Una experiencia en la implementación del método AFP

Carlos J. Fernández Candel[1], José R. Hoyos Barceló[1], Jesús García-Molina[1], Francisco J. Bermúdez Ruiz[1] y Benito J. Cuesta Viera[2]

[1]Grupo Modelum, Facultad de Informática, Universidad de Murcia
[2]OpenCanarias

{carlosjavier.fernandez1, jose.hoyos, jmolina,fjavier}@um.es, bcuesta@opencanarias.es

Abstract. OMG lanzó en 2014 la propuesta *Automated Function Point* (AFP) para automatizar el conteo de puntos de función de una aplicación *legacy* en procesos de modernización a partir de modelos KDM. En el contexto de una colaboración entre el grupo Modelum (Universidad de Murcia) y la empresa Open Canarias se ha desarrollado una implementación de AFP que se está evaluando para código Oracle Forms. En este trabajo se describe la experiencia de implementación: motivación, arquitectura y desafíos para completarla.

Keywords. MDE, KDM, Automated Function Points, Oracle Forms

1 Introducción

El análisis de puntos de función (*Function Point Analysis*, FPA) es una técnica ampliamente usada para estimar el tamaño funcional de una aplicación [1]. Un punto de función (FP) es una métrica para medir la funcionalidad de una aplicación desde el punto de vista de los requisitos funcionales. A mayor número de FP, mayor será la complejidad de una aplicación y el esfuerzo de implementación.

La aplicación de FPA no es sencilla y requiere un esfuerzo considerable. El conteo FPA es realizado por expertos con la certificación que otorga la IFPUG (asociación encargada de promover el uso de FPA). Por ello, hay un interés creciente en automatizar el proceso de conteo y existen varias herramientas que lo llevan a cabo para diferentes plataformas de desarrollo. La mayoría asisten al experto a realizar el conteo y unas pocas realizan el conteo automáticamente (de ellas destaca CAST [2]).

En enero de 2014, OMG publicó la propuesta *Automated Function Point* (AFP) para automatizar el cálculo de FP en proyectos de modernización [3]. AFP es parte de la iniciativa ADM (*Architecture-Driven Modernization*) [4] que consiste en la definición de varios metamodelos para facilitar principalmente la interoperabilidad de herramientas de modernización. KDM (*Knowledge Discovery Metamodel*) [5] es el principal metamodelo de ADM y está destinado a representar el código de las aplicaciones de una forma independiente del lenguaje. SMM (*Software Metrics Metamodel*) [6] es otro metamodelo que forma parte de ADM para representar métricas.

Desde mediados de la década pasada son muchas las empresas que desean migrar sus aplicaciones *legacy* Oracle Forms a plataformas modernas. La modernización de aplicaciones es una de las líneas de negocio de la empresa Open Canarias. Recientemente, esta empresa ha arrancado el proyecto CDTI "MORPHEUS: Modemization of Reports, Procedural Code and The User Interface" destinado a automatizar la migración de aplicaciones Oracle Forms. El grupo Modelum (Universidad de Murcia) ha sido contratado para realizar tareas de investigación dentro del proyecto, en concreto abordar la migración de la lógica de negocio a código Java y definir una estrategia de estimación de costes. En la actualidad Open Canarias realiza la estimación de los costes de migración de Forms mediante su estrategia *OraAssesment* que se basa en métricas de esfuerzo establecidas por ella y en su experiencia en proyectos anteriores.

Durante la ejecución del proyecto Morpheus se decidió realizar una estimación de costes basada en AFP, ya que se disponía de un inyector de modelos KDM del código. La única herramienta que se considera compatible con AFP es CAST, pero esta herramienta tiene un precio elevado y las medidas que calcula tienen que ver con cambios en una aplicación existente en vez del coste de la migración, además no se han publicado detalles de su implementación. Por ello, hemos explorado obtener una implementación propia de AFP.

En este artículo presentamos el trabajo realizado hasta la fecha en la implementación del algoritmo AFP. Se describe la solución ideada, las dificultades encontradas y los desafíos a los que deberemos enfrentarnos.

2 Estrategia de implementación

La Figura 1 muestra los pasos que se deben seguir para implementar AFP. El código y los datos deben ser inyectados en un modelo KDM. Este modelo debe ser independiente de la plataforma y contener toda la información necesaria para el conteo. Si esto no fuese así sería necesario refinarlo para obtener un modelo KDM de la Aplicación (KDM *Application model*). A continuación se aplica una transformación modelo a modelo que implementa el algoritmo AFP de conteo de FP para generar un modelo SMM que representa las métricas calculadas. A partir de este modelo se obtendrían los diferentes informes por medio de transformaciones modelo a texto.

Para probar la implementación, se ha usado el inyector desarrollado por Open Canarias que genera el modelo KDM de la base de datos y un modelo de la interfaz de usuario que tiene relaciones a elementos KDM. Algunas partes del modelo están ligadas a Forms, en especial el uso de estereotipos propios para registrar los tipos de operaciones en vez de usar micro-KDM.

El conteo de los FP en AFP se basa en calcular cuatro tipos de magnitudes relacionadas con: datos de la aplicación (ILF), datos externos (EIF) y transacciones sobre datos (EI para las de creación y mantenimiento y EO para el resto). A su vez, la medición de esas magnitudes requiere calcular el número de DET (dato elemental en una unidad lógica de datos, ej. un campo de una tabla), RET (agregación de datos

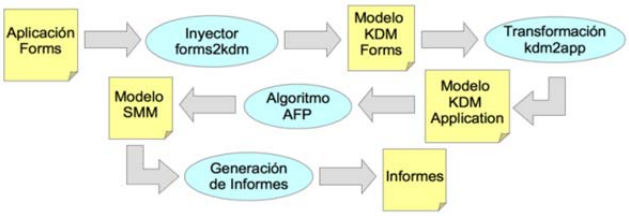

Fig. 1. Transformaciones para la obtención de PF.

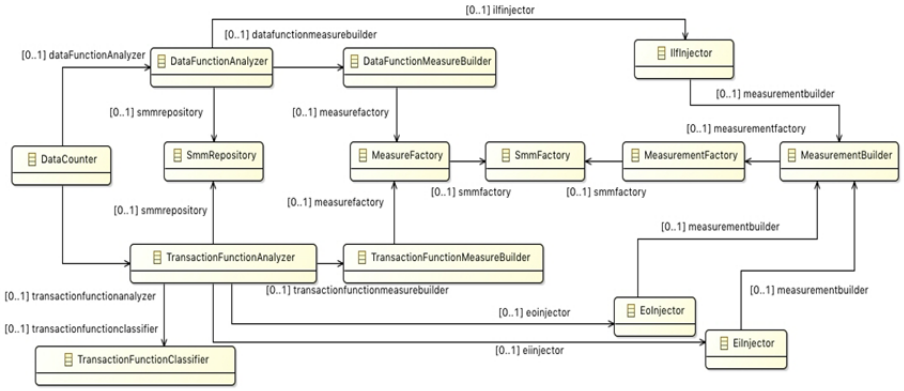

Fig. 2. Diagrama de clases de la aplicación.

elementales, ej. una relación maestro-detalle) y FTR (tablas o ficheros referenciados desde una transacción). Para realizar el conteo de estas magnitudes, hemos organizado la ejecución en tres etapas: i) realizar la medición de los DET, RET y FTR; ii) procesar las mediciones anteriores para medir las magnitudes ILF, EIF, EI y EO; y iii) inyectar estas mediciones en un modelo SMM.

La Figura 2 muestra el diagrama de clases de la aplicación que implementa AFP. La clase *DataCounter* mide los DET, RET y FTR. Para ello requiere de un análisis de datos (*DataFunctionAnalyzer*) y de código (*TransactionFunctionAnalyzer*). Las clases *MeasurementBuilder* y *DataFunctionMeasureBuilder* se encargan de ir creando el modelo SMM a partir de las mediciones realizadas, y usan factorías para la construcción de los elementos.

La identificación de los ILF se realiza en el *DataFunctionAnalyzer* utilizando el modelo de datos de KDM que representa las diferentes tablas. Para cada tabla se cuenta el número de DET identificando las columnas que no son autogeneradas y el número de RET realizando un análisis del modelo KDM para encontrar sentencias SELECT que realizan una unión entre dos tablas. La distinción entre EI y EO se realiza en la clase *TransactionFunctionClassifier* a partir del modelo de la UI recorriendo los bloques de código asociados a elementos visuales. Para comprobar si un bloque corresponde a un EI se examina el código para identificar inserciones, actualizaciónes o borrado de datos. En el caso de un EO se comprueba si existen sólo sentencias de lectura. El conteo de los DET y FTR para una función EI o EO se realiza recorriendo todas las sentencias del bloque de código y accediendo a todas las

sentencias SQL. Para cada sentencia se realiza la suma de todas las columnas utilizadas para determinar los DET y de todas los ILF utilizados para contar los FTR. La clase *MeasureBuilder* se encarga de inyectar las métricas en un modelo SMM.

3 Problemas en la implementación de AFP

La aplicación del algoritmo AFP requiere disponer de un modelo KDM de la aplicación que disponga de toda la información necesaria para el conteo y que ésta se registre de una forma independiente de la plataforma. En cuanto a que los modelos KDM sean completos, hemos observado que las plataformas pueden dificultar que sea posible inyectar toda la funcionalidad. Por ejemplo, Oracle Forms ofrece funcionalidad común a todas las aplicaciones que no puede ser descubierta por un análisis de código dado que es implícita y no requiere escribir código (ej. ejecutar una consulta de datos con sólo pulsar una tecla). De igual forma, los *widgets* suelen estar ligados con elementos de la base de datos de manera que se puede ejecutar automáticamente una sentencia interna INSERT o UPDATE al escribir un dato en una casilla de texto o marcar una checkbox, y dichas instrucciones tampoco requieren escribir código. Por otro lado, hemos encontrado que los modelos obtenidos con el inyector de la empresa tienen información cuya representación es propia de Forms (por ejemplo, se han usado estereotipos propios para especificar los tipos de operación). El primer problema requiere la definición de un modelo de plataforma que especifique la funcionalidad implícita y el segundo se podría resolver usando micro-KDM. En la actualidad, nos encontramos abordando ambas tareas.

Por otra parte, en el caso de aplicaciones Oracle Forms (y aplicaciones RAD en general), al tratarse de una programación dirigida por eventos, el código está disperso en “triggers” y el orden en el que se ejecutan está condicionado por los eventos que se generan. Esto hace difícil para el inyector identificar todo el código asociado a una misma función transaccional y los datos que maneja.

Finalmente, comentar que se ha ejecutado la versión actual de nuestra aplicación para diferentes *forms* proporcionados por la empresa y que los resultados son conformes a su tamaño pero no estamos considerando toda la funcionalidad.

References

1. Function Point Counting Practices Manual, version 4.3.1, IFPUG, 2010
2. http://www.castsoftware.com/products/automated-function-points-counting-sizing-estimation-tool
3. Architecture-Driven Modernization, OMG, http://adm.omg.org/
4. Architecture-Driven Modernization: Knowledge Discovery Meta-Model (KDM) v1.3, August 2011, OMG Document Number: formal/2011-08-04
5. Software Metrics Meta-Model (SMM) Version 1.0, January 2012, OMG Document Number: formal/2012-01-05, http://www.omg.org/spec/SMM/1.0/
6. Luigi Lavazza: Automated Function Points: Critical Evaluation and Discussion. WETSoM@ICSE 2015: 35-43