Una infraestructura de meta-nivel para el desarrollo de software multiplataforma

Julieta Alvarez, Mariano Appendino, Alexis Ferreyra,
Ricardo Medel, Emanuel Ravera

Laboratorio de Investigación de Software
Departamento de Ingeniería en Sistemas de Información
Universidad Tecnológica Nacional – Facultad Regional Córdoba

Maestro M. Lopez esq. Cruz Roja Argentina
Ciudad Universitaria – Córdoba
{juli.cba, marianoapp, alexis.ferreyra, ricardo.h.medel, ravera.emanuel}@gmail.com

Resumen

Dentro de esta línea de I+D nuestro grupo se concentró en generar una solución que permita la introspección de código para los desarrolladores multiplataforma que utilicen JavaScript.

El objetivo principal se centra en entregar una capa de meta-programación que permite re-escribir código JavaScript al vuelo o de manera estática para que el mismo pueda correr en diferentes plataformas de manera transparente. Otra de las principales aplicaciones permite la reescritura por código demostradamente más eficiente y la obtención de información de tipos del código para evitar errores en *runtime*.

Palabras Clave: *Desarrollo Multiplataforma, JavaScript, HTML5, Introspección, Re-escritura.*

Contexto

Las tareas descriptas en este trabajo se enmarcan en el proyecto "*Desarrollo de infraestructura de meta-nivel sobre JavaScript para el desarrollo de software multiplataforma*", el cual se realiza como parte de las actividades de investigación y desarrollo del Departamento de Ingeniería en Sistemas de Información de la Facultad Regional Córdoba de la Universidad Tecnológica Nacional. El proyecto, comenzado el 1 de mayo de 2013, es dirigido por el Dr. Ricardo Medel y tiene como objetivo principal la generación de una infraestructura para la construcción de software multiplataforma sobre el lenguaje JavaScript. El proyecto es financiado por la Secretaría de Ciencia, Tecnología y Posgrado de la UTN bajo el código UTN1701 (Disposición SCTyP 173/13).

Introducción

Aunque existen varios estándares para la programación de páginas web y dispositivos móviles, aún es un problema para los desarrolladores la multiplicación de plataformas incompatibles entre sí. En este proyecto proponemos utilizar técnicas de meta-programación a fin de superar estos inconvenientes.

A pesar de los constantes avances tecnológicos, las diferencias entre las distintas plataformas web y móviles siguen siendo un desafío diario para los desarrolladores de software. Tecnologías como JavaScript [1] y HTML se han posicionado como una posible

solución al desarrollo ágil de aplicaciones multiplataforma. Sin embargo, a pesar del desarrollo de estándares para HTML, aún existen importantes diferencias entre las entidades que definen los estándares (W3C, ECMA, etc.) y los proveedores de los *runtimes* para la web (Opera, Google, Microsoft, Mozilla, etc.).

Por otra parte, los lenguajes dinámicos han incluido algunas capacidades de meta programación prácticamente desde el comienzo de la historia de la programación. Entre los más importantes podemos nombrar a Lisp [2], cuya función "eval" acepta como argumento una instancia de estructura de datos que representa directamente código ejecutable (*S-expression*). Otros lenguajes de programación más recientes, tales como Groovy [3] o Escala [4] también brindan mecanismos de reescritura del árbol de sintaxis abstracto (AST) en tiempo de compilación, llegando al punto de poder agregar nuevas fases en el proceso de compilación. Otro ejemplo es Meta D++ [5], que fue desarrollado por este grupo de investigación en un proyecto anterior y provee técnicas de reescritura de código a través del uso de clases especiales en tiempo de compilación que pueden escribir partes de AST de las funciones llamadoras. En cuanto a la introspección de código, muchos lenguajes proveen mecanismos simples, tales como la función "instanceOf" de JavaScript o la capacidad de preguntar si un objeto responde a un cierto mensaje, pero casi ninguno de ellos permite la introspección completa del programa.

Es por ello que, luego de observar la diversidad existente de implementaciones de JavaScript y HTML y la carencia de un *framework* que provea técnicas de meta-programación, se decidió proponer el desarrollo de una infraestructura de meta-programación para estos lenguajes, que permita a los desarrolladores reescribir su código JavaScript para las diferentes plataformas, a la que denominamos "PumaScript" [6].

Líneas de investigación y desarrollo

Dentro de las principales líneas de investigación que se plantearon para el proyecto podemos caracterizar las de introspección, análisis de tipos y reescritura de código JavaScript.

En primer lugar se atacarán casos en los que a través de la reescritura de código se pueden realizar mejoras en la eficiencia del programa. Esta funcionalidad permite cambiar ciertas instrucciones del código por aquellas que han sido probadas como más eficientes.

Una vez logrado esto se piensa agregar inteligencia al *runtime*, es decir, recolectar información de tipos que, entre otras ventajas, permita alertar al desarrollador si se quiere realizar una asignación a una variable que tiene valores de otro tipo.

Etapas del proyecto

El proyecto se dividió de manera incremental para ir utilizando los resultados en cada una de las etapas en las siguientes. Dentro de las etapas podemos destacar:

1° Etapa: Estudio del estado del arte. Se analizaron varios lenguajes dinámicos para obtener sus características, ventajas y desventajas con respecto a la reescritura e introspección de código.

2° Etapa: Diseño del metalenguaje. Se tomaron como entradas los requerimientos obtenidos en la primera etapa y se realizó el diseño de arquitectura y funcionamiento del metalenguaje.

3° Etapa: Implementación de la infraestructura de metalenguaje. Creación del soporte para la ejecución completa de JavaScript e infraestructura que permita la reescritura código. Se realizó en esta etapa la puesta a pun-

Código "PumaScript" de la función de meta nivel:

```
/* @meta */
function $(a){
   return pumaAst( jQuery(document.getElementById($a)) );
}
```

El framework realiza la siguiente reescritura en la cual se reemplaza la llamada al selector *jQuery* por el llamando a la instrucción nativa, lo cual mejora la eficiencia del programa:

```
$("#main"); → jQuery( document.getElementById('#main'));
```

Figura 1: Ejemplo de código PumaScript y el resultado de la reescritura.

to de la infraestructura del proyecto, tales como repositorios, integración continua, etc.
4° Etapa: Prueba sobre casos. Análisis de casos de mejora de performance reescribiendo el código JavaScript y generación de los resultados.
5° Etapa: Prueba con casos complejos. Búsqueda de casos más complejos donde se pueda utilizar la información de tipos con la que cuenta el *runtime* para mejorar el código del desarrollador. Dentro de las mismas es destacable el hecho de poder informarle al mismo cuando ocurran violaciones de tipos u otros posibles errores de *runtime* comunes a lenguajes dinámicos, los cuales no son fácilmente detectables por humanos.

Resultados y Objetivos

Al momento de la escritura del presente reporte el trabajo de investigación se encuentra finalizando su tercera etapa y comenzando la cuarta, en la cual la infraestructura desarrollada ha comenzado a aplicarse a casos de reescritura simple.
En las primeras etapas del proyecto se determinaron los requerimientos analizando lenguajes tales como Smalltalk, Groovy, Scala, Self y Lisp, en la búsqueda de métodos de reescritura e introspección a utilizar en el proyecto.
En la etapa posterior se realizó el diseño de arquitectura en donde se determinó el uso de Esprima [7] y Escodegen [8] como parser y como motor de escritura de JavaScript, respectivamente, a partir del AST.
Para finalizar, en la cuarta etapa se han probado casos de reescritura de funciones simples utilizando un editor desarrollado para que el usuario pueda escribir el código en notación "PumaScript" y obtener el resultante en JavaScript obteniendo resultados satisfactorios. La Figura 1 muestra un ejemplo de código "PumaScript" y su reescritura resultando en código más eficiente.

Formación de Recursos Humanos

El grupo está formado por un director formado, tres ingenieros recientemente recibidos en formación y un alumno.

Becas y Prácticas Supervisadas realizadas en el marco del proyecto:

- Un alumno de la UTN-FRC realizó la Práctica Supervisada (PS, requisito para obtener el título de Ingeniero en Sistemas de Información) en el marco de este proyecto.
- Dos ingenieros recién recibidos se encuentran realizando trabajos en el marco de la Beca de Iniciación a la Investigación y Desarrollo – BINID, financiada

por la Secretaría de Ciencia, Tecnología y Posgrado de la UTN.

Referencias

[1] ECMA Standard, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (Visitado el 13/03/2014)

[2] J. McCarthy, "The implementation of Lisp", History of Lisp, Stanford University, 1979.

[3] Groovy Compile Time Meta programming - http://groovy.codehaus.org/Compile-time+Metaprogramming+-+AST+Transformations (Visitado el 13/03/2014)

[4] Scala Language - http://www.scala-lang.org/ (Visitado el 13/03/2014)

[5] LayerD - Introducción a Tiempo de Compilación - http://code.google.com/p/layerd/wiki/Introduction_to_Compile_Time (Visitado el 13/03/2014)

[6] “PumaScript” en GitHub, https://github.com/emravera/puma (Visitado el 13/03/2014)

[7] Esprima - Project Page, http://esprima.org/ (Visitado el 13/03/2014)

[8] Escodegen - Project Page, https://github.com/Constellation/escodegen (Visitado el 13/03/2014)