Hacia una herramienta Model-Driven para mejorar la Interacción Programador-Computadora

Pedro C. Santana, Francisco J. González, Ricardo Acosta-Díaz, Juan Contreras-Castillo
Facultad de Telemática
Universidad de Colima
C.P. 28040, Colima, Colima, México
{psantana, fgvega, acosta, juancont}@ucol.mx

Resumen— Este trabajo presenta la herramienta CASE llamada Modelin, la cual propone la utilización del concepto Model-Driven Development en los entornos integrados de desarrollo (IDE por sus siglas en inglés), esto con el objetivo de mejorar la Interacción Humano-Computadora entre los desarrolladores de software y las herramientas para su creación. Esta herramienta CASE permitirá reducir la curva de aprendizaje para los programadores novatos, debido al uso de modelos en lugar de código fuente.

Palabras clave — model driven desarrollo de software, interacción programador-computadora**.**

I. INTRODUCCIÓN

El desarrollo de software en la actualidad se enfoca en escribir programas para computadora con el fin de controlar, auxiliar y llevar a cabo diferentes tipos de procesos.

Los desarrolladores de software en la actualidad escriben programas para controlar diferentes tipos de procesos. Los lenguajes de programación y herramientas utilizados en la actualidad han sido diseñados para programadores profesionales y sin énfasis en la usabilidad lo cual hace que sean muy difíciles de aprender a utilizar por programadores novatos. Esto aunado a los problemas ya conocidos en el desarrollo de software: proveer alta calidad, poco tiempo para salir al mercado, debe ser fácil de darle mantenimiento. Especialmente en un escenario de cambios constantes de los requerimientos.

El proyecto Modelin consiste en una herramienta que permitirá reducir significativamente la curva de aprendizaje y el esfuerzo necesario para desarrollar software por programadores no profesionales. La contribución clave de nuestro trabajo es el uso del paradigma Model-Driven Development (MDD) [1, 2], en el cual en lugar de requerir que los desarrolladores de software usen un lenguaje de programación para escribir cómo un sistema es implementado, les permite utilizar modelos para indicar qué funcionalidad del sistema es requerida. Los modelos se representan como una combinación de texto y gráficos. Estos modelos son los creados durante la fase de diseño, por lo que se reduce la brecha entre esta fase y la de programación.

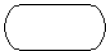
Este movimiento a niveles aún más altos de especificación permite una reducción drástica de la complejidad a problemas considerados difíciles con los estándares y métodos actuales.

II. HERRAMIENTAS DE MODELADO

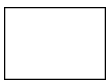
Las herramientas de modelado asisten a los desarrolladores de software a construir los modelos. El objetivo principal de las herramientas de modelado es ocultar a los desarrolladores la sintaxis de los lenguajes de programación, y proveer una interfaz conveniente para que los desarrolladores especifiquen la gran cantidad de información que se almacena en los modelos.

Como reporta [3], un modelo concreto apropiado puede tener un efecto positivo fuerte en la usabilidad de un lenguaje de programación. Probablemente el sistema de programación para usuarios finales (end-user programming system) más exitoso a la fecha son las hojas de cálculo, esto debido en parte a su efectiva metáfora con las tablas financieras [4].

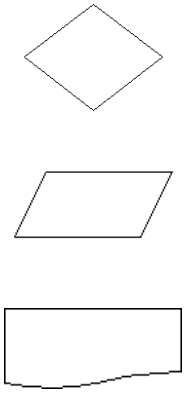
Desafortunadamente la metáfora de la hoja de cálculo no puede ser usada de manera general, por lo que se propone utilizar la metáfora de los diagramas de flujo (flowcharts). Un diagrama de flujo es el modelo gráfico de un algoritmo el cual consiste en una serie de pasos definidos, precisos y finitos representados por medio de una serie de elementos gráficos (ver tabla 1).



TERMINAL: Indica el inicio y el fin de un Diagrama de Flujo.



PROCESO: Cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.



DECISIÓN: Indica operaciones lógicas o de comparación de datos, normalmente dos, y en función del resultado de la misma determina cual de los distintos caminos alternativos del programa se debe seguir.

ENTRADA: Indica la asignación de un valor de una variable tomando este valor desde un dispositivo de entrada.

SALIDA: Indica que el resultado será presentado ya sea por pantalla o por impresora.

Tabla 1. Modelos en diagramas de flujo

Los diagramas de flujo son lo que en términos de MDD se conoce como PIM (platform independent model – por sus siglas en inglés), los PIM son transformados en PSM (platform specific model – por sus siglas en inglés), lo que incluye la información sobre tipos de datos, bases de datos, interfaces, etc., y los cuales serán convertidos a código fuente de un lenguaje de programación específico.

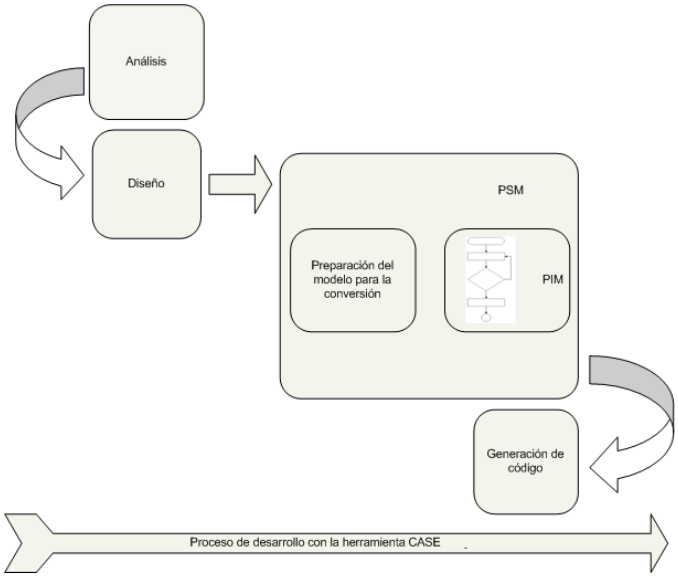


Figura 1. Arquitectura de Modelin.

Partiendo de esta idea Modelin permite la creación de diagramas de flujo y realiza la conversión a código fuente en Pascal de dicho modelo (ver Figura 1).

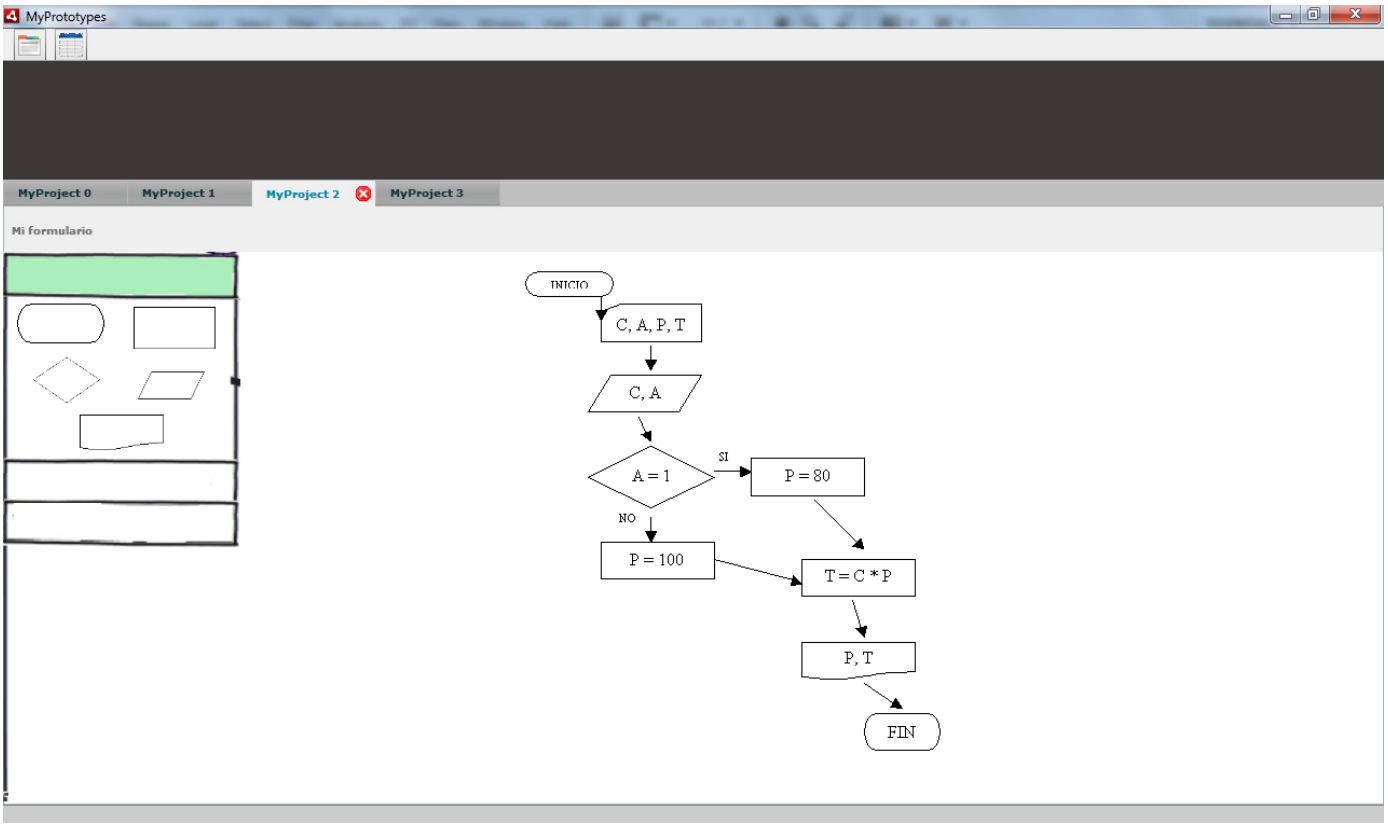


Figure 2. Modelin una herramienta que permite a los desarrolladores de software crear aplicaciones diagramando sólo los modelos.

I. Enfoque centrado en los programadores

Programar es una actividad humana, por lo que si deseamos mejorar la habilidad para programar de las personas, tiene sentido que veamos los factores humanos, es decir la HCI. Conociendo la importancia de los aspectos humanos dentro de la programación de computadoras, y tomando un enfoque que se centre en la usabilidad y el diseño centrado en el usuario (en

este caso los programadores) a través de todo el proceso de diseño se pretende lograr que Modelin (ver figura 2) permita alcanzar la simetría en la cual el esfuerzo de un programador no sea mayor al esfuerzo que le toma a la computadora traducir el programa a su forma ejecutable y de esta forma el programador sea más eficiente en su trabajo.

II. TECNOLOGÍA

Modelin es desarrollado con tecnología Flash utilizando el framework Flex, lo cual le permite ejecutarse en navegadores web que cuenten con el plugin de de Flash instalado.

Actualmente Modelin facilita la generación de código desde un diagrama de flujo al lenguaje de programación Pascal. Esto debido a que Pascal es ampliamente utilizado como lenguaje de programación para enseñar la lógica computacional a los programadores novatos, ya que permite programar de una forma estructurada utilizando las mejores prácticas de programación.

III. CONCLUSIONES Y TRABAJO FUTURO

Esta investigación considera que para obtener el objetivo propuesto es muy importante y necesario conducir estudios empíricos para identificar escenarios de uso reales, los cuales envuelvan programadores en su ambiente de trabajo. El enfoque de la evaluación se centrará en como Modelin puede ser usado para mejorar la interacción programador-computadora y reducir la curva de aprendizaje. La validación del sistema se llevará acabo conduciendo una evaluación de largo plazo con los alumnos de la Facultad de Telemática de la Universidad de Colima durante un curso de programación. El objetivo es compararlo con las técnicas de desarrollo de software convencionales. Además se espera que esta evaluación nos provea información adicional que nos permita refinar nuestro diseño y explorar algunas características que no hayan sido implementadas de forma correcta en este prototipo. También es importante en el futuro ampliar el número de tipos de modelos soportados, como por ejemplo incluir soporte para UML, así como el número de lenguajes de programación para transformar los modelos creados.

REFERENCES

[1] J. Mukerji & J. Miller (ed), Model Driven Architecture, http://www.omg.org/cgibin/doc?ormsc/2001-07-01, July 2001.

[2] David S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing, OMG Press, 2003.

[3] Mayer, R.E. (1989). The Psychology of How Novices Learn Computer Programming. Studying the Novice Programmer. E. Soloway and J. C. Spohrer. Hillsdale, NJ, Lawrence Erlbaum Associates: 129-159.

[4] Nardi, B.A. (1993). A Small Matter of Programming: Perspectives on End User Computing. Cambridge, MA, The MIT Press.