ISSN 1870-4069

Despliegue de monitores con los mecanismos de reflexión y las extensiones de gestión de Java

Andoni Rodríguez-Díaz, Ulises Juárez-Martínez

Instituto Tecnológico de Orizaba, División de Estudios de Posgrado e Investigación, Orizaba, Veracruz, México

{arodriguezd,ujuarez}@ito-depi.edu.mx

Resumen. La monitorización permite a los sistemas auto-adaptables observar el estado actual de los mismos y de esta forma aplicar los cambios necesarios. El despliegue de monitores a un sistema implica añadir las instrucciones correspondientes en el código fuente y la posibilidad de afectación en el rendimiento durante la ejecución, sobre todo si los monitores emplean los mecanismos de reflexión. Este trabajo presenta un esquema para aplicar los monitores en los sistemas compilados, limitando el uso de reflexión.

Palabras clave: Monitores, reflexión, sistemas compilados.

Monitors Deployment Using Reflection Mechanisms and Java Management Extension

Abstract. Monitoring process let self-adapting systems to watch their current state and therefore making the needed changes. Deploying monitors into a system involves adding the necessary statements into the source code and the possibility to impact the system's performance during execution, also if monitors use reflection mechanisms. This paper features a scheme to deploy monitors on compiled systems, reducing the use of reflection.

Keywords: Monitors, reflection, compiled systems.

1. Introducción

Los sistemas auto-adaptables tienen la capacidad para modificarse a sí mismos en función del estado actual del entorno de ejecución, sin la necesidad de la intervención del usuario. IBM propuso el esquema de ciclo de control *MAPE-K*: monitorización, análisis, planeación y ejecución sobre una base de conocimiento; el cual es una secuencia de procesos para el diseño y desarrollo de sistemas auto-adaptables. Posteriormente, *MAPE-K* se integró en la arquitectura de "elementos autonómicos", el cual mediante el uso de sensores y actuadores se obtiene el estado de un sistema y se

aplican los cambios en los recursos de interés (elementos administrados), respectivamente [1,2].

La monitorización de un sistema permite obtener la información de su estado en tiempo de ejecución y en base a sus resultados aplican los cambios necesarios. Los sistemas distribuidos hacen uso de los monitores para la detección de errores imprevistos o para detectar modificaciones en el entorno de ejecución, ya sea desde la parte de software o hardware. Sin embargo, el despliegue de monitores a un sistema implica algunos inconvenientes como la modificación del código fuente o la generación de altos costos computacionales, lo que afecta en el rendimiento, sobre todo, si los monitores trabajan con mecanismos de análisis para obtener la información de la estructura del programa, como por ejemplo la reflexión. En [3] se enumeran algunas desventajas en cuanto al uso excesivo de la reflexión.

Este trabajo presenta un esquema que limita el uso de la reflexión para la monitorización y el despliegue de la misma en sistemas compilados. Este esquema se implementa con las herramientas de desarrollo de Java y el lenguaje AspectJ para la inyección de código en *bytecode*.

Este artículo se compone de la siguiente forma: la sección 2 se describen los trabajos relacionados con respecto a monitorización. En la sección 3 se describen brevemente las tecnologías que se utilizaron para la implementación de monitores. En la sección 4 se presenta el esquema de monitorización y la implementación del mismo. En la sección 5 se aplica la monitorización en un programa compilado. En la sección 6 se analizan los resultados. En la sección 7 se dan a conocer las conclusiones y el trabajo a futuro.

2. Trabajos relacionados

En [4] se presentó la herramienta *SPASS-meter*, la cual aplica la monitorización en los programas de las plataformas Java y Android, realiza análisis en tiempo de ejecución y presenta los resultados en tiempo real. En [5] se introdujo un marco de trabajo para la monitorización e instrumentación con capacidades de adaptación, dicho marco utiliza la interfaz de herramientas de la máquina virtual de Java (*JVM TI* por sus siglas en inglés) y los elementos de instrumentación de Java. En [6] se desarrolló una herramienta para construir modelos de rendimiento que se basan en el análisis en relación a las invocaciones en los comportamientos; esta herramienta es un agente que se aplica en la máquina virtual a través de la interfaz de *profiling* (*JVMPI*). En [2] se usó la reflexión y la característica de *proxy* dinámico para desplegar monitores, con la capacidad de adaptación en sí mismos, y realizar los trabajos de *logging* en cada invocación de un sistema. En [7] se presentó un enfoque que habilita la monitorización de programas en Java, con base en la compilación de *scripts* en código y su inclusión al sistema a través de AspectJ.

3. Tecnologías en Java

En esta sección se analizan las herramientas de programación que implementan la tecnología Java con la capacidad de interactuar con sistemas compilados.

3.1. Reflexión de Java

La característica de reflexión que ofrece Java permite realizar la introspección, que es la capacidad para analizar las estructuras de datos de un programa en tiempo de ejecución. Además, ofrece mecanismos para obtener información adicional tanto la descripción de una estructura de datos, así como también aquella en relación a los campos y métodos [8].

3.2. Extensión de gestión en Java

La extensión de gestión de Java (*JMX* por sus siglas en inglés) es una tecnología que ofrece dicha plataforma para el acceso a los recursos de los programas o servicios en ejecución. Los recursos se representan como objetos que se conocen como *MBeans*, los cuales se registran en un servidor que se denomina ``servidor *MBean*''; posteriormente estos recursos son accesibles de forma remota a través de los conectores que *JMX* ofrece, permitiendo la monitorización en los programas. Los objetos *MBean* se representan en forma estática durante la definición de una clase o de forma dinámica mediante la implementación de funcionalidades genéricas para el acceso a los datos de la clase de interés [9].

3.3. AspectJ

AspectJ es un lenguaje de programación orientado a aspectos, el cual extiende al lenguaje de programación de Java. A través del modelo de puntos de unión, AspectJ permite la separación de asuntos que afectan a distintas clases del programa principal y la encapsulación de los mismos en módulos, permitiendo la reutilización y facilitando el mantenimiento de software [10].

4. Esquema de monitorización propuesto

En esta sección se presenta el esquema para la monitorización de objetos en un sistema en tiempo de ejecución. El esquema implementa los elementos de la extensión de gestión de Java y los mecanismos de reflexión de la misma plataforma. En la figura 1 se muestra el diagrama de clases del modelo de monitorización. La clase Reflected obtiene la información de los campos y los métodos de la instancia de tipo java.lang.Class, por ejemplo java.lang.Object.

Una instancia de tipo Guardian interviene en el acceso a un objeto del sistema, dicho tipo implementa la interfaz DynamicMBean lo que permite su registro en el servidor *MBean* para la monitorización; además hace uso de la información de la clase Reflected para identificar a qué campo corresponde un valor.

El aspecto Adapting (Código 1, figura 2) realiza un corte en la llamada al constructor de una clase del sistema (línea 2), en el aviso around se construye el objeto (línea 7) y se pasa como parámetro para el constructor de la clase Guardian (línea 8), finalmente se devuelve la referencia del objeto que se construyó (línea 13).

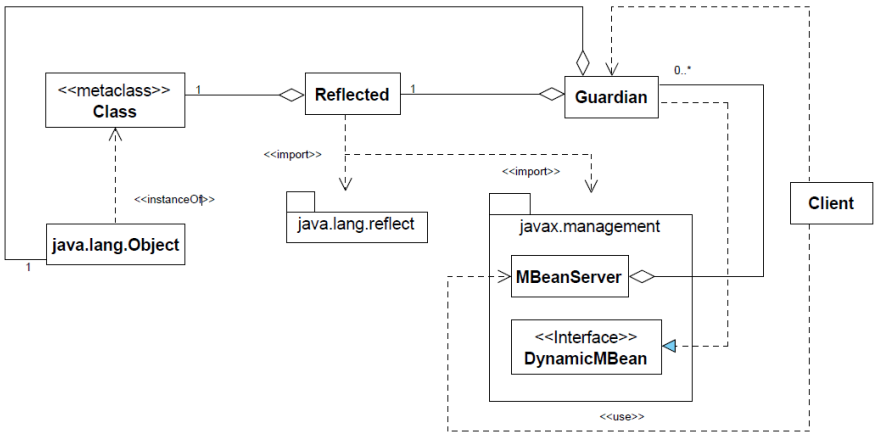

Fig. 1. Diagrama de clases del esquema de monitorización propuesto.

```
1  public aspect Adapting {
2      pointcut toBeAdaptable() : call([nombredelaclase].new(..));
3
4      Object around() : toBeAdaptable() {
5          Object object = null;
6          try {
7              object = proceed();
8              new Guardian(object);
9          }
10         catch (Exception ex) {
11             ex.printStackTrace();
12         }
13         return object;
14     }
15 }
```

Fig. 2. Aspecto para la construcción de un objeto de monitorización.

La clase Guardian (Código 2, figura 3) se compone de las referencias al objeto del sistema (línea 2) y al objeto Reflected (línea 3); durante el proceso de construcción, se obtiene la referencia a la instancia de tipo Reflected (línea 7), finalmente se agrega al servidor *MBean* (línea 8).

```
1  public class Guardian implements DynamicMBean {
2      private Object target;
3      private Reflected reflected;
4
5      public Guardian(Object target) {
6          this.target = target;
7          reflected = Reflected.getReflected(this.target.getClass());
8          ManagementServer.getServer().registerMBean(this);
9      }
10
11     //...
12 }
```

Fig. 3. Fragmento de código de la clase Guardian.

La clase Reflected (Código 3, figura 4) se compone de las referencias a la información de una clase perteneciente al sistema (líneas 3-5) y una referencia de tipo MBeanInfo, cuya información es de interés para el servidor *MBean*. La clase Reflected cuenta con un campo estático que representa el conjunto instancias de este tipo (línea 2), que se acceden a través de la invocación del método Reflected.getReflected() en la clase Guardian (Código 2); si una instancia de este conjunto coincide con la clase del objeto del sistema se devuelve la referencia, en caso contrario se crea la instancia y se aplica la reflexión para obtener la información del tipo del objeto y se devuelve la nueva referencia. Esto reduce el uso de la reflexión para consultar algún elemento de la estructura de una clase, en su lugar, dichas consultas se realizarán en los objetos que representan los campos y los métodos (línea 4 y 5).

```
1  public class Reflected {
2      private static Set<Reflected> reflectedSet = new HashSet<>();
3      private Class target;
4      private Map<String, Field> fields;
5      private Map<String, List<Method>> methods;
6      private MBeanInfo info;
7
8      //...
9  }
```

Fig. 4. Fragmento de código de la clase Reflected.

5. Ejemplo de monitorización

El ejemplo de monitorización consiste en obtener y visualizar las propiedades de las instancias de un tipo que se especifique en el aspecto Adapting (Código 1). Para visualizar la información de interés es necesario de un segundo programa, el cliente, que implemente los elementos de acceso que ofrece la tecnología *JMX* para acceder a los elementos que están bajo monitorización, el servidor. En este caso, se requiere visualizar las propiedades de cada instancia del tipo Cubo que forma parte de un programa para proyectar figuras en tres dimensiones. En el corte en punto del aspecto Adapting se define la clase Cubo (Código 4, figura 5).

```
1  public aspect Adapting {
2      pointcut toBeAdaptable() : call(treD.Cubo.new(..));
```

Fig. 5. Especificación de la clase Cubo en el corte en punto del aspecto Adapting.

Para el programa cliente (Código 5, figura 6) se obtiene la instancia al servidor *MBean* (línea 1). De esta conexión se consultan las instancias que se identifiquen con un prefijo en una expresión regular. Por cada instancia resultante de la consulta (línea 2), se muestra en pantalla el identificador completo de la misma, el nombre y el tipo del atributo que se compone en la clase de la instancia y su valor correspondiente (líneas 3-10).

```
1  MBeanServerConnection connection = getMBeanServerConnection();
2  connection.queryMBeans(new ObjectName("adapted.not.invaded.app:*"), null).
       forEach(i -> {
3      System.out.println(i.getObjectName() + " attributes:");
4      MBeanInfo info = connection.getMBeanInfo(i.getObjectName());
5      for(MBeanAttributeInfo attribute : info.getAttributes()) {
6          System.out.print("\tName: " + attribute.getName());
7          System.out.print(", Type: " + attribute.getType());
8          try {
9              Object value = connection.getAttribute(i.getObjectName(),
                   attribute.getName());
10             System.out.println(", Value = " + value);
11         }
12         catch (Exception ex) {
13             System.out.println(", Value = Not_Supported");
14         }
15     }}); //...
```

Fig. 6. Código para visualizar las propiedades de las instancias bajo monitorización.

Es necesario configurar las propiedades en la máquina virtual para permitir al cliente el acceso a los elementos de monitorización; principalmente el puerto de escucha bajo la propiedad -Dcom.sun.management.jmxremote.port=[puerto].

Al ejecutar el programa, se inicia el servidor *MBean* y se registran los objetos de monitorización. Posteriormente, el programa cliente accede al servidor y se muestra en pantalla la información que se requiere. En la figura 7 se muestra la salida resultante de la monitorización, se detectaron dos objetos del tipo Cubo (líneas 1 y 9), los cuales se identifican por el prefijo que se indicó anteriormente, el tipo de la instancia y el código *hash* de la misma; seguido de la lista de atributos de cada instancia.

```
1  adapted.not.invaded.app:type=treD.Cubo,at=64a294a6 attributes:
2      Name: lato, Type: float, Value = 2.0
3      Name: lung, Type: int, Value = 8
4      Name: poligoni, Type: [LtreD.Poligono;, Value = Not_Supported
5      Name: correnti, Type: [LtreD.Punto;, Value = [LtreD.Punto;@4ccabbaa
6      Name: origi, Type: [LtreD.Punto;, Value = Not_Supported
7      Name: posiz, Type: treD.Posicion, Value = Not_Supported
8
9  adapted.not.invaded.app:type=treD.Cubo,at=7e0b37bc attributes:
10     Name: lato, Type: float, Value = 3.0
11     Name: lung, Type: int, Value = 8
12     Name: poligoni, Type: [LtreD.Poligono;, Value = Not_Supported
13     Name: correnti, Type: [LtreD.Punto;, Value = Not_Supported
14     Name: origi, Type: [LtreD.Punto;, Value = Not_Supported
15     Name: posiz, Type: treD.Posicion, Value = Not_Supported
```

Fig. 7. Salida en pantalla del resultado de monitorización.

6. Resultados

El esquema de monitorización obtuvo la referencia de cada objeto del sistema compilado y del tipo de dato que se indicó en el aspecto (Código 1, figura 2), dicha referencia se encapsuló en una instancia y se registró en el servidor *MBean*. Durante el proceso, se analizó con la reflexión la estructura de datos del objeto perteneciente al sistema compilado, generando una referencia del análisis resultante, que posteriormente

se asoció a los objetos del mismo tipo, evitando aplicar nuevamente la reflexión, por lo tanto, se reduce el impacto al desempeño del sistema.

Se implementó un programa para acceder a los elementos de monitorización. La salida resultante de la figura 7 muestra el nombre y el tipo de cada propiedad, sin embargo, solo se visualizan los valores de las propiedades cuyos tipos son primitivos (líneas 2 y 3); al obtener los valores de las propiedades de tipos complejos, se lanza una excepción la cual indica que dicho tipo no es serializable, por lo tanto, ese valor no es accesible desde el cliente con los mecanismos de *JMX*. Una solución es incluir en la especificación de clases en el aspecto Adapting (Código 1, figura 2) aquellos tipos cuyos valores no son accesibles directamente por *JMX* y en su lugar aplicar la reflexión en tiempo de construcción.

7. Conclusión y trabajo a futuro

En este trabajo se presentó un esquema de monitorización para los sistemas compilados, el cual incorpora la reflexión de Java y los mecanismos de la tecnología *JMX*. Cuando se construye el primer objeto que corresponde a una clase, su estructura se analiza y se representa como una referencia, la cual se asocia en la creación de los siguientes objetos del mismo tipo, sin recurrir a la reflexión nuevamente. Esto reduce el impacto al desempeño del sistema evitando el uso de la reflexión para obtener la estructura interna de cada objeto.

Además, los elementos del sistema compilado se representan como instancias que *JMX* tomará en cuenta para los efectos de monitorización, evitando la modificación del programa en su representación en código intermedio. Como trabajo a futuro se implementarán mecanismos para simplificar las consultas de objetos en el servidor *MBean* y se analizarán alternativas para definir clases como puntos de unión en el aspecto para el despliegue de monitores.

Agradecimientos. Este trabajo cuenta con apoyo por parte del Consejo Nacional de Ciencia y Tecnología (CONACYT).

Referencias

1. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing mape-k feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15, Piscataway, NJ, USA, IEEE Press, pp. 13–23 (2015)
2. Dawson, D., Desmarais, R., Kienle, H. M., Müller, H. A.: Monitoring in adaptive systems using reflection. In: Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems, SEAMS '08, New York, NY, USA, ACM, pp. 81–88 (2008)
3. Eichelberger, H., Schmid, K.: Flexible resource monitoring of java programs. Journal of Systems and Software, pp. 163–186 (2014)

4. Wert, A., Schulz, H., Heger, C.: Aim: Adaptable instrumentation and monitoring for automated software performance analysis. In: Proceedings of the 10th International Workshop on Automation of Software Test, AST '15, Piscataway, NJ, USA, IEEE Press, pp. 38–42 (2015)
5. Harkema, M.: JPMT: A Java Performance Monitoring Tool. CTIT technical report series. Centre for Telematics and Information Technology, University of Twente (2003)
6. Colombo, C., Pace, G. J., Schneider, G.: Larva - safer monitoring of real-time java programs (tool paper). In: 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods, pp. 33–37 (2009)
7. Forman, I. R., Forman, N.: Java Reflection in Action (In Action Series). Manning Publications Co., Greenwich, CT, USA (2004)
8. Oracle: Lesson: Overview of the jmx technology
9. Laddad, R.: AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., Greenwich, CT, USA (2003)