Investigación y Desarrollo Nº 1 volumen 7 Julio - Diciembre 2014 ISSN 1390-5546 UTA, Ecuador

CREACIÓN DE SOFTWARE INTERACTIVO A PARTIR DE COMPONENTES

Tony Flores[1]

[1]Escuela Superior Politécnica de Chimborazo

Instituto de Investigaciones, Facultad de Informática y Electrónica

RESUMEN

La creación de software interactivo a partir de componentes es una aproximación de actualidad y bien dominada por los software no interactivos. Para los software interactivos la dimensión IHM (Interfaz hombre-maquina) posee problemas específicos, precisamente a causa de la necesidad de poder evolucionar la interfaz a la demanda de los usuarios y en el momento de la re-composición del software. En este trabajo se indican los conceptos, las características y las aproximaciones actualmente utilizadas para la creación de software interactivo. El termino componente es utilizado en diferentes marcos de la ingeniería como piezas (elementos) para la creación de un objeto o de un dispositivo especifico, también esta idea es llevada a la informática para la creación de software. Para la creación de software se han identificado tres tipos de manipulación de componentes: integración, extensión y personalización; que son expuestos en este trabajo.

Palabras clave: Paradigmas de programación, Análisis de tareas, Reingeniería de software

ABSTRACT

The software creation with components is an approach of present time for the non-interactive software. In interactive software, the HCI has specific problems because the interfaces don't satisfy the necessities of the users. The term "component" is used in engineering structures for the production of devices. This idea is transported to the computer science to create software. This work presents the concepts, the characteristics and current approaches for the creation of interactive software. Three types of manipulation of components to create software have been identified: integration, extension and customization.

Keywords: Programming Pattern, Task Analysis, Software Reengineering.

INTRODUCCIÓN

La programación por componentes apareció por la necesidad de crear software de una manera más simple. Hace varios años diferentes métodos o “paradigmas” de la programación se han desarrollado y son hoy en día muy utilizados, (ej.: la programación secuencial o la programación orientada a objetos). Entre sus contribuciones se encuentran el mejoramiento de las interfaces gráficas de usuario que pretenden una mejor comunicación entre los humanos y las maquinas.

La interacción hombre – máquina se ha perfeccionado y toma cada vez más tiempo programar. La preocupación de crear programas de mejor calidad esta igualmente presente. Entonces fue importante clasificar los programas en dos categorías: programas interactivos y programas no interactivos.

Los componentes de software aparecen como una posible respuesta a la exigencia de rapidez y de calidad de desarrollo. Construir software gracias al ensamblaje de partes de código existentes constituye una aproximación interesante. Desde el punto de vista de ingeniería de software, el término “componente” se encuentra en la influencia de las “técnicas orientadas a objetos” como una extensión natural de esta. De las aproximaciones de descomposición utilizadas en “técnicas de descomposición de problemas” o “análisis de tareas” y el deseo de desarrollar los sistemas abiertos son en razón los que dan el sentido de reutilización [9].

La idea de esta concepción es la creación de programas con componentes ya producidos (desarrollados), es decir, evitar el uso de los paradigmas de programación comunes. El uso de componentes es ampliamente aplicado en la actualidad como la mecánica (ej.: piezas de un automóvil) o la electrónica (ej.: circuitos integrados) siendo esta la idea que genera el concepto de la programación por componentes al existir una analogía entre un programa y un circuito que realizan una tarea específica. Para la creación de los diferentes componentes se usara las técnicas de análisis de tareas ya que como se sabe una tarea puede dividirse en sub-tareas, cuando una tarea es mas pequeña es mas fácil de realizar.

Este documento está organizado como sigue: en el estado del arte se revisan algunos conceptos sobre paradigmas de programación, el análisis de tareas y las características del software interactivo. Luego se estudian los conceptos relacionados a la noción de componente y las características de los componentes generales y comerciales. Después se explican las aproximaciones utilizadas para la creación de software a partir de componentes. Para luego presentar las ventajas y desventajas de utilizar componentes en la creación de software interactivo. Finalmente, se exponen las conclusiones así como las pistas de trabajo a continuar.

ESTADO DEL ARTE

Los Paradigmas de la Programación

Los paradigmas de programación son las aproximaciones, las formas de crear aplicaciones, herramientas, sistemas operativos, etc. Cada uno de estos paradigmas presenta ventajas e inconvenientes y por esta razón no se puede decir que uno sea mejor que otro.

Los lenguajes de programación pueden ser clasificados según el paradigma que ellos utilizan. Por ejemplo: lenguajes como Pascal o C utilizan el paradigma imperativo, Fig. 1, lenguajes como Smalltalk y Java utilizan el paradigma orientado a objetos, Fig. 2, o; el lenguaje Scheme que utiliza el paradigma funcional.

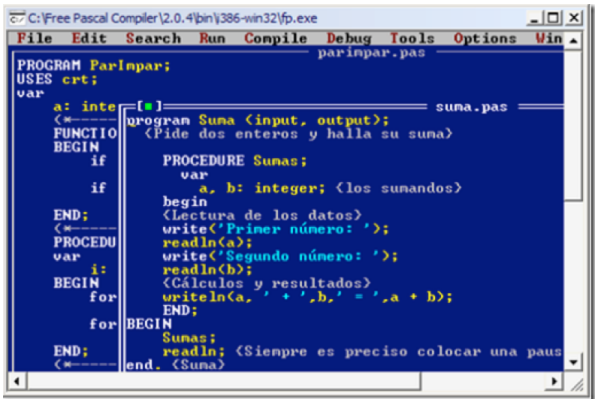

Figura 1. Programación secuencial

Los principales paradigmas de programación son: Imperativo, Declarativo, Funcional, Estructurado, Orientado a Objetos, y, Lógico (“alegsaonline.com,” 2006).

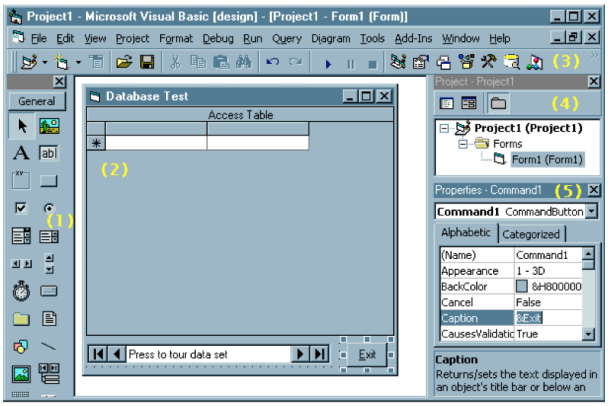

Figura 2. Programación Orientada a Objetos, (1) Barra de objetos, (2) Área de trabajo, (3) Barra de herramientas, (4) Ubicación del proyecto y, (5) propiedades del objeto seleccionado

Existen otros paradigmas como: la programación orientada a aspectos, la programación dirigida por eventos, la programación por capas, la programación basada en prototipos, etc. [23].

Análisis de Tareas

El análisis de tareas consiste en identificar todas las acciones que constituyen la tarea y a organizar el orden de sus acciones. La división del trabajo en sub-tareas ha sido ideado en el siglo XVIII pero no fue hasta los años 30 del siglo XX que este fue perfeccionado e implementado, el ejemplo más notorio fue la construcción de un automóvil, dividiendo el trabajo en pequeñas

sub-tareas que fueron entregadas a cada obrero y cuyo objetivo era ganar en tiempo y eficiencia por acumulación de experiencia y especialización.

El objetivo del análisis de tareas es determinar y dividir un trabajo en sub-tareas más simples para los diferentes actores (programadores, ergónomos y usuarios finales) que deben trabajar en conjunto para analizar, concebir e implementar de una manera correcta este tipo de realizaciones. El análisis de tareas permite identificar también la duración, la frecuencia, la importancia, la dificultad, las relaciones, las decisiones a tomar, la información necesaria y generada y los criterios de ejecución [6], La documentación, la observación y la entrevista son maneras de reunir la información sobre las tareas realizadas por un usuario [5]. Existen otras maneras como por ejemplo, la utilización de cuestionarios o registros de video del trabajador realizando su tarea diaria.

Existe una gran variedad de métodos para el análisis de tareas entre los más importantes: Goal Operator Method Selection Rules – Objetivo Operador Método Selección Reglas (GOMS), Méthode Analytique de Description des Tâches – Método Analítico de Descripción de Tareas (MAD), Task Knowledge Support – Soporte de Conocimiento de Tareas (TKS), Concur Task Trees – Arbol de Trabajo Concurrente (CTT), Groupware Task Analysis – Software Colaborativo para el Análisis de Tareas (GTA), etc. Siendo los más usados el análisis jerárquico de tareas, GOMS y MAD.

El análisis de tareas es parte de esta investigación, pues será el método a usar para determinar las sub-tareas de un trabajo y que se convertirán en los componentes de un programa más complejo. En la aproximación descrita en Componentes Orientados a Tareas o COTs esta se apoya sobre una aproximación híbrida compuesta de CTTE (Mori, Paternò y Santoro, 2002) y MAD llamada STOrM (Bourguin, Lewandowski y Tarby, 2007) mientras que en la aproximación Reingeniería de interfaces de software por extensión se utiliza el análisis jerárquico de tareas, Fig. 3 (Wiklik, 2004).

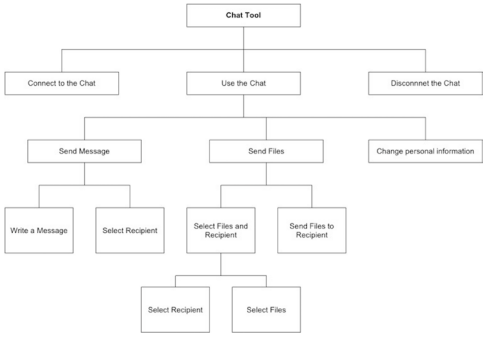

Figura 3. Análisis Jerárquico de Tareas

Software Interactivo

Según Berenguer (s.f), el software interactivo debe permitir una navegación apropiada, parcial o total. Se trata de estimular la interacción para no aburrir al usuario, de abastecerlo de toda información solicitada y sobre todo satisfacer de una manera sostenida su interés con la utilización de recursos audiovisuales abundantes y atrayentes. Berenguer propone que las características de los contenidos del software interactivo sean elaboradas en función de tres variables, Fig. 4:

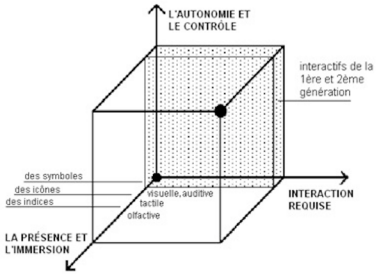

Figura 4. Plano en 3D de características de los programas interactivos

La primera concierne a la cantidad y la complejidad de interacción del software (simple como un libro electrónico o complejo como un juego de video). La segunda concierne a la capacidad de control o autonomía dados al usuario (restringidas como la enseñanza programada o control extendido como en las enciclopedias). La tercera concierne a la "presencia" e implicación personal del usuario con el software (llamado a los diferentes sentidos y partes del cuerpo humano).

CONCEPTOS

Antes de comenzar la creación de software con la ayuda de componentes, es necesario conocer los conceptos elementales para su desarrollo. Primero, es importante recordar las características de la creación de software mediante programación orientada a objetos. La programación orientada a objetos conduce al desarrollo de la aplicación basado en la colocación de un modelo mental apoyándose sobre objetos reales o imaginarios, mientras que el desarrollo a partir de componentes estipula que la aplicación debe ser desarrollada a partir de componentes prefabricados (Rojas y Molina, 2004).

Componentes

Es importante considerar que las características de un componente de software no son similares con las de clase, módulo, paquete, librería, subsistema, recurso, framework u objeto por este motivo es necesario determinar un concepto para este. Al no existir un acuerdo entre los actores de la programación para determinar una definición única de componente (Fuentes, Troya y Vallecillo, 2000) existen varias entre las cuales tenemos las siguientes:

• Una unidad de composición con las interfaces de modo contractual especificadas y que aclaran las dependencias al contexto de utilización. Un componente de software puede ser desarrollado independientemente y puede ser integrado a través del tercer elemento para obtener su composición (Szyperski y Pfister, 1997).

• Un conjunto de componentes "atómicos" desarrollados simultáneamente. Un componente atómico es un "modulo" más un conjunto de "recursos". Un modulo es un conjunto de clases y puede ser una construcción no orientada a objetos, tales como procedimientos y funciones. Un recurso es una colección de artículos escritos (para personalizar el componente) (Szyperski, 1998).

• Los "siete criterios" de definición de un componente (Meyer, 1999):

1. Puede ser utilizado por las otras partes del software
2. Puede ser utilizado por los clientes sin la intervención de desarrolladores de componentes
3. Incluyen una especificación de todas las dependencias
4. Incluyen una especificación de funcionalidades que el ofrece
5. Es utilizable sobre la base sola de sus especificaciones
6. Es compatible con los otros componentes
7. Puede ser integrado en un sistema rápidamente y fácilmente

• Un modulo lógicamente coherente, débilmente acoplado, que concierne a una sola abstracción (concepto de Grady Booch).

• Prefabricado, pre testeado, autónomo, modulo re-utilizable; un paquete de datos y procedimientos que ejecutan funciones especificas (concepto de Mettez Group).

• Una abstracción estática con conectores (Nierstrasz, 1995).

• Un pedazo de software suficientemente pequeño para ser creado y mantenido, y suficientemente grande para ser desarrollado y soportado, y con las interfaces estándar para la interoperatibilidad (concepto de Jed Harris también adoptado por Orfali).

• Cualquier cosa que pueda ser compuesta (Weck, Bosch y Szyperski, 1997).

• Autónomo, parte claramente identificable que describe y/o ejecuta funciones especificas, con una interfaz clara, una documentación apropiada, y condiciones de reutilización definidas (concepto de James Sametinger).

• Es una unidad binaria de sistema independiente que ejecuta una o más interfaces (concepto de James Sametinger).

• Una unidad binaria que exporta e importa usando funcionalmente un mecanismo de interfaz normalizado (Broy et al., 1998).

• Una parte de un equipamiento, que define un conjunto común de protocolos entre sus miembros. Un pedazo genérico de software con las interfaces robustas y bien precisadas (Henderson-Sellers, Pradhan, Szyperski, Taivalsaari y Wills, 1999).

• Una parte física, reemplazable de un sistema que empaqueta la aplicación y provee la realización de un conjunto de interfaces. Es una meta subclase de clasificador (especificación OMG UML 1.3)

Otros conceptos de la Programación Orientada a Componentes (Programming Oriented Components - POC)

Además del concepto de componente de software, existe otro conjunto de conceptos que se deben considerar en POC para poder diferenciales de otros paradigmas de programación, estos son: composición tardía, ambiente, evento, reutilización, contrato, polimorfismo, seguridad y reflexibilidad (Fuentes et al., 2000).

Características de los componentes

La principal característica de los componentes es que son reutilizables (Montilva, Arape y Colmenares, 2003), y además deben ser muy cercanos a las propiedades de los componentes materiales (hardware).

Las siguientes son características que los componentes de software deben satisfacer: Identificación, Accesibilidad solo a través de su interfaz, Servicios invariables, Documentación, Genérico, Autónomo, Entretenido, Independiente de la plataforma, del lenguaje de programación y de las herramientas de desarrollo, Reutilización dinámica, Certificación y Acceso uniforme cualquiera que sea su localización. En la sociedad de desarrollo de software la utilización de códigos de programa bajo la forma de componentes ha permitido reducir el tiempo de desarrollo, mejorar la fiabilidad de los productos y por ende la competitividad en precios (Bertoa, Troya y Vallecillo, 2003; "scruz334.blogspot.es," s.f). Por esta razón las empresas de desarrollo de software han optado por comercializar componentes llamados Commercial Off The Shelf (COTS) (Bertoa et al., 2003), donde las características son las siguientes:

- Son vendidos a los poseedores de una licencia o al público en general.
- El vendedor los mantiene y los actualiza, es él quien conserva los derechos de propiedad intelectual.
- Son disponibles bajo la forma de múltiples copias, todas idénticas entre ellas.
- Su código fuente no pueden ser modificado por los usuarios.

APROXIMACIONES

Compartiendo la idea del trabajo de Morch et al., (2004) en la parte que concierne a la "maleabilidad" ("tailorability" en Ingles) se identifica que la manipulación de componentes puede ser estructurado en tres niveles: la integración (cuando se toma una lista de componentes y se los ensambla hasta formar el programa deseado), la extensión (cuando a un programa existente se le adjunta nuevos componentes que le hace evolucionar) y la personalización (adaptar o editar los componentes por la elección de opciones fácilmente manipulables bajo la forma de parámetros).

De estos tres niveles o tipos, el más empleado es la extensión mediante la instalación en la aplicación de componentes denominados PlugIns con la finalidad de adjuntar nuevas funcionalidades al programa.
La personalización es utilizada principalmente para mejorar la presentación de una aplicación por la modificación de características de los objetos que forman el componente (cajas de texto, botones, botones de radio, etc.)
La integración es una nueva forma de crear programas, que se inspira de la aproximación utilizada en mecánica o electrónica para el ensamblaje de dispositivos con la ayuda de componentes que realizan un trabajo específico.
A continuación, veremos dos aproximaciones que utilizan el análisis de tareas como base fundamental para la creación de nuevos componentes apoyándose en diferentes tipos de implementaciones.

Componentes Orientados a Tareas o COTs

La ventaja de utilizar esta aproximación es que el modelo de tareas está presente en todos los ciclos de vida del componente, durante su concepción, desarrollo, integración y evaluación. En razón de la dificultad de unir los componentes existentes a partir de la perspectiva de componentes ya existentes (aproximación botton-up) se ha decidido utilizar una aproximación top-down para construir y desarrollar los COT (Lewandowski, Bourguin, Tarby, 2007). La herramienta utilizada para la creación del modelo de tareas se denomina STOrM (Simple Task Oriented Modeler). Esta herramienta ha sido implementada como un PlugIn de Eclipse y para representar el modelo de tareas usa una combinación de CTTE y K-Made. Este trabajo ha permitido identificar los problemas que existen con la tecnología a base de componentes y de su integración, Fig. 5.

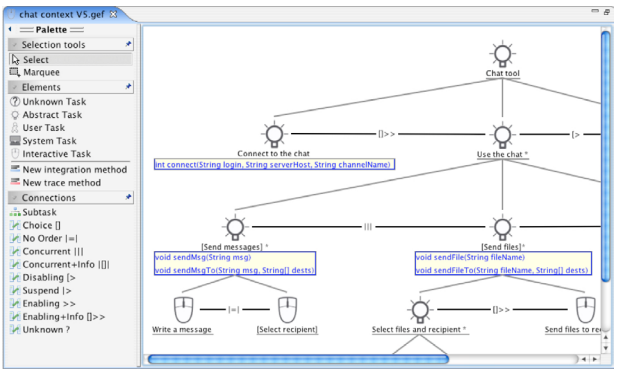

Figura 5. Modelo de tareas descrito por el desarrollador

Etapas de la aproximación. La proposición de la creación de componentes es definido en varias etapas donde las cinco más importantes para el proceso son las siguientes:

1. Fase de movilización de tareas, afín de describir el comportamiento esperado del futuro componente. Es el modelo de tareas que será el punto de partida de la creación de un COT. El trabajo colaborativo facilita la creación de componentes entre los desarrolladores y los usuarios finales (la construcción de un árbol de tareas), cada tarea debe ser modelada y esta debe ser bien comprendida por los actores.
2. El modelo de tareas es entendido por el programador especificando los métodos que serán implementados junto con las tareas y comentarios que serán insertados en el código en enlace directo con las tareas.
3. Los esqueletos de implementación y de documentación de los componentes pueden ser directamente deducidos (o generados de manera automática) del modelo de tareas extendido por los actores en las etapas descritos aquí antes.
4. Los esqueletos son implementados usando la aproximación orientado a objetos, para respetar las especificaciones del modelo de tareas extendido. Fig. 6.
5. Los componentes son entregados con la documentación habitual en una aproximación clásica orientada a objetos, pero también con su modelo híbrido atando las tareas del usuario y el código del componente (formando la documentación de un TOC). Este modelo se presenta bajo la forma de un fichero XML describiendo el árbol de tareas del componente, así como sus enlaces con su código funcional.

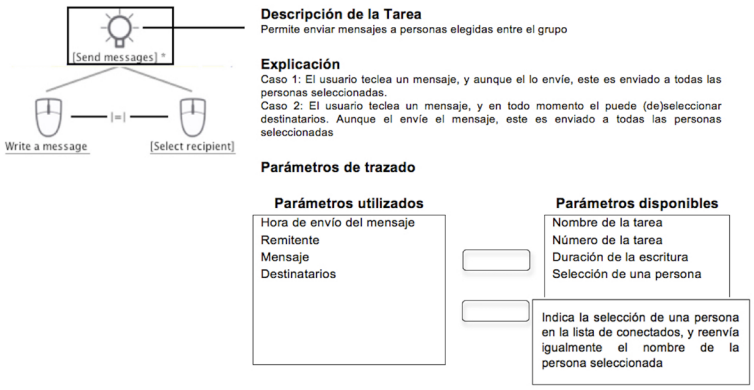

Figura 6. Perspectiva de la Interface Hombre-Máquina

Reingeniería de interfaces de software por extensión

El fin del trabajo de A. Wiklik (2004) era concebir una nueva interfaz para el software GIMP que respete la mayoría de los criterios de facilidad de utilización y que este orientado a tareas, esto debido a las interfaces complejas que el software presentaba, la participación activa de los usuarios es importante para su desarrollo. GNU Image Manipulation Program (GIMP) es una aplicación de dominio público creada bajo licencia de GNU para el tratamiento y manipulación de imágenes. Es un programa modular y las nuevas funcionalidades se le adjuntan como PlugIns al núcleo. Funciona en Linux, Windows y MacOS.

El método utilizado en este trabajo es el análisis jerárquico de tareas. Una interfaz orientada a tareas permite una mejor utilización del programa por el crecimiento de la eficacia, la eficiencia y la satisfacción de los usuarios, Fig. 7.

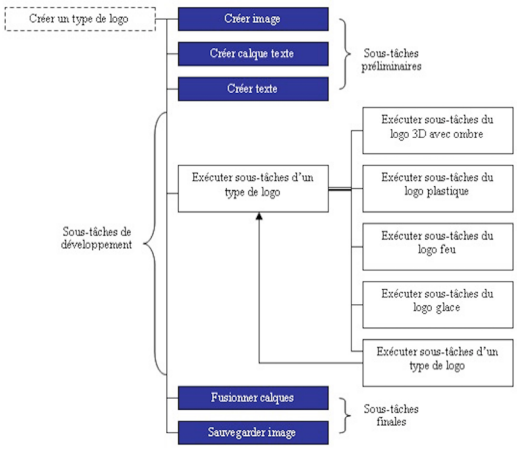

Figura 7. Agrupamiento de Sub-Tareas

Automatización de las tareas. Para evitar repetir las mismas tareas y modificar los parámetros varias veces, A. Wiklik ha elaborado una herramienta integrada a GIMP, Script-Fu que automatiza las tareas (definición de los parámetros de entrada estáticos y dinámicos). No es necesario ir al código fuente para aportar los cambios. Los PlugIns permiten adjuntar las nuevas

funcionalidades a GIMP. Estos son adjuntados al núcleo central del programa y son registrados en la base de datos sumarial de GIMP.

Las extensiones del programa pueden estar puestas a punto por los usuarios, pero los usuarios deben conocer la programación en C/C++ y la librería GTK para la creación de los PlugIns. Los usuarios deben igualmente utilizar la herramienta GimpTool para instalar y desinstalar los PlugIns en la base de datos sumarial, Fig. 8.

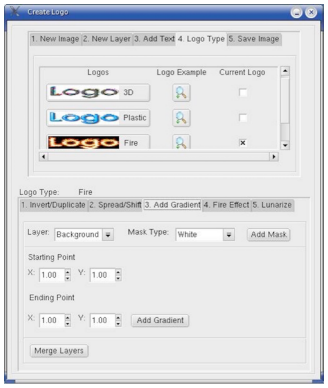

Figura 8. Presentación de la nueva interface

Descripción del proceso y de las etapas de la agregación de PlugIns en GIMP. El ciclo de desarrollo adoptado es una combinación de los ciclos en cascada y en espiral. Además, es centrado al usuario, que participa en la elaboración de la nueva interfaz (mejoramiento de la ergonomía). Este ciclo se compone de seis etapas: análisis (definición de usos, usuarios, tareas y de requisitos), concepción (elaboración de las maquetas de la interfaz), evaluación (corrección entre los programadores y los usuarios a través de sus comentarios), implementación (creación del sistema), evaluación (test de utilidad del software y de los componentes adjuntados por los usuarios) y mantenimiento (si el software ya está en servicio)

VENTAJAS Y DESVENTAJAS

A continuación se lista algunas de las ventajas y desventajas de utilizar componentes:

Ventajas

- Se emplea menos tiempo en la creación o desarrollo de programas a diferencia del paradigma imperativo y la programación orientada a objetos donde los usuarios deben tener conocimiento de estos tipos de programación.
- Desarrollo de componentes de manera independiente, principalmente con la utilización del análisis de tareas.
- Reemplazo de componentes, de la misma forma que en la electrónica donde se reemplaza los componentes que son defectuosos.
- Reutilización del código fuente, facilita la programación por la existencia de diferentes programas ya creados.
- Introspección, es más fácil determinar las propiedades y los métodos de un componente de manera dinámica.
- La ventaja de conocer el código fuente del componente nos permite una conexión dinámica.
- Aumento de la fiabilidad del producto final.
- Esta aproximación aumenta la competencia entre empresas y por consecuencia la calidad de los componentes aumenta y el precio disminuye.
- Los componentes de código abierto pueden ser de mejor calidad y gratuitos.
- Con el tiempo los usuarios podrán crear ellos mismos sus propios programas por sus propios trabajos específicos obteniendo de esta forma aplicaciones personalizadas.
- Simplifica el análisis, permite verificar separadamente los componentes antes de comprobarlos en conjunto. De esta manera no será necesario tener el programa completo para verificar su funcionamiento.
- Simplifica el mantenimiento del sistema, pues solamente verifican los componentes que no se encuentran en buen funcionamiento.

Desventajas

- En el caso de no existir componentes, se deberá continuar el desarrollo de programas por los métodos anteriores, y por consiguiente la pérdida de tiempo. Además de los conflictos generados por el desarrollo de nuevas versiones que puedan aparecer.
- Los programadores podrán crear una gran cantidad de componentes sin llegar a satisfacer las expectativas de los usuarios.
- Es difícil concebir un componente si no se conoce quienes son los usuarios y los usos; es decir, los otros componentes de la aplicación final.

- Para el desarrollo de los componentes se debe utilizar los modelos existentes y con los mecanismos que le permitirán interaccionar (Modelos CORBA, DCOM, SimpleBeans), es decir el desarrollo de programas se limita a las personas que conocen de la programación.
- Es necesario crear estándares, mecanismos y herramientas que permitan la descripción del componente y adjuntar esta información en ellos.
- Entre los fabricantes de componentes existen desacuerdos para determinar los parámetros de calidad, lo importante es que exista una institución que reglamente estos parámetros.
- Ausencia de documentación de los componentes es decir, la descripción de cada uno en diferentes aspectos como: funcionales, extra funcionales, de calidad, técnicas de distribución y empaquetado, marketing y comerciales.
- Necesidad de extender la funcionalidad de los depósitos y repertorio de componentes (una especie de guía)
- Existen diferentes problemas en el momento del desarrollo de componentes a causa del tipo de usuario y que cada uno tiene diferentes calidades, direcciones, intereses y recursos y existen principalmente problemas técnicos en el momento de ensamblar componentes (recubrimiento, lagunas, incompatibilidad)(Bertoa et al., 2003).
- Metodología del diseño (Top-Down y Botton-Up): Basados en un trabajo específico se va descendiendo hasta crear los componentes individuales (Top-Down), pero para crear la aplicación se debe utilizar (Botton-Up). El problema es que nosotros podemos crear componentes para cada tarea pero no se sabe si estos componentes creados puedan servir para crear nuevas aplicaciones más complejas.

CONCLUSIONES

La utilización de componentes en diferentes sectores del conocimiento humano ha favorecido de gran manera a la industrialización, así como es notoriamente conocido en diferentes sectores como la mecánica y la electrónica. En este caso la utilización de componentes de distintos fabricantes no ha sido obstáculo para la creación de sistemas cada vez más complejos.
La idea de utilizar el análisis de tareas para la creación de componentes pueden ayudar significativamente al usuario final puesto que es él quien puede describir de mejor manera las tareas que realiza en su trabajo y crear su propio software a partir de esto.
La ausencia de un lenguaje de modelización unificado para el análisis de tareas comprensible para todos (usuarios finales, ergónomos y programadores), impide en una cierta medida la correcta determinación de tareas y sub-tareas.
La creación de programas a partir de componentes favorece la interacción entre el hombre y la máquina principalmente porque ella permite que el desarrollo de un nuevo programa sea realizado por los usuarios finales, no dejando únicamente el trabajo a los programadores. Además esto reducirá a futuro la necesidad de aprender diferentes lenguajes de programación.
Existen diferentes problemas al momento de juntar los componentes, por el momento son identificados tres tipos, las lagunas (cuando al unir los componentes no cumplen con una exigencia), los recubrimientos (cuando al juntar los componentes ellos realizan parcialmente el mismo trabajo o ellos cumplen una misma condición requerida con características diferentes) y la incompatibilidad de interfaces (interfaces incompatibles dado que son construidas utilizando diferentes lenguajes de programación).

TRABAJO FUTURO

Los componentes pueden o deben ser desarrollados por programadores en código abierto a fin de evitar la necesidad de comprar los componentes, pero debería existir un equipo que revise los avances en la creación y la modificación de cada componente.
La creación de un depósito o de un catálogo de componentes de software, similares a los que existen para los componentes de hardware es necesario. De esta manera comenzará la verdadera ingeniería de software orientado a componentes.
Permitir asimismo que la creación de programas sea posible con la participación de los usuarios finales y no solamente los programadores que no tienen la misma visión de uso.
La solución a la falta de un lenguaje de modelización unificado para el análisis de tareas será la creación de un programa que permita el análisis de tareas de tal manera que sea un proceso simple y que todos puedan comprender, que esta nueva aplicación se adapte a todos los métodos utilizados por el análisis de tareas (CTT, MAD, GOMS, Jerárquico, etc.).

RECONOCIMIENTO

El autor aprecia el apoyo económico dado por la Secretaria Nacional de Educación Superior, Ciencia, Tecnología e Innovación (SENESCYT - Ecuador) a través de su programa de becas para estudios de Maestría y Doctorado en el extranjero, al soporte científico del Laboratorio de Informática para la Empresa y los Sistemas de Producción de la Escuela Central de Lyon (LIESP-ECL), la Universidad Claude Bernard Lyon 1 y a las personas que revisaron el trabajo gracias por sus valiosos comentarios.

REFERENCIAS BIBLIOGRÁFICAS

Aruquipa, M., Márquez, E. (2003). Desarrollo de Software Basado en Componentes. XIII Asamblea General del ISTEC. Bolivia.

Baron, M., Lucquiaud, V., Autard, D., Scapin, D.L. (2006). K-MADe: un Environnement pour le Noyau de Modèle de Description de l'Activité. Proceedings of the 18th international Conference of the Association Francophone D'interaction Homme-Machine (IHM '06), 133, 287-288.

Bertoa, M., Troya, J.M., Vallecillo, A. (2003). Capitulo 8. Aspectos de Calidad en el Desarrollo de Software Basado en Componentes. Calidad En El Desarrollo Y Mantenimiento Del Software, Ed. RA-MA, 159 – 177.

Bourguin, G., Lewandowski, A., Tarby, J.C. (2007). Defining Task Oriented Components. TAMODIA 2007. LNCS, Springer, 4849, 170 – 183.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plasil, F., Pomberger, G., Pree, W., et al. (1998). What characterizes a (Software) Component? Software–Concepts and Tools, 19. 49–56.

Dix, A., Finlay, J. E., Abowd, G. D., Beale, R. (2004). Human-Computer Interaction (3rd Edition). Prentice-Hall, Inc.

Fuentes, L., Troya, J., Vallecillo, A. (2000). Lección 1 - Desarrollo de Software Basado en Componentes. Depto. Lenguajes y Ciencias de la Computación. Universidad de Málaga, España.

Henderson-Sellers, B., Pradhan, R., Szyperski, C., Taivalsaari, A., Wills, A. C. (1999). Are Components Objects?. OOPSLA'99 Panel Discussions.

Lewandowski A, Bourguin G, Tarby J.C. (2007). De l'Oriente Objet à l'Oriente Tâches – Des modèles embarqués pour l'intégration et le traçage d'un nouveau type de composants. Revue d'Interaction Homme Machine - Journal of Human-Machine Interaction, 8(1), 1 – 33.

Meyer, B. (1999). The Significance of Components. Beyond Objects column, Software Development, 7(11).

Montilva, J., Arape, N., Colmenares, J.A. (2003). Desarrollo de Software Basado en Componentes. Actas del IV Congreso de Automatización y Control (CAC03), Mérida, Venezuela.

Morch, A., Stevens, G., Won, M., Klann, M., Dittrich, Y., Wulf, V. (2004). Component-Based Technologies for End-User Development. Communications of the ACM. 47(9), 59 – 62.

Mori, G., Paternò, F., Santoro, C. (2002). CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Transactions on Software Engineering, 28(9), 797-813.

Nierstrasz, O. (1995). Requirements for a Composition Language. Proc. of the ECOOP'94 Workshop on Object-Based Models and Languages for Concurrent Systems. LNCS, Springer-Verlag, 924, 147–161.

Rojas, M., Molina, J.C. (2004). Introducción y Principios Básicos del Desarrollo de Software Basado en Componentes, Proyecto: Jerarquía y Granularidad de Componentes de Software para PYMES en Bogota.

Szyperski, C., Pfister, C. (1997). Special Issues in Object-Oriented Programming. Summary of the Workshop on Component Oriented Programming (WCOP'96). Workshop Reader of ECOOP'96. Dpunkt Verlag.

Szyperski, C. (1998). Component Software. Beyond Object-Oriented Programming. Addison-Wesley.

Weck, W., Bosch, J., Szyperski, C. (1997). Summary of WCOP'97. Proc. Of the ECOOP'97. Workshop on Component Oriented Programming (WCOP'97), LNCS, Springer-Verlag. 1357.

Wiklik, A. (2004). Réingénierie de L'Interface du Logiciel GIMP (GNU Image Manipulation Program), Mémoire de maîtrise, Université de Montréal.

Clasificación de los lenguajes de programación, http://www.alegsaonline.com/art/13.php

Desarrollo de Software Basado en Componentes, http://scruz334.blogspot.es/1193553000/desarrollo-de-software-basado-en-componentes-/

Berenguer , Escribir Programas Interactivos, http://www.iua.upf.es/formats/formats1/a01et.htm

Escribir Programas Interactivos, http://www.iua.upf.es/formats/formats1/a01et.htm

Paradigme de programmation, http://fr.wikipedia.org/wiki/Paradigme_(programmation)