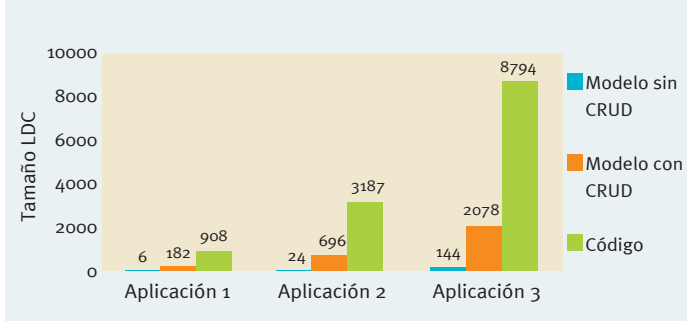
Para aumentar la productividad, Heinsohn desarrolló un proyecto en el que se aplicaron los principios de MDE. En particular, se propuso un lenguaje de modelaje llamado ISML (*Interface Specification Meta-Language*), que aporta facilidad de abstracción, es comprensible, permite la automatización, es flexible, soporta los elementos adecuados para modelar proyectos de *software*, controla versiones y es fácil de adoptar. Además, se puede integrar con *frameworks* y componentes existentes.

Luego se escogió una arquitectura por capas. En la de presentación, las páginas invocan los controladores, que a su vez acceden a la capa de servicios y lógica. Por su parte, la lógica utiliza funcionalidades de persistencia, para guardar, consultar, actualizar o remover un objeto. Para generar los artefactos de las diferentes capas, el ingeniero solo debe especificar las entidades de negocio y las páginas usando ISML. Éste ofrece constructores en lenguaje natural, independientes de las tecnologías específicas abordadas en la arquitectura de referencia.

En la segunda etapa se implementaron la gramática de ISML usando XText y el generador de las diferentes capas para tecnología Java. Las tecnologías usadas en cada capa siguen: presentación en JSF (*Java Server Faces*) y JavaFX, servicios en REST, lógica y persistencia en JEE. En fases próximas se diseñarán nuevos generadores del *stack* completo para .NET y capa de presentación en Angular. "Esto no es muy difícil, ya que la semántica de ISML es clara y se tienen los generadores base. Por ejemplo, el de JavaFX lo hizo un solo ingeniero en mes y medio", explicó Catalina Acero.



Fuente: Heinsohn

El gráfico muestra la cantidad de líneas de código escritas trabajando con y sin modelos en distintas aplicaciones.

Este proyecto ha sido probado con pilotos en aplicaciones muy pequeñas. De acuerdo con las estadísticas de Heinsohn, para desarrollar el Crud de una entidad de negocio en las tecnologías descritas, se tienen que escribir 8794 líneas de código. En contraste, trabajando con modelos, solo se tienen que escribir 4,6 líneas. El código generado es de alta calidad, ya que la arquitectura de referencia aplica buenas prácticas como, por ejemplo, asegurar un correcto uso del manejo de las transacciones, uno de los errores más habituales. Además, en presentación no se verán los llamados 'errores groseros', porque se surten las validaciones básicas. "Por supuesto es inevitable la participación de los ingenieros, posterior a la generación, ya que ellos son los encargados de la implementación de lo grueso del negocio, de la lógica, pues, por muy bueno que sea el lenguaje de modelado, ese aspecto no se refleja. El reto es cómo madurar esta situación, aunque no sea 100 % automática", señaló.

Todo esto, afirmó Catalina Acero, impacta el paradigma de los roles y actividades de los ingenieros. "Si pudiéramos masificar el uso de esta aproximación, podrían centrarse más en el modelo de negocio porque los detalles específicos de la tecnología se manejan desde los generadores. También podrían percatarse de inconsistencias en los requerimientos y aportarle más al cliente".

Los desafíos para implantar MDE incluyen la curva de aprendizaje, así como pensar diferente, en otro nivel de abstracción. También el cambio en el paradigma de programación, no solo entre los ingenieros sino también en la industria. Un reto más se presenta por el hecho de que los generadores evolucionan rápidamente, tal como la tecnología y las arquitecturas. Es decir, se deben seguir manteniendo. ■

Modelaje, eficiencia para un **equipo de desarrollo pequeño**

Crear aplicaciones en una compañía ajena a la tecnología fue el reto de este proyecto que ha encontrado en MDE una herramienta para mejorar la productividad. Aunque el proceso continúa, ya dio origen a una *startup* emergida del Grupo GHL.

Versatilidad en la generación de código, facilidad en la corrección de errores y problemas en la capa de presentación son los principales resultados que mostró Andrés Yie, director de tecnología del GHL. Hace tres años emprendió este proyecto para producir

software de manera eficaz con un pequeño grupo de desarrolladores, en una compañía que no es de tecnología. Su charla se tituló "Modelos en el mundo real, experiencias aplicando MDE (*Model-driven Development Engineering*) en la industria".

Hasta el momento han generado seis aplicaciones basadas en las tecnologías de modelaje de .NET. En los tres años han producido cerca de 400.000 líneas de código .NET, 244.000 de ellas generadas automáticamente. En algunos proyectos se puede llegar a un mejoramiento de la productividad del 74 %; en otros este índice apenas es el 30 %.

Simplificar la capa de presentación

La experiencia comenzó definiendo una arquitectura por capas. Se dio a cada capa y a cada componente una razón de ser para que las aplicaciones sean fáciles de mantener. Esto permite ubicar rápidamente cualquier problema y corregirlo.

Luego decidieron separar la generación de código: una capa para la lógica de negocio y la otra para la presentación (la que ve el usuario), con dos modelos diferentes. En el de negocio se especificaron las entidades (representaciones de las tablas de la base de datos), atributos y relaciones.

El de presentación permite diseñar formularios, vistas, páginas y componentes gráficos. Este modelo está compuesto de varios archivos, de tal forma que los desarrolladores trabajan independientemente.

Foto: Natalia Fernanda Madrid Vidales

Andrés Yie contó cómo fue la experiencia al trabajar con modelos para el pequeño equipo de tecnología del Grupo GHL.

El Grupo GHL es una empresa de operadores hoteleros cuya misión no es la producción de *software*. Aun así, el trabajo del equipo de tecnología dio origen a una *startup* gracias a la eficiencia en la producción con modelos.

Cumplida esa etapa, se procedió a generar el código para la capa de lógica, especialmente el Crud y algunos elementos del manejo de idiomas y auditorías. El mayor porcentaje de producción automática ha sido en las capas de lógica y acceso a datos.

Del modelo de presentación se derivaron los formularios básicos. Pero el 70 % de la creación de esta capa sigue siendo manual ya que hay mucha funcionalidad que corre del lado del cliente. Esto último es una decisión arquitectónica para favorecer la interacción entre el usuario y la aplicación generada. Hay un reto de mejorar aún más la generación de la capa de presentación, así como las funcionalidades de seguridad. El avance que se ha hecho con respecto a este punto es simplificar el modelo de la capa de presentación para producir artefactos de base, que sirvan de punto de partida al desarrollador para construir formularios más complejos. La principal dificultad para generar esta capa radica en el modelaje de las relaciones entre los distintos eventos.

Para conectar el modelo de lógica y de presentación, es posible establecer qué formulario corresponde a cuál entidad y qué campo a cuál propiedad de determinada entidad.

Al hablar del mantenimiento de lo generado, Andrés Yie relató la experiencia con aplicaciones financieras. "Cuando había muy alta concurrencia, detectamos que se producían muchos errores de conexión a la base de datos, por un problema en la arquitectura originalmente definida. Teníamos unas 200.000 líneas de código que habríamos tenido que corregir a mano. Sin embargo, lo que hicimos fue modificar el generador y este produjo el 90 % del código sin defectos. El 10 % restante se hizo manualmente".

Conclusiones y retos

Aunque las tecnologías de modelaje de .NET son buenas, versátiles, fáciles de usar y generan editores gráficos profesionales para crear los modelos sin esfuerzo, una gran limitante es que el soporte de transformaciones es deficiente.

El lenguaje C# tiene herramientas especiales para separar en archivos el código generado del código manual y eso permite trabajar con claridad y establecer dónde el desarrollador debe agregar el código faltante.

En el trabajo futuro está previsto, además de replantear el modelo de la capa presentación, estudiar si en este caso se puede emplear uno de variabilidad, para lo cual habría que definir qué librerías se usarán, qué componentes; y generar y modelar lo que corresponde a seguridad. ■