Un Ambiente de desarrollo para Interfaces de Usuarios Adaptativas

Lic. Javier Diaz (jdiaz@ada.info.unlp.edu.ar)
Lic. Laura Fava (lfava@ada.info.unlp.edu.ar)
Departamento de Informática.
Universidad Nacional de La Plata.
Buenos Aires. Argentina.

Un sistema adaptativo es aquel que tiene un comportamiento dinámico frente a diferentes usuarios, por lo tanto requiere inevitablemente de un modelo de estereotipos de usuarios y/o de un modelo explícito del usuario particular del sistema adaptativo. Este paper describe un gerenciador de Interfaces de Usuarios Adptativas construído para facilitarles a los diseñadores de Sistemas Adaptivas administrar los Modelos de Usuario así como también crear y mantener la conexión entre la Interfaz de Usuario y la aplicación.

1. Introducción

Un sistema adaptativo es definido 'como un sistema basado en conocimiento el cual automáticamente altera aspectos de la funcionalidad e interfaz del sistema para acomodarse a las diferentes preferencias y requerimientos de un usuario individual' [DM88]. La construcción de interfaces con estas características sin una herramienta que facilite cada una de las etapas de desarrollo, es una tarea sumamente difícil; para poder lograr la customización necesariamente debe disponerse de modelos de clases de usuarios y/o de modelos explícitos de usuarios. Sin modelos de usuarios la aplicación debería tener opciones especiales para customizar la interfaz de acuerdo al usuario o tipo de usuario que está interactuando, por lo tanto las posibilidades se verían limitadas a un pequeño número de casos o, de lo contrario la aplicación se volvería incodificable. El *modelo de usuario* es imprescindible en este tipo de sistemas, los sistemas que carecen de éste modelo y se adaptan -por configuración, set up, etc.- no son sistemas adaptativos, sino simplemente, sistemas adaptables.

No todos los ambientes de desarrollo de interfaz tradicionales facilitan y/o toleran la construcción de este tipo de Interfaces, en este artículo mostramos un ambiente para el desarrollo de Interfaces Adaptativas -AUI-, que permite el diseño de la interfaz y la construcción y mantenimiento de los modelos de usuarios. Este ambiente de desarrollo fue montado sobre Smalltalk V y constituye un sistema de alta complejidad en si mismo, cumple con todos los requisitos para considerarse una UIMS. y constituyó la tesis para la obtención de la Licenciatura en Informática en la Universidad Nacional de La Plata. CUIMS* -nombre dado a la UIMS-, ha sido concluída y testeada por alumnos de 5º año de la Cátedra de *'Prototipación e Interfaces de Usuarios'* .

CUIMS* Es una UIMS que fue desarrollada como trabajo de grado para la Licenciatura en Informática en la Universidad Nacional de La Plata en conjunto con la Lic. Marisa Lacosta y con la dirección del Lic. Javier Diaz.

2. Conexión de una Interfaz Adaptativa y una Arquitectura de UIMS

La incorporación de adaptabilidad a la Interfaz de Usuario, requiere conocimientos específicos, que extiende el uso de las UIMS clásicas. En este sentido, durante la construcción de la interfaz adaptativa, la herramienta es utilizada por personas con distintos roles, la siguiente figura gráfica estas interacciones asi como tambien la del Usuario Final con la AUI.

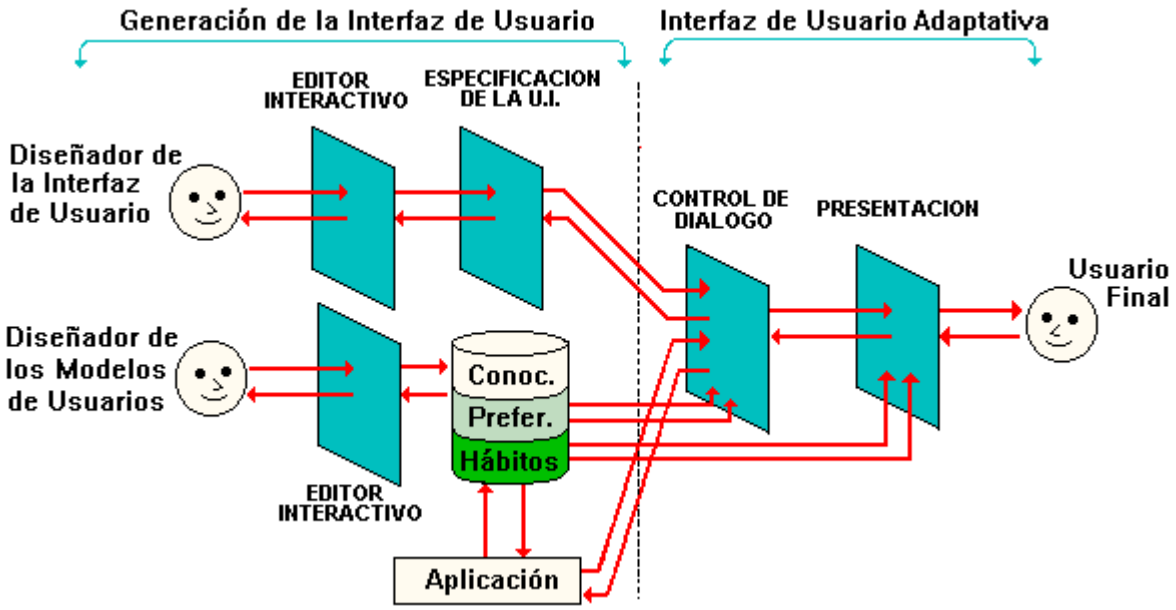
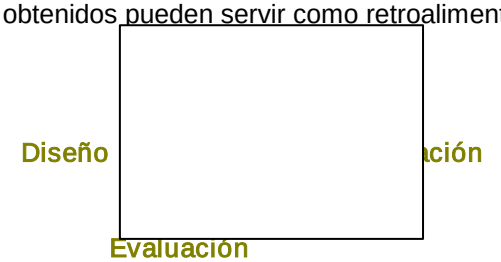


Figura 1 : Interfaz Adaptativa y Arquitectura de una UIMS

El *Diseñador de la Interfaz de Usuario* puede utilizar la herramienta para desarrollar todo el ciclo de construcción de la Interfaz (Diseño->Implementación->Evaluación->ReDiseño...). Para construir la parte gráfica de la UI, la herramienta le permite construir/insertar objetos gráficos y establecer su comportamiento dinámico para su asociación con la aplicación subyacente y con la interfaz en general. A medida que se va diseñando/implementando la interfaz puede ser evaluada sin necesidad de salir del ambiente para compilar, linkeditar y correr, puede ser testeada por el diseñador y por usuarios, y los resultados obtenidos pueden servir como retroalimentación para continuar el ciclo de desarrollo.



El *Diseñador de los Modelos* de Usuarios la utiliza para definir los Modelos de Usuarios que son consultados por la aplicación para adaptar la interfaz y son alimentados por las sucesivas interacciones del usuario con el sistema adaptativo. El Modelo de Usuario que define CUIMS esta formado por tres tipos de patrones, a saber :

- ***Patrón de Hábitos*** : esta compuesto por hábitos del usuario, esto es, secuencias de comandos que el usuario haya aplicado frecuentemente. Para determinar estas secuencias analizamos la historia de las interacciones del usuario con el sistema. La existencia de esta customización alivia al usuario

la tarea de desarrollar repetidas veces la misma secuencia de comandos o le evita la tarea de crear macros específicas para esas secuencias. Un hábito tiene 2 estados posibles : **Activo / Pasivo**. Cuando un hábito es detectado, por default, está en estado activo, el usuario puede en cualquier momento desactivarlo por no encontrarlo conveniente o apropiado en el contexto. Cuando la Interfaz detecta que una secuencia que conforma un hábito esta produciéndose y el hábito esta en estado Activo, ésta la finaliza por el usuario, automáticamente.

- ***Patrón de Preferencias*** : esta compuesto por un conjunto de características observadas en un usuario particular, específicamente preferencias en cuanto aspectos presentacionales de la Interfaz. La adaptación de los aspectos de la presentación de las interfaces, consiste en cambiar los estilos de letras, colores de los objetos de interacción y de las ventanas, gráficos de acuerdo al usuario que este interactuando.
- ***Patrón de Conocimiento*** : este modelo brinda un nivel de conocimiento del usuario respecto a la comprensión que este tenga de la aplicación que esta utilizando. Los helps pueden adquirir distintos aspectos, ya sea totalmente gráficos, totalmente textuales o combinaciones de ambos de acuerdo al nivel cognitivo del usuario.

Por último el tercer rol lo cumple el *Usuario Final* quien interactúa con la interfaz adaptativa de una manera natural y sin tener que adaptarse a lo que le imponga la interfaz.

Los Editores interactivos los utilizan los diseñadores para especificar la Interfaz de Usuario y los Modelo de Usuarios, y al especificación de la Interfaz de Usuario es generada automáticamente por la misma herramienta. El control de Diálogo define la estructura del diálogo entre el usuario y la aplicación.

3. CUIMS

CUIMS es una UIMS que soporta todas las etapas del ciclo de construcción de la Interfaz de Usuario Adaptativa (UIA). Esta herramienta fue construida sobre Smalltalk V y provee la construcción de Interfaz de Usuario por medio de manipulación directa y fácil conexión y mantenimiento con la aplicación.

Cada UIA generada automáticamente por CUIMS está compuesta por objetos, formados como lo indica el paradigma MVC por 3 componentes Model-View-Controller. En el paradigma MVC el Model contiene los datos específicos de la aplicación, la View despliega esos datos sobre la pantalla y el Controller maneja las interacciones del usuario que afectan a los dos anteriores [K&P88].

Las virtudes del MVC son:

- Permite múltiples *Views* del mismo *Model.*
- Permite reusar componentes porque las Views son construidas desde subViews.
- Permite cambiar fácilmente el comportamiento de la aplicación cambiando el *Controller.*

En CUIMS el cambio mas notable, se ve en la relación que existe entre el Model y las Views, porque para un Model existen una variedad de Views y esas Views cambian cuando el usuario cambia. Para poder lograr esto en todas las interfaces debimos modificar la jerarquía de Smalltalk agregando algunas clases nuevas :

Objetos de Interacción

AnimationP { pane para animación }
Botones { comportamiento común para botones }
Boton
Check
Radio
ThreeState
DrawnB
EntryF { campos de edición }
GraphP { pane para gráficos }
GroupB { pane para agrupar objetos como Radio_Buttons, Check_Boxes, etc. }
Listas { comportamiento común para listas }
Combo
List
MultipleList
StaticB { pane para líneas, figuras, etc. }
StaticT { panel para texto estático }
TextP { pane para editar amplios textos }

Cada uno de estos Objetos de Interacción tiene características y comportamiento propios; cada uno de ellos esta compuesto por un conjunto de atributos, que son instancias de la clase Atributos. *Atributos* es otra clase incorporada para los efectos de customización. Por ejemplo una instancia de Check esta formada por un conjunto de atributos: color de fondo, font, un Bag en donde se guardan los pedidos de cambio en algún valor del Objeto de Interacción -OI-, efectuados por el usuario en las distintas sesiones.

Las distintas apariencias que puede tomar una interfaz depende del contenido de la base de conocimiento -formada por los tres patrones antes mencionados-, que va variando con las sucesivas interacciones del usuario con las aplicaciones :

- Si es la primera vez que se logonea al sistema, pueden ocurrir dos cosas :
  - no existe modelos para su tipo de usuario, por lo que la apariencia de la UI, será aquella que definió el diseñador de la interfaz. -modelo genérico para cualquier tipo de usuario-.
  - existen los modelos para su tipo de usuario, entonces la apariencia de la interfaz será aquella que definió el Diseñador de Modelos de Usuarios, para su tipo de usuario. -modelo genérico para un tipo de usuario particular-.

- Si no es la primera vez que se logonea al sistema -ya interactuó con alguna aplicación anteriormente-, pueden darse dos situaciones :
  - Que ingrese a una aplicación por primera vez : tomará el modelo del usuario inferido por su interacción con otros usuarios. -modelo genérico de un usuario particular-.
  - Que ya haya interactuado con tal aplicación : tomará el modelo inferido para esta aplicación en particular. -modelo particular de un usuario para una aplicación determinada-.

La otra clase incorporada es *Aplicaciones* que implementa los métodos que heredan todas las aplicaciones generadas por CUIMS para efectuar la adaptación de la interfaz y la ejecución automática de los hábitos del usuario. Esta clase es la que permite a todas las ventanas cambiar del *Nivel Presentacional* al *Nivel Funcional*, abrir los editores especiales para cada OI, asi como tambien disparar la ejecución de los hábitos, que detallamos debajo, automáticamente.

Otro aspecto a modelar serían los hábitos del usuario. Consideramos hábito a una secuencia ordenada de 5 eventos de la aplicación que se detectan en el comportamiento del usuario repetidas veces en las sucesivas sesiones. Dentro del modelo de usuario, para cada una de las aplicaciones del dominio, se mantienen las últimas sesiones, y de ellas se infieren los patrones de conducta, eliminando ambigüedades. Veremos como un hábito detectado se convierte en activo para el usuario para las sucesivas sesiones.

Sean H1 y H2 hábitos detectados durante la interacción y $Ev_{I,J}$ eventos de la aplicación, tal que :
H1 = (D1, R1) , H2 = (D2, R2), tal que :
D1=($Ev_{1,1}$; $Ev_{1,2}$) y R1=($Ev_{1,3}$; $Ev_{1,4}$; $Ev_{1,5}$).
D2=($Ev_{2,1}$; $Ev_{2,2}$) y R2=($Ev_{2,3}$; $Ev_{2,4}$; $Ev_{2,5}$).
Si D1=D2 y R1<>R2 => H1 ambiguo con H2, por los tanto ambos hábitos se rechazan.
Si D1<>D2 y R1<>R2 => H1 y H2 pasan a conformar patrón de hábitos del usuario.

Vimos que los hábitos pueden tener dos estados: *activo o pasivo.* Cuando se detecta un hábito, esta en estado activo, por lo tanto cuando se detecte alguna secuencia Di se disparará automáticamente la subsecuencia Ri. Pero en cualquier momento, el usuario puede consultar cuales son los hábitos que el sistema detectó y en caso que crea conveniente cambiarle el estado a pasivo, esto implica una desactivación temporaria del mismo. (Ver **Figura Ejemplo III**)

4. Características de las Interfaces Generadas con CUIMS

Las interfaces de usuario obtenidas con CUIMS, reúnen ciertas características deseables a saber :

Bimodal : En una interfaz generada con CUIMS, un usuario final podría operar con los objetos de la misma en dos modos :

- *Funcional* : los efectos producidos por tal interacción tendrán consecuencias directamente sobre la aplicación.
- *Presentacional* : los efectos producidos en este modo realizarán un cambio automático en la apariencia de los objetos.

Con ambos modos el usuario alimenta las bases de conocimiento, a nivel funcional servirá para detectar hábitos en la interacción del usuario y a nivel presentacional, para determinar cambios de gustos y preferencias.

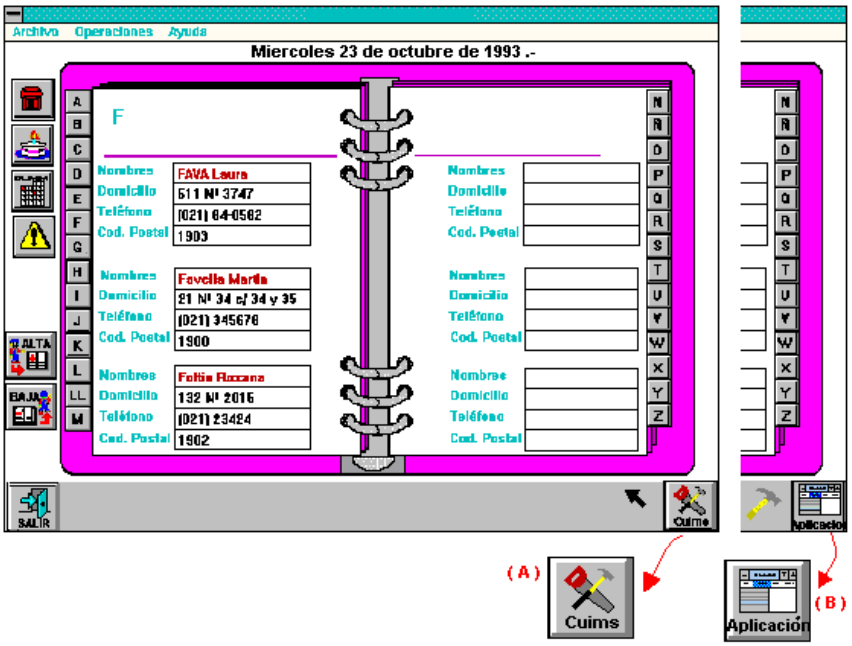


Figura ejemplo I :Este es un ejemplo de una 'Agenda Personal' generada con CUIMS en los 2 modos de operación. (A) La aplicación esta actuando a nivel funcional, por lo tanto los botones responden a sus eventos sobre la aplicación como agregar/eliminar personas de la agenda, consultar datos importantes (Hospitales, Policías, Aeropuertos, etc.) o eventos personales (Reuniones, Aniversarios, etc.), si se clickea sobre estas herramientas pasa al modo Presentacional o de diseño (B) en donde el cursor cambia su forma por la de un martillo y los efectos afectaran la presentación de la Interfaz.

Todas las interfaces generadas con esta herramienta tendrán un botón ubicado en la esquina inferior derecha que le permitirá al usuario intercambiar de modo. Para ir al modo presentacional, debe clickearse el botón de herramientas, y para ir al modo funcional debe clickearse el botón que dice Aplicación.

En el modo presentacional, el click sobre un objeto, ocasionará la apertura de un editor, para peticionar cambios en los atributos del objeto corrientemente seleccionado pudiendo tener este cambio un alcance local o global. Un cambio local afectará solamente al objeto seleccionado mientras que un cambio global, afectará a todos los objetos del tipo seleccionado. Esto es, si hay 5 check box en la interface, y fue seleccionado uno de ellos, los 4 restantes, también serán seleccionados. Además, en este nivel clickeando el botón derecho del mouse se podrán visualizar los hábitos detectados por el sistema, pudiendo también cambiar el estado de los mismos. **(Ver Figura ejemplo III)**

Adaptativas : Cuando el sistema logra tener un patrón confiable para un usuario, todas las características del mismo se verán reflejadas en las interfaces automáticamente. Un patrón confiable, significa que la cantidad y calidad de pedidos efectuados por el usuario para un objeto determinado, ha alcanzado la cantidad definida por el diseñador de modelo de usuarios, para que se efectúe la adaptación.

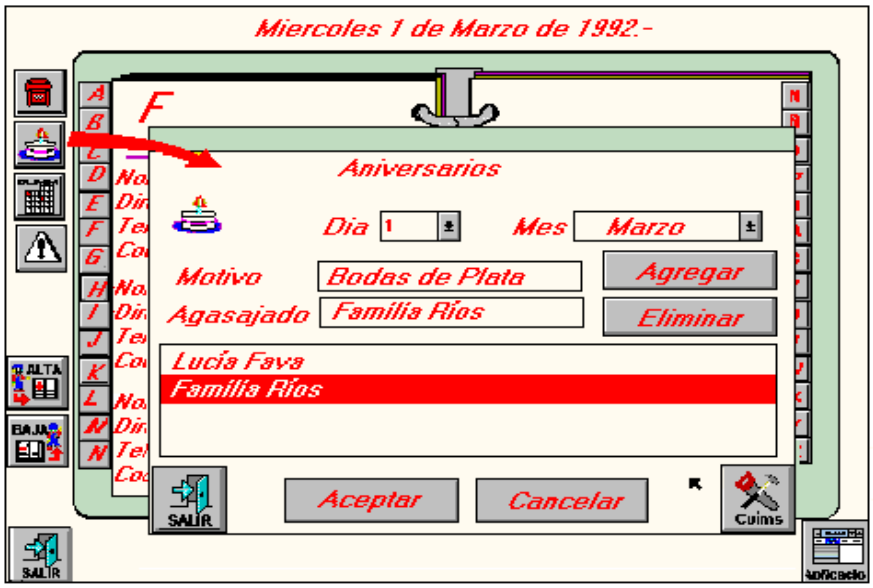


Figura ejemplo II : En el mismo ejemplo de una 'Agenda Personal' si abrimos cualquiera de las Ventanas de un usuario particular tendrá la misma apariencia en cuanto a colores, fonts, bitmaps -en este caso la interfaz ha detectado que sus preferencias son para el fondo el color beige, para las letras el color rojo, un font especial, etc.-.

Uniformes : Todas las interfaces de un usuario, generalmente tendrán el mismo aspecto, salvo que éste explícitamente, solicite lo contrario, ya que el sistema detecta un patrón de preferencia genérico para cada usuario, que lo aplicará a todas las interfaces que se vayan incorporando a su dominio de aplicaciones. Las distintas aplicaciones generadas utilizando CUIMS, mantienen una uniformidad le permite al usuario final actuar por analogía.

Asistentes : Es sabido, que el sistema recolecta datos relevantes de las interacciones del usuario con el sistema, y de estos trata de inferir hábitos. Se desprende, que, después de un número de sesiones estudiadas, un usuario tendrá ciertos hábitos para cada aplicación, por ello, cuando se detecta una sucesión de eventos que forman parte de un hábito, se lo asiste finalizando esas tareas automáticamente.

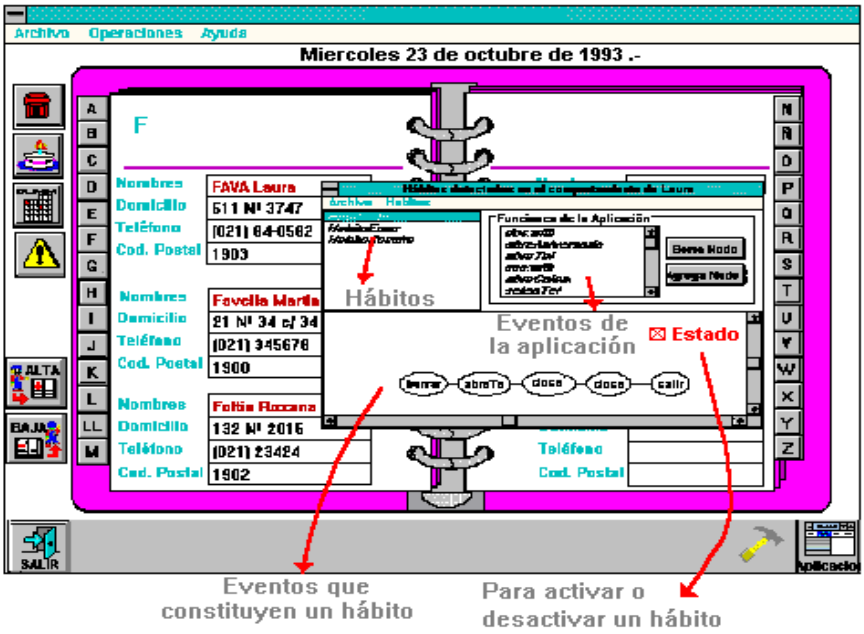
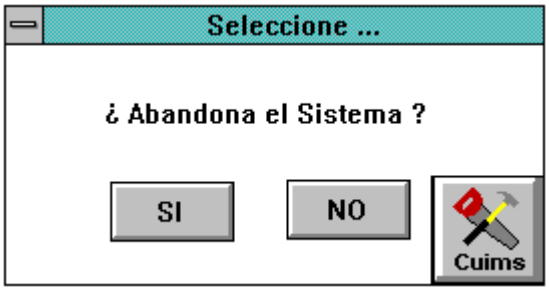


Figura Ejemplo III : Esta pantalla muestra la ventana de hábitos inferidos que el usuario final puede consultar y modificar si cree conveniente

5. Código generado para la Construcción de una pequeña ventana de control.

La Construcción de esta ventana muy simple, que no demandó mas que unos segundos nos servirá para estudiar el código generado automáticamente por CUIMS :



Desde el ambiente para construir la UI, el Diseñador puede definir cada uno de los Objetos de Interacción, su disposición sobre la ventana y su asociación con la aplicación -nombres de los eventos a que responderán cada uno de los Objetos de Interacción-. CUIMS genera automáticamente el código de la UIA, mas precisamente genera 3 métodos indispensables open, creaEstructura, close indispensables para la ejecución y métodos secundarios que corresponden al comportamiento que tendrán los distintos Objetos de Interacción de la interfaz.

Comencemos con los tres métodos primncipales generados por CUIMS :

open

```
| v alto rect cuim fram |

  cuim:= Bitmap fromFile:'c:\interfac\iconos\cuim1.bmp'.
  self creaEstructura.
  habito:= OrderedCollection new.
  self addView: (
     v := self topPaneClass new
       owner: self;
       labelWithoutPrefix:  'Seleccione ...';
       noSmalltalkMenuBar;
       viewName: 'mainView';
       framingBlock: ( FramingParameters new leftCorner: 615 @ 284; xC; yC; rectangle: (11 @ 274
                                                    rightBottom: 603 @ 48));
       pStyle: #(sysmenu modal titlebar);
       foreColor: (( estructura at: 'mainView' ) pane ) colorFrente ;
       backColor: (( estructura at: 'mainView' ) pane ) colorFondo ;
       when: #close perform: #close: ).
   self
      addSubpane: (
         AcetatePane new
            owner: self;
               framingBlock: (FramingParameters new leftCorner: 615 @ 284; xC; yC; rectangle: (
                                                                  11 @ 274 rightBottom: 553 @ 40)); );
            paneName:'invisible';
             when: #button1Down perform: #abreEditor: ;
             when: #button2Down perform: #abreHabito: ;
             yourself   );
      addSubpane: (
           Button new
             owner: self;
             framingBlock: (                                  w leftCorner: 142 @ 60; left: 119 r: #left; right:
                          261                                 ; bottom: 196 r: #top);
             paneName: 'bot
             when: #clicked
             contents:  (( estr                               ane ) label;
             font:  (( estructu                                font;
             foreColor:  (( es                                pane ) colorFrente;
             backColor:  (( estructura at: botonIsr ) pane ) colorFondo;
             yourself  );
           •
           •    ( otros objetos de interacción )
           •
      addSubpane: (
         DrawnButton new
            owner: self;
            framingBlock: [:box1| Rectangle new leftTop: ( ( (( (v rectangle)width ) ) - 57 ) @ ( ((  (v
            rectangle)height ) ) - 57) ) extent: ( 59 @ 57 ) ];
          startGroup;
          paneName:'cuims';
          when: #clicked perform: #cambiaNivel: ;
          contents: cuim;
          yourself).
     cuims := false.
      self openWindow.
      (self paneNamed:'invisible') disable.
```

Este pane habilita en el modo presentacional

Para cada uno de los atributos del IO, toma del modelo de usuario los valores correpondientes.

Este pane es el botón que permite cambiar del Nivel Funcional al Presentacional

El método 'close:' se codifica automáticamente y debe engancharse al evento que cierra la ventana, porque la interfaz necesita guardar información relevante de toda la sesión .

close: aPane

```
"Método Generado por CUIMS,cierra la aplicacion y actualiza las bases de conocimiento del usr. "
 | tipo tipoUsr sesiones |

 ( Login isNil )  ————— Login tiene el nombre del Usuarios logoneado, ni es nil
 ifTrue: [ ^super close ].   la Interfaz de Usuario está siendo testeada por el Diseñdor.

"Para el patrón de Preferencias. "
tipo := ( Usrs at: Login).
(tipo aplicaciones) at: (self class) put: estructura.
"Para el patrón de Hábitos. "
(habito size) >= 5                    Si el Usuario ejecutó más de 5 eventos
      ifTrue:[ sesiones := ( tipo phab ) agregaSesion: habito en: (self class) ].   uardamos ese comportamiento.
Usrs at: Login put: tipo.
^super close.
```

Este método decide que valores desplegar para cada uno de los OI de la interfaz como se explicó en el punto 3.

```
creaEstructura
"Método Generado por CUIMS"
 | tipo tipoUsr pref a b c unPane unFont tipoU modelo colorfr colorfo clas patp |

(Login isNil) ifTrue: [ ^ self creaEstructura1 ].  ————— Corre una Demo con valores definidos por el
                                                          Diseñador de la UI.

 tipo := ( Usrs at: Login).
 pref := ( tipo aplicaciones ) at: ( self class).
 patp := ( tipo ppref ) atributos.                  —— Toma los valores del Modelo de Usuario
 tipoU := tipo tipoUsr.                                  para aplicarselos a los OI. Si no encuentra
 modelo := (( Patr at: tipoU ) ppref ) atributos .       valores es porque es la primera vez que
 ( pref notEmpty )                                       interactúa, sigue en *
     ifTrue: [ estructura := Dictionary new.
               pref do: [ :elem |  c:= Association new key: (( elem pane ) nombre );
                                                  value: ( elem llenaValores ).
                                   estructura add: c
                         ].
          ^ estructura ].

estructura := Dictionary new.  * Crea una nueva estructura para este usuario.
StaticT clase  notNil                                   si existen valores para el modelo de
   ifTrue:[   ((patp at: clas) font) notNil             usuario al que el pertenece toma esos
                    ifTrue:[ unFont:= (pat              valores. Patrón del estereotipo al que
                    ifFalse:[ (( modelo at:              pertenece este usuario.
                                 unFont:                    nt ]].
                              Frente ) n
                              fr:= ( pa                     e ]
toma los val                  elo at: cla                   il
definidos                     colorfr:=                     orFrente ]].
por el          ((                  ndo ) not
Diseñador                           o:= (patp at: clas) colorFondo ]
de la UI,                      ifFalse:[ ((modelo at: clas) colorFondo ) notNil
Patrón genérico.          ifTrue:[ colorfo:= ( modelo at: clas) colorFondo ]].

 a := Association new key:'mensaje'
 unPane := StaticT new  ——— crea el IO -en este caso un texto estático-
                nombrePane: 'mensaje';
                label: '¿ Abandona el Sistema ?';
                colorFrente: colorfr;
```

```
                colorFondo: colorfo;
                tamano:17 @ 189;
                posicion:62 @ 58;
                font: unFont.
 b := ViewUser new
                pane: unPane ;
                colorFondo:  ( Atributo new inicializa: ( unPane colorFondo ) );
                colorFrente:  ( Atributo new inicializa: ( unPane colorFrente) );
                etiqueta: ( Atributo new inicializa: ( unPane label ) );
                font: ( Atributo new inicializa: ( unPane font ) );
                estilo: ( Atributo new inicializa: ( unPane estilo ) ).
 a value: b.
 estructura add: a.   ——— inserta el IO  a la estructura del usuario.
                •
                •    ( idem para cada uno de los subPanes y Ventana Principal )
                •
^estructura.
```

Ahora vemaos que código genera para los métodos asociados a un evento. Supongamos que el usuario setea para el botón con Label -SI- un evento llamado '*Si:*', el gerenciador automáticamente produce un código como el que sigue :

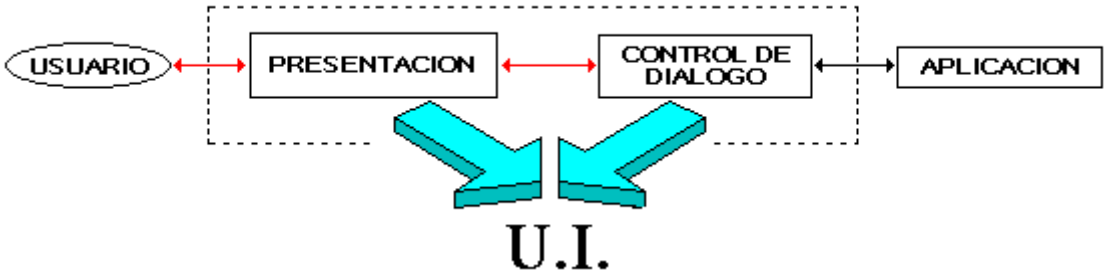
```
Si: aPane
 | metodo |
" A partir de aquí codifique su método  "
     ———      Acá el Programador de la Aplicación debe completar el código de la aplicación. "

Si Borra esta Validación se perderá la Customización"
cuims ifFalse:[ (( aPane handlers ) includesKey: #clicked )
                ifTrue:[ metodo := (( aPane handlers) at: #clicked ).
                        ( metodo = ( 'Si:' asSymbol ) )
                          ifTrue:[ self disparaHabito: 'Si:'].
                          ].                    método permite ejecutar
                                                automáticamente un hábito del usuario.
```

Habrá tantos encabezados como cantidad de eventos a los que respondan los objetos, el Programador de la aplicación se limitará a completar el código de la aplicación, como indica el ejemplo anterior.

6. Conclusión

Hay dos objetivos que guiaron el desarrollo de esta UIMS y se han visto consumadas; una fue el desarrollo de una herramienta que separe el código de la Interfaz de Usuario del de la aplicación y la otra fue buscar un diseño centrado en el usuario, esto es poder modelarlo para adaptar consecuentemente la interfaz a sus necesidades.

En cuanto a la primera, es decir *Separación de la Interfaz de Usuario del código subyacente,* CUIMS enfatiza la separación de la Interfaz del Usuario del corazón funcional, pudiendo evidenciarse lo que antes veíamos en el conocido modelo interactivo :



Separar las componentes de un sistema en Aplicación, Control de Diálogo y Presentación acelera el desarrollo y facilita su posterior mantenimiento.

El otro punto fue *Modelar al usuario,* el usuario puede modelarse en varios aspectos, el diseñador de Modelo de Usuarios debe analizar a los potenciales usuarios del sistema adaptativo que se está desarrollando y agruparlos por características. El Diseñador de MU, puede definir aspectos presentacionales que crea conveniente para cada tipo de usuario, así como también, los mensajes y helps que se le desplegará a cada modelo.

Durante dos ciclos lectivos la herramienta fue utilizada para el desarrollo de distintos sistemas adaptativos en la Cátedra de Prototipación e Interfaces de Usuarios, y les brindó a los alumnos la posibilidad de comprobar tópicos teóricos como :

- Cada componente de un sistema puede ser desarrollada independientemente. El desarrollo en paralelo es por lo general más rápido que el desarrollo lineal.
- El desarrollo de cada una de las partes puede y es conveniente que sea hecho por personas distintas, más precisamente por expertos en el tema. Se recomienda que los Modelo de Usuarios sean definidos por personas con conocimientos en factores humanos, la interfaz de usuario es conveniente que la desarrollen expertos en tecnologías junto con diseñadores gráficos y la aplicación expertos en desarrollo de sistemas.
- Los cambios en una componente, generalmente no afectan a las otras componentes. La interfaz puede cambiar y la aplicación puede quedar intacta, esto reduce el costo del mantenimiento.

Estos puntos serían características de una UIMS tradicional, con CUIMS además pudieron evidenciar otras características como :

- El código de la Interfaz de Usuario Adaptativa es generado automáticamente por la herramienta. Todas las clases generadas con CUIMS se crean como subclases de *Aplicaciones Adadptativas* y heredan todas las variables de instancias y el comportamiento necesarios para la adaptación.
- Las interacciones del usuario permiten detectar hábitos para luego ejecutarlos automáticamente, evitándole al usuario que desarrolle repetidas veces la misma secuencia de acciones.

- El código de la Aplicación no crece en magnitud con los usuarios, ya que este es completamente parametrizado por lo que su complejidad o tamaño no depende de la cantidad de usuarios sino de la aplicación en sí misma.
- Los helps y mensajes tendrán apariencia y contenido de acuerdo al nivel de usuario.

7. Referencias

[1] Dianne Murray, "Building a User Modelling Shell".

[2] Krasner and Poper. "A Cookbook for using the Model-View-Controller User Interface paradigm in Smalltalk 80".