE	572.2	34.71	0.235	0.269	98	-61.92	0.784	-0.000444	9.36e-08
				...						
				...						
				...						
65	HYDROGEN	33.3	12.97	-0.215	0.3	2	27.14	0.00927	-1.38e-05	7.64e-09
66	NITROGEN	126.1	33.94	0.04	0.292	28	31.15	-0.0135	2.7e-05	-1.16e-08
67	CARBON MONOXIDE	132.9	34.99	0.066	0.295	28	30.87	-0.0128	2.8e-05	-1.27e-08
68	CARBON DIOXIDE	304.2	73.82	0.228	0.274	44	19.79	0.0734	-5.6e-05	1.71e-08
69	HYDROGEN SULFIDE	373.5	89.37	0.081	0.283	34	31.94	0.00144	2.43e-05	-1.176e-08
70	CARBON DISULFIDE	552	78	0.115	0.293	76	27.444	0.08126	-7.666e-05	2.67e-08
71	HYDRAZINE(N2H4)	653	145	0.328	0.26	32	9.77	0.189	-0.000166	6.024e-08
72	HYDROGEN CHLORIDE	324.6	82	0.12	0.249	36	30.29	-0.0072	1.24e-05	-3.9e-09
73	HYDROGEN CYANIDE	456.8	53.2	0.407	0.197	27	21.863	0.06062	-4.96e-05	1.81e-08
74	ACETONE	508.2	47.01	0.306	0.233	58	6.301	0.26	-0.000125	2.037e-08
75	ACETONITRILE	545.5	48.33	0.353	0.184	41	20.482	0.108	-4.49e-05	3.2e-09
76	ACETIC ACID	592.7	57.86	0.462	0.2	60	4.84	0.254	-0.000175	4.98e-08
77	AMMONIA	406.6	112.7	0.252	0.242	17	28.73	0.0179	2.39e-05	1.42e-08
78	WATER	647.3	221.2	0.344	0.233	18	32.243	0.00192	1.055e-05	-3.59e-09

Figura 48: Base de Datos en Valentina Studio

Una vez creada la base de datos, se almacenará en una carpeta interna de la aplicación, para que cada vez que ésta se instale, se genere la lista de componentes en el propio dispositivo.

La clase 'SQLite' se encarga de tomar dicho archivo y convertirlo en la "*Database*" de nuestra aplicación.

Una vez hecho esto, en la clase se definirán las funciones de apertura y clausura de la base de datos, así como las que se encargan de la extracción de datos.

```
public class SQLite {

    private SQLiteHelper sqliteHelper;
    private SQLiteDatabase db;
    private static final String __DATABASE = "compprueba";
    public final String __campo_id = "_id";
    private final Context myContext;
    private static final String __PATH = "/data/data/com.pfc.ecuaciones/databases/";



    /** Constructor de clase */
    public SQLite(Context context)
    {
        sqliteHelper = new SQLiteHelper( context );
        this.myContext = context;

    }


    * @throws IOException */
   public void abrir() throws IOException{
       boolean flag = false;
       SQLiteDatabase checkDB = null;
       String myPath = __PATH + __DATABASE;
       try{
       checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READWRITE);

       }catch(SQLiteException e){
           //Aún no existe la base de datos
       }
       if(checkDB != null){

           checkDB.close();
           flag = true;
       }

       //Obtiene el nombre de la base de datos y la abre
       if(flag==true){
           db = sqliteHelper.getWritableDatabase();
       }
       else{
               db = sqliteHelper.getWritableDatabase();
               InputStream myInput = myContext.getAssets().open(__DATABASE);
               String outFileName = __PATH +__DATABASE; // Path to the just created empty db
               OutputStream myOutput = new FileOutputStream(outFileName); //Open the empty db
               //transfer bytes from the inputfile to the outputfile
               byte[] buffer = new byte[1024];
               int length;
               while ((length = myInput.read(buffer))>0){
                   myOutput.write(buffer);
               }
               //Close the streams
               myOutput.close();
               myOutput.flush();
               myInput.close();
       }
   }
```

```
    /** Cierra conexion a la base de datos */
    public void cerrar()
    {
        sqliteHelper.close();
    }


    //genera un hueco en la base de datos para añadir un registro
    public boolean addRegistro( String nombre, String CT, String CP, String AF,
            String ZC, String MW, String ca, String cb, String cc, String cd )
    {
        if( nombre.length()> 0 )
        {
            ContentValues contentValues = new ContentValues();
            contentValues.put( sqliteHelper.__campo_nombre , nombre);
            contentValues.put( sqliteHelper.__campo_tempCri , CT  );
            contentValues.put( sqliteHelper.__campo_presCri , CP);
            contentValues.put( sqliteHelper.__campo_facAc , AF);
            contentValues.put( sqliteHelper.__campo_facCom , ZC);
            contentValues.put( sqliteHelper.__campo_pesoMol , MW);
            contentValues.put( sqliteHelper.__campo_ca , ca);
            contentValues.put( sqliteHelper.__campo_cb , cb);
            contentValues.put( sqliteHelper.__campo_cc , cc);
            contentValues.put( sqliteHelper.__campo_cd , cd);
            Log.i("SQLite", "Nuevo registro " );
            return ( db.insert( sqliteHelper.__comp , null, contentValues )
                    != -1 )?true:false;
        }
        else
            return false;
    }

    //Extrae el ID del último componente de la BdD
    public int getUltimoID()
    {
        int id = -1;

        Cursor cursor = db.query( sqliteHelper.__comp ,
                new String[]{ sqliteHelper.__campo_id },
                null, null, null,null,
                sqliteHelper.__campo_id + " DESC ", "1");
        if( cursor.moveToFirst() )
        {
            do
            {
                id = cursor.getInt(0);
            } while ( cursor.moveToNext() );
        }
        return id;
    }

    //Elimina el registro seleccionado
    public boolean borrar_registro( int id )
    {
        //table , whereClause, whereArgs
        return  (db.delete( sqliteHelper.__comp , sqliteHelper.__campo_id + " = " + id ,
                null) > 0) ? true:false;

    }

//Obtiene el nombre del registro seleccionado
public Cursor getRegistrosComponentes()
{
    return db.query( sqliteHelper.__comp ,  new String[]{sqliteHelper.__campo_nombre},
                null, null,
            null, null, null);
}
```

```
//Obtiene el ID y el nombre del registro seleccionado
public Cursor getRegistros()
{
    return db.query( sqliteHelper.__comp ,  new String[]{sqliteHelper.__campo_id,
            sqliteHelper.__campo_nombre},
            null, null,
            null, null, sqliteHelper.__campo_id);
}

//Obtención de parámetros del registro seleccionado
public Cursor getID(String nombre){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_id},
            sqliteHelper.__campo_nombre + "=?",new String[]{nombre}, null, null, null);
}



public Cursor getTC(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_tempCri},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getPC(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_presCri},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getZC(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_facCom},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}


public Cursor getw(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_facAc},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getMW(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_pesoMol},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getca(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_ca},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getcb(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_cb},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getcc(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_cc},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}

public Cursor getcd(int id){
    return db.query(sqliteHelper.__comp, new String[]{sqliteHelper.__campo_cd},
            sqliteHelper.__campo_id + " = " + id,
            null, null, null, null);
}
```

```
//Obtiene todos los datos del registro seleccionado
public Cursor getRegistro( int id )
{
    return db.query( sqliteHelper.__comp ,
                new String[]{
                        sqliteHelper.__campo_id ,
                        sqliteHelper.__campo_nombre,
                        sqliteHelper.__campo_tempCri,
                        sqliteHelper.__campo_presCri,
                        sqliteHelper.__campo_facAc,
                        sqliteHelper.__campo_facCom,
                        sqliteHelper.__campo_pesoMol
                              },
            sqliteHelper.__campo_id + " = " + id ,
            null, null, null, null);
}

//Convierte el ID del registro seleccionado en una cadena de caracteres
public ArrayList<String> getFormatList0( Cursor cursor )
{
    ArrayList<String> listData0 = new ArrayList<String>();
    String item = "";
        if( cursor.moveToFirst() )
        {
            do
            {
                item += cursor.getString(0);
                listData0.add( item );
                item="";

            } while ( cursor.moveToNext() );
        }
    return listData0;
}



//Convierte el ID del registro seleccionado en una variable entera
//y el nombre del mismo en una cadena de caracteres
public ArrayList<String> getFormatList( Cursor cursor )
{
    ArrayList<String> listData = new ArrayList<String>();
    String item = "";
        if( cursor.moveToFirst() )
        {
            do
            {
                item += "[" + cursor.getInt(0) + "] -> ";
                item += cursor.getString(1) + "";
                listData.add( item );
                item="";

            } while ( cursor.moveToNext() );
        }
    return listData;
}
```

```
//Obtiene los datos fundamentales del componente para mostrarlos por pantalla
public ArrayList<String> getFormatList2( Cursor cursor )
{
    ArrayList<String> listData2 = new ArrayList<String>();
    String item = "";
        if( cursor.moveToFirst() )
        {
            do
            {
                item += "ID: [" + cursor.getInt(0) + "]\r\n";
                item += "Component: " + cursor.getString(1) + "\r\n";
                item += "Critical Temperature: " + cursor.getString(2) + "\r\n";
                item += "Critical Preassure: " + cursor.getString(3) + "\r\n";
                item += "Acentric Factor: " + cursor.getString(4) + "\r\n";
                item += "Critical Compressibility Factor: " + cursor.getString(5) + "\r\n";
                item += "Molecular Weight: " + cursor.getString(6) + "";
                listData2.add( item );
                item="";

            } while ( cursor.moveToNext() );
        }
    return listData2;
}

//Convierte el ID del registro seleccionado en una variable entera
public int getFormatList3( Cursor cursor )
{
    int item=0;
    if( cursor.moveToFirst() ){
        do{
            item = cursor.getInt(0);
        }while ( cursor.moveToNext() );
    }
    return item;

}

  //Convierte el ID del registro seleccionado en una variable decimal
  public double getFormatList4( Cursor cursor )
  {
      double item=0;
      if( cursor.moveToFirst() ){
          do{
              item = cursor.getDouble(0);
          }while ( cursor.moveToNext() );
      }
      return item;
  }
}
```

Listcomp.java

La clase 'Listcomp' genera el listado de componente tal y como se muestra en la aplicación. Para lograrlo, abre la base de datos SQL, obtiene los registros, y los muestra por pantalla. Además, en caso de pulsar sobre alguno de ellos, accede a las características del mismo.

Para conseguir esto, el programa extrae el valor del ID del componente seleccionado, accede a la base de datos, y busca el componente que posea un ID del mismo valor que el seleccionado. Cuando lo ha encontrado, se

extiende una nueva clase, en la que se muestran impresos los datos que caracterizan al componente elegido.

```
public class listcomp extends Activity implements OnItemClickListener{

    private ArrayAdapter<String> adaptador ;
    private SQLite sqlite;
    private ListView listView;
    public static final int C_EDITAR = 553 ;

    @Override
    protected void onCreate(Bundle savedInstanceState){
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.nombreslista);
        //
        listView = (ListView) findViewById( R.id.list );

        sqlite = new SQLite( this );
        try {
            sqlite.abrir();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        Cursor cursor = sqlite.getRegistros();
        ArrayList<String> listData = sqlite.getFormatList( cursor );
        adaptador = new ArrayAdapter<String>( this ,
                android.R.layout.simple_list_item_1  , listData );
        listView.setAdapter( adaptador );
        listView.setOnItemClickListener( this );

        if( listData.size()== 0 )
        {
            Toast.makeText(getBaseContext(), "No records"
                    ,Toast.LENGTH_SHORT).show();
        }

    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.lstcmp, menu);
        return true;
    }

    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        //
        Object object = listView.getItemAtPosition( position );
        //Se extrae el ID = [X]
        int posicionInicial = object.toString().indexOf("[") + 1;
        int posicionFinal = object.toString().indexOf("]",posicionInicial);
        String resultado =  object.toString().substring(posicionInicial, posicionFinal);
        //ejecuta nueva actividad
        Bundle b = new Bundle();
        b.putInt("id", Integer.valueOf(resultado) );
        Intent iRegs = new Intent( listcomp.this, carcomp.class );
        iRegs.putExtras(b);
        startActivity( iRegs );
    }
}
```

Carcomp.java

Como se indicó en páginas anteriores, esta clase es la encargada de mostrar las características del componente seleccionado anteriormente en la base de datos, además de ofrecer al usuario la posibilidad de interactuar con la misma. Para ello, en primer lugar obtendrá los datos del componente y los mostrará, y a continuación ejecutará una función diferente en función del botón seleccionado (Ver Anexo IV).

```
public class carcomp extends Activity implements OnClickListener{

    private TextView textView;
    private Button btnNuevo;
    private Button btnDelete;
    private Button btnLista;
    private Button btnedit;
    private SQLite sqlite;


   protected void onCreate(Bundle savedInstanceState) {
       getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
               WindowManager.LayoutParams.FLAG_FULLSCREEN);
       super.onCreate(savedInstanceState);
       setContentView(R.layout.details) ;

       textView = (TextView) findViewById( R.id.txtResultado );
       btnNuevo = (Button) findViewById( R.id.btnNuevo );
       btnDelete = (Button) findViewById( R.id.btnDelete );
       btnLista = (Button) findViewById( R.id.btnLista );
       btnedit = (Button) findViewById(R.id.btnedit);
       btnNuevo.setOnClickListener( this );
       btnLista.setOnClickListener( this );
       btnDelete.setOnClickListener( this );
       btnedit.setOnClickListener( this );
       textView.setTextSize(14);
       //Recupera parametro ID de registro
       textView.setText( ""  );
       Intent i = getIntent();
       Bundle bundle = i.getExtras();
       if ( bundle != null ) {
            int id = bundle.getInt("id");

            //base de datos
            sqlite = new SQLite( this );
            try {
               sqlite.abrir();
           } catch (IOException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
           }
            Cursor cursor = sqlite.getRegistro(id);
            ArrayList<String> reg = sqlite.getFormatList2(cursor);
            textView.setText( reg.get(0)  );
        }
   }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.lstcmp, menu);
        return true;
    }

    public void onClick(View v ) {
        switch ( v.getId() )
        {
            case R.id.btnNuevo:
                Intent iMain = new Intent( carcomp.this, nuevo.class );
                startActivity( iMain );
                break;
            case R.id.btnDelete:
                //Muestra una ventana de dialo para confirmar eliminacion de registro
                new AlertDialog.Builder(this)
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setTitle("Delete")
                .setMessage("Really want to delete this record?")
                .setPositiveButton("Yes", new DialogInterface.OnClickListener() {


                    public void onClick(DialogInterface dialog, int which) {
                        //Se extrae el ID = [X]
                        int posicionInicial = textView.getText().
                                toString().indexOf("[") + 1;
                        int posicionFinal = textView.getText().
                                toString().indexOf("]",posicionInicial);
                        String resultado =  textView.getText().
                                toString().substring(posicionInicial, posicionFinal);
                        //Elimina registro y abre activity RegistrosActivity.class
                        if(sqlite.borrar_registro( Integer.valueOf(resultado) ))
                        {
                            goRegistrosActivity();
                        }
                    }

                })
                .setNegativeButton("No", null)
                .show();

                break;
            case R.id.btnLista:
                goRegistrosActivity();
                break;

            case R.id.btnedit:
                int posicionInicial = textView.getText().
                toString().indexOf("[") + 1;
                int posicionFinal = textView.getText().
                        toString().indexOf("]",posicionInicial);
                String resultado =  textView.getText().
                        toString().substring(posicionInicial, posicionFinal);
                sqlite.borrar_registro( Integer.valueOf(resultado));
                Intent edition = new Intent( carcomp.this, edit.class );
                startActivity( edition );
                break;
        }
    }

    public void goRegistrosActivity()
    {
        Intent iRegs = new Intent( carcomp.this, listcomp.class );
        startActivity( iRegs );
    }
}
```

Nuevo.java

Esta clase simplemente se encarga de recoger los datos introducirlos por el usuario y añadirlos a la base de datos, para el posterior uso del componente generado.

```
public class nuevo extends Activity implements OnClickListener{

    private EditText txttc;
    private EditText txtName;
    private EditText txtpc;
    private EditText txtfa;
    private EditText txtfc;
    private EditText txtpm;
    private EditText txtca;
    private EditText txtcb;
    private EditText txtcc;
    private EditText txtcd;

    private Button btnRegistrar;
    private Button btnCancelar;
    private Button btnRegistros;
    private SQLite sqlite;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.insertarnuevo);

        txtName = (EditText) findViewById( R.id.txtName );
        btnRegistrar = (Button) findViewById(R.id.btnRegistrar );
        btnRegistrar.setOnClickListener( this );
        btnCancelar = (Button) findViewById(R.id.btnCancelar );
        btnCancelar.setOnClickListener( this );
        btnRegistros = (Button) findViewById(R.id.btnRegistros );
        btnRegistros.setOnClickListener( this );
        txttc = (EditText) findViewById(R.id.txttc );
        txtpc = (EditText) findViewById(R.id.txtpc );
        txtfa = (EditText) findViewById(R.id.txtfa );
        txtfc = (EditText) findViewById(R.id.txtfc );
        txtpm = (EditText) findViewById(R.id.txtpm );
        txtca = (EditText) findViewById(R.id.txtca );
        txtcb = (EditText) findViewById(R.id.txtcb );
        txtcc = (EditText) findViewById(R.id.txtcc );
        txtcd = (EditText) findViewById(R.id.txtcd );

        //String[] components = getResources().getStringArray(R.array.componentes);
        //Obtiene fecha actual y coloca en el textview

        //base de datos
        sqlite = new SQLite( this );
        try {
            sqlite.abrir();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //
    }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public void onClick(View v) {
        switch ( v.getId() )
        {
            case R.id.btnRegistrar:

                //Registra en la base de datos
                if ( sqlite.addRegistro( txtName.getText().toString(),
                                         txttc.getText().toString(),
                                         txtpc.getText().toString(),
                                         txtfa.getText().toString(),
                                         txtfc.getText().toString(),
                                         txtpm.getText().toString(),
                                         txtca.getText().toString(),
                                         txtcb.getText().toString(),
                                         txtcc.getText().toString(),
                                         txtcd.getText().toString()) )
                {
                    //recupera ID de ultimo registro y pasa como parametro
                    int id = sqlite.getUltimoID();
                    Bundle bundle = new Bundle();
                    bundle.putInt("id", id);
                    Intent intent = new Intent( nuevo.this, carcomp.class );
                    intent.putExtras( bundle );
                    startActivity( intent );
                }

                 else
                 {
                     Toast.makeText(getBaseContext(),
                             "Error: Compruebe que los datos sean correctos"
                             ,Toast.LENGTH_SHORT).show();
                 }
                 break;
             case R.id.btnCancelar: sqlite.cerrar(); finish(); break;
             case R.id.btnRegistros:
                 Intent iRegs = new Intent( nuevo.this, listcomp.class );
                 startActivity( iRegs );
                 break;
        }
    }
}
```

Edit.java

El código es prácticamente el mismo que el de la clase ‘nuevo’, por lo que no se añadirá a la memoria, con la salvedad de que, antes de permitir al usuario añadir el nuevo registro, elimina el antiguo.

Graphic.java

La clase 'Graphic' solicita al usuario que seleccione las características de la gráfica que quiere obtener. Como ya se comentó en el Anexo IV, las gráficas binarias no están implementadas, pero en el código queda descrita la estructura que abría que seguir en futuras versiones. Las líneas correspondientes a las gráficas binarias están comentadas, como se puede ver a continuación.

Una vez seleccionadas las características, las almacenará para que la siguiente clase pueda acudir a ellas cuando pretenda graficar el compuesto.

```
public class graphic extends Activity{

    private ListView list;
    private RadioGroup radio;
    private Bundle ecuacion;
    private RadioButton puro;
    private ArrayAdapter<String> listAdapter;
    private Button btn1,btn2,btn3;
    int ec = 10;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.graficas);
        super.onCreate(savedInstanceState);


        list = (ListView) findViewById(R.id.list);
        radio = (RadioGroup) findViewById(R.id.RadioGroupg);
        puro = (RadioButton) findViewById(R.id.pure);
        btn1 = (Button) findViewById(R.id.Btn);
//      btn2 = (Button) findViewById(R.id.Btn2);
        btn3 = (Button) findViewById(R.id.buttonback);

        ecuacion = getIntent().getExtras();
        ArrayList<String> optionlist = new ArrayList<String>();
        String[] eos =getResources().getStringArray(R.array.eosg);
        optionlist.addAll( Arrays.asList(eos) );
        listAdapter = new ArrayAdapter<String>(this, R.layout.rowlist, optionlist);
        list.setAdapter( listAdapter );
        list.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
//      list.setItemChecked(st, true);
        btn1.setVisibility(View.VISIBLE);
//      btn2.setVisibility(View.INVISIBLE);

        btn3.setVisibility(View.VISIBLE);

        puro.setOnCheckedChangeListener(

                new RadioButton.OnCheckedChangeListener() {
```

```
                    //Si se selecciona el componente puro, muestra el botón para dibujar
                    public void onCheckedChanged(CompoundButton buttonView,
                            boolean isChecked) {

                        if (isChecked) {

                            btn1.setVisibility(View.VISIBLE);
//                          btn2.setVisibility(View.INVISIBLE);

                        }
                        //Si se selecciona la gráfica binaria, aparece aviso
                        else {

                            btn1.setVisibility(View.INVISIBLE);
                            Toast toast=Toast.makeText(getApplicationContext()
                                    ,getResources().getString(R.string.noim),
                                    Toast.LENGTH_LONG);
                            toast.show();
                        //  btn2.setVisibility(View.VISIBLE);
                        }
                    }
                });

        btn3.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {

                Intent intent= new Intent();
                ecuacion.putInt("EOFS", getec());
                ecuacion.putBoolean("a", getCh());
                intent.putExtras(ecuacion);
                setResult(0,intent);
                finish();
                overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);


            }

        });

         btn1.setOnClickListener(new View.OnClickListener() {

             @Override
             public void onClick(View v) {
                 Intent draw1 = new Intent(graphic.this, drawpure.class);
                 ecuacion.putInt("EOFS", getec());
                 ecuacion.putBoolean("a", getCh());
                 draw1.putExtras(ecuacion);
                 finish();
                 startActivity(draw1);
                 overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);

             }
         });

/*       btn2.setOnClickListener(new View.OnClickListener() {

             @Override
             public void onClick(View v) {
                 Intent draw2 = new Intent(graphic.this, drawbinary.class);
                 ecuacion.putInt("EOFS",getec());
                 draw2.putExtras(ecuacion);
                 startActivityForResult(draw2,0);
                 overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);

             }
         });*/
    }
```

```
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if(requestCode == 0){
            ecuacion = data.getExtras();

        }
    }
    public int getec() {

            if(list.isItemChecked(0)==true){
                ec=1;}
            else{if(list.isItemChecked(1)==true){
                ec=2;}
            else{if(list.isItemChecked(2)==true){
                ec=3;}
            else{
                ec=10;
                }}}

        return ec;
    }

    public boolean getCh() {

        boolean res=false;

        return res;
    }

}
```

Drawpure.java

Esta clase solicita al usuario el resto de datos necesarios para realizar la gráfica pertinente. En este caso será: Componente a graficar, gráfica que mostrar, y rango del eje de abscisas.

Es importante rellenar todos los campos, ya que si no la aplicación no generará el dibujo correctamente.

```
public class drawpure extends Activity implements TextWatcher{

    private SQLite sqlite;
    private ArrayAdapter<String> listAdapter;
    private String item;
    private RadioButton pvt;
    private EditText inicial, finale, b, c;
    private Button gra, back;
    private Bundle bundle;
    private AutoCompleteTextView edit, a;
    private float f, f2;
    double ecu, t;
    int j;
```

```
public void Init(){


    getWindow().setSoftInputMode(WindowManager.LayoutParams.
            SOFT_INPUT_STATE_ALWAYS_HIDDEN);

    sqlite = new SQLite( this );
    try {
        sqlite.abrir();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    Cursor cursor = sqlite.getRegistrosComponentes();
    ArrayList<String> optionlista = sqlite.getFormatList0( cursor );
    listAdapter = new ArrayAdapter<String>(this, R.layout.list, optionlista);
    edit = (AutoCompleteTextView)findViewById(R.id.editd);
    inicial=(EditText) findViewById(R.id.sub0);
    finale=(EditText) findViewById(R.id.subf);
    pvt = (RadioButton) findViewById(R.id.pvt);
    gra=(Button) findViewById(R.id.Btngra);
    back=(Button) findViewById(R.id.bback);

    item="PLEASE SELECT...";

    a=edit;

    a.addTextChangedListener(this);
    a.setAdapter(listAdapter);
    a.setText("PLEASE SELECT...");
    a.setSingleLine(true);


    //Selección del componente deseado
    a.setOnKeyListener(new OnKeyListener()
    {
            public boolean onKey(View v, int keyCode, KeyEvent event)
        {
            if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER &&
                    event.getAction() == KeyEvent.ACTION_DOWN)
            {
                    InputMethodManager imm = (InputMethodManager)
                            getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow
                (a.getWindowToken(), 0);
                return true;
            }
            if (keyCode == KeyEvent.KEYCODE_ENTER &&
                    event.getAction() == KeyEvent.ACTION_DOWN)
            {
                    InputMethodManager imm = (InputMethodManager)
                            getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow(a.getWindowToken(), 0);
                    return true;
            }
            return false;
        }
    });
```

```
    //Selección de valor inicial del eje de abscisas
    b.setOnKeyListener(new OnKeyListener()
    {

        public boolean onKey(View v, int keyCode, KeyEvent event)
        {
            if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER &&
                    event.getRepeatCount() == 0)
            {

                InputMethodManager imm = (InputMethodManager)
                        getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow(a.getWindowToken(), 0);
                return true;
            }
            if (keyCode == KeyEvent.KEYCODE_ENTER &&
                    event.getRepeatCount() == 0)
            {

                InputMethodManager imm = (InputMethodManager)
                        getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow(a.getWindowToken(), 0);

                return true;
            }
            return false;
        }
    });
     //Selección de valor final del eje de abscisas
     c.setOnKeyListener(new OnKeyListener()
     {

         public boolean onKey(View v, int keyCode, KeyEvent event)
         {
             if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER &&
                     event.getRepeatCount() == 0)
             {

                 InputMethodManager imm = (InputMethodManager)
                         getSystemService(Context.INPUT_METHOD_SERVICE);
                 imm.hideSoftInputFromWindow(a.getWindowToken(), 0);
                 return true;
             }
             if (keyCode == KeyEvent.KEYCODE_ENTER &&
                     event.getRepeatCount() == 0)
             {

                 InputMethodManager imm = (InputMethodManager)
                         getSystemService(Context.INPUT_METHOD_SERVICE);
                 imm.hideSoftInputFromWindow(a.getWindowToken(), 0);

                 return true;
             }
             return false;
         }
     });
     b.setText("0.0");
     c.setText("0.0");
     b.setSingleLine(true);
     c.setSingleLine(true);

}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    getWindow().setFlags(WindowManager.LayoutParams.
            FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.drawpuro);
    super.onCreate(savedInstanceState);

    Init();
    bundle = getIntent().getExtras();

    ecu = bundle.getInt("EOFS");

    edit.setOnItemClickListener(new OnItemClickListener() {

        public void onItemClick(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {


        item= arg0.getItemAtPosition(arg2).toString();


    }});
    //Recupera los datos y los pasa a la siguiente clase
    gra.setOnClickListener(new View.OnClickListener() {
         public void onClick(View v) {
             if(pvt.isChecked()){
                 Intent show = new Intent(drawpure.this, Lienzop1.class);
                 bundle.putStringArray("comp", getComp());
                 bundle.putFloat("tinicial", getDatai());
                 bundle.putFloat("tfinal", getDataf());
                 bundle.putDouble("presion", -1.0);
                 bundle.putDouble("estado", 1.0);
                 bundle.putDoubleArray("fr", getFr());
                 bundle.putInt("j",  getj());
                 show.putExtras(bundle);
                 finish();
                 startActivity(show);
                 overridePendingTransition(R.anim.push_left_in,
                         R.anim.push_left_out);
             }else{
                 Intent show = new Intent(drawpure.this, Lienzop3.class);
                 bundle.putStringArray("comp", getComp());
                 bundle.putFloat("pinicial", getPai());
                 bundle.putFloat("pfinal", getPaf());
                 bundle.putDouble("temperatura", -2000);
                 bundle.putDoubleArray("fr", getFr());
                 bundle.putDouble("estado", 1.0);
                 show.putExtras(bundle);
                 finish();
                 startActivity(show);
                 overridePendingTransition(R.anim.push_left_in,
                         R.anim.push_left_out);
             }


         }

    });
```

```
        //Regresa a la clase anterior
        back.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {

                Intent back = new Intent(drawpure.this, graphic.class);
                bundle.putStringArray("comp", getComp());
                bundle.putFloat("tinicial", getDatai());
                bundle.putFloat("tfinal", getDataf());
                bundle.putDouble("estado", 1.0);
                bundle.putDoubleArray("fr", getFr());
                bundle.putInt("j",  getj());
                back.putExtras(bundle);
                startActivity(back);
                finish();
                overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);

            }

        });
    }

    public void afterTextChanged(Editable arg0) {


    }
    public void beforeTextChanged(CharSequence s, int start, int count,
            int after) {


    }
    public void onTextChanged(CharSequence s, int start, int before, int count) {


    }


      //Obtención de datos
      public String [] getComp(){

          String[] A={item};

          return A;
      }

      public float getDatai(){

          try{f=Float.parseFloat(inicial.getText().toString());
          }
          catch(NumberFormatException e){f=0;}

          float F=f;

          return F;
      }

  public float getDataf(){

          try{f2=Float.parseFloat(finale.getText().toString());
          }
          catch(NumberFormatException e){f=0;}

          float F=f2;

          return F;
      }
```

```
public double [] getFr(){

        try{t=Double.parseDouble("1.0");
        }
        catch(NumberFormatException e){t=1.0;}


        double [] T={t};

        return T;
    }

public int getj(){
    ;

    return j;}

public float getPai(){
    float Pp= 0;
    try{Pp = Float.parseFloat(inicial.getText().toString());
    }
    catch(NumberFormatException e){Pp=0;}




    return Pp;}

public float getPaf(){
    float Pp= 0;
    try{Pp = Float.parseFloat(finale.getText().toString());
    }
    catch(NumberFormatException e){Pp=0;}

    return Pp;}

}
```

Lienzop1.java y Lienzop3.java

En estas clases se dibujará la gráfica con los datos seleccionados anteriormente.

Podemos distinguir dos partes en ambos códigos. La primera correspondería a los cálculos realizados para obtener los puntos que serán representados posteriormente. Estos cálculos son iguales que los expuestos en la clase 'resultados', por lo que no se mostrarán aquí.

La segunda parte está formada por una clase auxiliar, que es la que define el lienzo sobre el que se dibujará, y los ejes de coordenadas. De forma relativa a la posición de dichos ejes, se posicionarán los puntos calculados, que posteriormente se unirán para formar la curva que se desea obtener.

La única diferencia entre ambas clases auxiliares, es que la primera obtiene los valores de la temperatura para el eje de abscisas, mientras que la segunda hace lo propio con los valores de la presión.

- *Clase auxiliar 'Lienzop2' (perteneciente a Lienzop1.java)*

```
class Lienzop2 extends View {

    public Lienzop2(Context context) {

        super(context);
    }

    public void onDraw(Canvas canvas) {

        paint.setColor(Color.BLACK);
        paint.setStrokeWidth(4);
        float [] pts =  new float[50];
        float [] temp = new float[12];
        float paso1 = (data1-data0)/5;
        double p = (P2-P1)/5;
        float paso2 = (float) p;
        float Pr1 = (float) P1;
        float Pr2 = (float) P2;
        float Pr3 = (float) P3;
        float Pr4 = (float) P4;
        float Pr5 = (float) P5;
        float Pr6 = (float) P6;
        float posx1, posx2, posx3, posx4, posx5, posx6, posy1,
        posy2, posy3, posy4, posy5, posy6;

        //Se definen los ejes de coordenadas por medio de lineas
        pts[0]=150; pts[1]=190; pts[2]=150; pts[3]=810;
        pts[4]=150; pts[5]=810; pts[6]=610; pts[7]=810;
        pts[8]=140; pts[9]=290; pts[10]=160; pts[11]=290;
        pts[12]=140; pts[13]=390; pts[14]=160; pts[15]=390;
        pts[16]=140; pts[17]=490; pts[18]=160; pts[19]=490;
        pts[20]=140; pts[21]=590; pts[22]=160; pts[23]=590;
        pts[24]=140; pts[25]=690; pts[26]=160; pts[27]=690;
        pts[28]=230; pts[29]=800; pts[30]=230; pts[31]=820;
        pts[32]=310; pts[33]=800; pts[34]=310; pts[35]=820;
        pts[36]=390; pts[37]=800; pts[38]=390; pts[39]=820;
        pts[40]=470; pts[41]=800; pts[42]=470; pts[43]=820;
        pts[44]=550; pts[45]=800; pts[46]=550; pts[47]=820;

        //Se colocan los puntos calculados en un vector
        temp[0] = data0; temp[1] = Pr1;
        temp[2] = data0 + paso1; temp[3] = Pr1 + paso2;
        temp[4] = data0 + 2*paso1; temp[5] = Pr1 + 2*paso2;
        temp[6] = data0 + 3*paso1;  temp[7] = Pr1 + 3*paso2;
        temp[8] = data0 + 4*paso1; temp[9] = Pr1 + 4*paso2;
        temp[10] = data1; temp[11] = Pr2;
```

```
        canvas.drawLines(pts, paint);
        write.setColor(Color.BLACK);
        write.setTextSize(25);
        write1.setTextSize(60);
        write1.setStrokeWidth(5);
        canvas.drawText("T", 600, 860, write);
        canvas.drawText("P", 110, 220, write);
        canvas.drawText("0", 110, 860, write);

        //Se posiciona cada punto en su coordenada correspondiente
        canvas.drawText(String.format("%.1f", temp[10]), 525, 860, write);
        canvas.drawText(String.format("%.1f", temp[11]), 90, 300, write);
        canvas.drawText(compound[0], 30, 70, write1);
        puntos.setStrokeWidth(4);
        posx1 = 150 + 400*data0/data1;
        posx2 = 150 + 400*(data0+paso1)/data1;
        posx3 = 150 + 400*(data0+2*paso1)/data1;
        posx4 = 150 + 400*(data0+3*paso1)/data1;
        posx5 = 150 + 400*(data0+4*paso1)/data1;
        posx6 = 150 + 400*(data1)/data1;
        posy1 = 810 - 510*Pr1/Pr2;
        posy2 = 810 - 510*(Pr3)/Pr2;
        posy3 = 810 - 510*(Pr4)/Pr2;
        posy4 = 810 - 510*(Pr5)/Pr2;
        posy5 = 810 - 510*(Pr6)/Pr2;
        posy6 = 810 - 510*(Pr2)/Pr2;
        canvas.drawPoint(posx1, posy1, puntos);
        canvas.drawPoint(posx2, posy2, puntos);
        canvas.drawPoint(posx3, posy3, puntos);
        canvas.drawPoint(posx4, posy4, puntos);
        canvas.drawPoint(posx5, posy5, puntos);
        canvas.drawPoint(posx6, posy6, puntos);


           //Se unen los puntos calculados
           unir.setStrokeWidth(1);
           unir.setColor(Color.RED);
           canvas.drawLine(posx1, posy1, posx2, posy2, unir);
           canvas.drawLine(posx2, posy2, posx3, posy3, unir);
           canvas.drawLine(posx3, posy3, posx4, posy4, unir);
           canvas.drawLine(posx4, posy4, posx5, posy5, unir);
           canvas.drawLine(posx5, posy5, posx6, posy6, unir);
    }
}
```

- *Clase auxiliar 'Lienzop4' (perteneciente a Lienzop3.java)*

```
class Lienzop4 extends View {

    public Lienzop4(Context context) {

        super(context);
    }

    public void onDraw(Canvas canvas) {

        paint.setColor(Color.BLACK);
        paint.setStrokeWidth(4);
        float [] pts =  new float[50];
        float [] temp = new float[12];
        float paso1 = (data1-data0)/5;
        double p = (T2-T1)/5;
        float paso2 = (float) p;
        float Pr1 = (float) ((float) T1 - 273.15);
        float Pr2 = (float) ((float) T2 - 273.15);
        float Pr3 = (float) ((float) T3 - 273.15);
        float Pr4 = (float) ((float) T4 - 273.15);
        float Pr5 = (float) ((float) T5 - 273.15);
        float Pr6 = (float) ((float) T6 - 273.15);
        float posx1, posx2, posx3, posx4, posx5, posx6, posy1,
        posy2, posy3, posy4, posy5, posy6;

      //Se definen los ejes de coordenadas por medio de lineas

      pts[0]=150; pts[1]=190; pts[2]=150; pts[3]=810;
      pts[4]=150; pts[5]=810; pts[6]=610; pts[7]=810;
      pts[8]=140; pts[9]=290; pts[10]=160; pts[11]=290;
      pts[12]=140; pts[13]=390; pts[14]=160; pts[15]=390;
      pts[16]=140; pts[17]=490; pts[18]=160; pts[19]=490;
      pts[20]=140; pts[21]=590; pts[22]=160; pts[23]=590;
      pts[24]=140; pts[25]=690; pts[26]=160; pts[27]=690;
      pts[28]=230; pts[29]=800; pts[30]=230; pts[31]=820;
      pts[32]=310; pts[33]=800; pts[34]=310; pts[35]=820;
      pts[36]=390; pts[37]=800; pts[38]=390; pts[39]=820;
      pts[40]=470; pts[41]=800; pts[42]=470; pts[43]=820;
      pts[44]=550; pts[45]=800; pts[46]=550; pts[47]=820;

      //Se colocan los puntos calculados en un vector
      temp[0] = data0; temp[1] = Pr1; temp[2] = data0 + paso1;
      temp[3] = Pr1 + paso2;  temp[4] = data0 + 2*paso1;
      temp[5] = Pr1 + 2*paso2; temp[6] = data0 + 3*paso1;
      temp[7] = Pr1 + 3*paso2; temp[8] = data0 + 4*paso1;
      temp[9] = Pr1 + 4*paso2; temp[10] = data1; temp[11] = Pr2;

      //Se posiciona cada punto en su coordenada correspondiente
      canvas.drawLines(pts, paint);
      write.setColor(Color.BLACK);
      write.setTextSize(25);
      write1.setTextSize(60);
      write1.setStrokeWidth(5);
      canvas.drawText("T", 600, 860, write);
      canvas.drawText("P", 110, 220, write);
      canvas.drawText("0", 110, 860, write);
```

```
            canvas.drawText(String.format("%.1f", temp[10]), 525, 860, write);
            canvas.drawText(String.format("%.1f", temp[11]), 75, 300, write);
            canvas.drawText(compound[0], 30, 70, write1);
            puntos.setStrokeWidth(4);
            posx1 = 150 + 400*data0/data1;
            posx2 = 150 + 400*(data0+paso1)/data1;
            posx3 = 150 + 400*(data0+2*paso1)/data1;
            posx4 = 150 + 400*(data0+3*paso1)/data1;
            posx5 = 150 + 400*(data0+4*paso1)/data1;
            posx6 = 150 + 400*(data1)/data1;
            posy1 = 810 - 510*Pr1/Pr2;
            posy2 = 810 - 510*(Pr3)/Pr2;
            posy3 = 810 - 510*(Pr4)/Pr2;
            posy4 = 810 - 510*(Pr5)/Pr2;
            posy5 = 810 - 510*(Pr6)/Pr2;
            posy6 = 810 - 510*(Pr2)/Pr2;
            canvas.drawPoint(posx1, posy1, puntos);
            canvas.drawPoint(posx2, posy2, puntos);
            canvas.drawPoint(posx3, posy3, puntos);
            canvas.drawPoint(posx4, posy4, puntos);
            canvas.drawPoint(posx5, posy5, puntos);
            canvas.drawPoint(posx6, posy6, puntos);

            //Se unen los puntos calculados
            unir.setStrokeWidth(1);
            unir.setColor(Color.RED);
            canvas.drawLine(posx1, posy1, posx2, posy2, unir);
            canvas.drawLine(posx2, posy2, posx3, posy3, unir);
            canvas.drawLine(posx3, posy3, posx4, posy4, unir);
            canvas.drawLine(posx4, posy4, posx5, posy5, unir);
            canvas.drawLine(posx5, posy5, posx6, posy6, unir);
        }
    }
```

Todo el código expuesto anteriormente es, a grandes rasgos, el implementado en lenguaje java para el correcto funcionamiento de la aplicación. No obstante, ésta emplea otras clases implementadas en la versión anterior de la aplicación (Cabeza Sánchez, 2012).

Por otra parte se encuentran los archivos '.xml', en los cuales se programa la estructura de cada una de las pantallas, pero que no se mostrará en el anexo debido a la densidad de las mismas. A pesar de ello, cabe resaltar el hecho de que dichos códigos han sido implementados de tal forma que, independientemente del tamaño que tenga la pantalla del dispositivo empleado, todos los elementos de la aplicación se reubican para mostrar siempre la misma apariencia, lo que facilita el uso de la aplicación y su comprensión por parte del usuario que la utilice.

El código completo de la aplicación puede encontrarse en el CD adjunto.