INGENIERIA DE SISTEMAS

Una propuesta para la clasificación de la programación reflexiva orientada al desarrollo de sistemas autónomos

SYSTEMS ENGINEERING

A proposal for classifying reflective programming aimed at the development of autonomous systems

Francisco Moreno*, Jovani Jiménez *§, Sebastián Castañeda**

**Departamento de Ciencias de la Computación y la Decisión, Facultad de Minas, Universidad Nacional de Colombia, Colombia.*
***Ingeniería de Sistemas e Informática, Universidad Nacional de Colombia, Colombia.*
fjmoreno@unal.edu.co, § jajimen1@unal.edu.co, sebcas23@gmail.com

Recibido: 05 de Febrero de 2013 - Aceptado: Junio 7 de 2014

Resumen

El fin de este trabajo es realizar una clasificación de la reflexión en niveles, con un enfoque orientado a alcanzar la programación autónoma. La reflexión es la capacidad de un programa de conocerse, examinarse y razonar para tomar acciones y modificarse a sí mismo en tiempo de compilación o ejecución. En este artículo, se proponen cuatro niveles de reflexión de acuerdo al grado de conocimiento, capacidad de modificación, tipo de modificaciones, tiempo en que se pueden realizar las modificaciones y la capacidad de razonamiento. Luego se analizan diferentes lenguajes de programación y se clasifican sobre dichos niveles, según la capacidad de reflexión que estos soportan. De cada nivel se presenta un ejemplo y finalmente se realizan una serie de experimentos donde se comparan con versiones equivalentes de programas no reflexivos. Los experimentos mostraron que aunque los programas reflexivos fueron más costosos, en cuanto a tiempo de ejecución y codificación, permiten una gran flexibilidad y más posibilidades para el diseño y desarrollo de aplicaciones. Siendo así la reflexividad el primer paso para llegar a desarrollar sistemas autónomos que puedan simular o igualar los sistemas biológicos.

Palabras clave: *Lenguajes reflexivos, metaprogramación, programación autónoma, programación reflexiva, sistemas autónomos.*

Abstract

In this paper, we propose a classification of reflection in levels, with an approach aimed to reach autonomous programming. Reflection is the ability of a program to reason, know and examine itself to act and modify its state at compilation or runtime. In this article, it is proposed four reflection levels according to the level of knowledge, modification ability, types of modifications, time to perform modifications and the reasoning level. Then, it is analyzed different programming languages and they are classified on those levels, based on the reflection capacity that they support. Each classification level is exemplified and finally we present a series of experiments where we compare them with equivalent versions of non-reflective programs. Out experiments showed that although reflective programs were more expensive, in terms of execution time and coding, they offer great flexibility and great potential for the design and development of programs. Therefore, it is the reflexivity the first step to achieve the development of autonomous systems which can simulate or pair biological systems.

Keywords: *Autonomous programming, autonomous systems, metaprogramming, reflective programming reflective languages.*

1. Introducción

Los sistemas autónomos sensan, obtienen los cambios de su entorno y se modifican a sí mismos con el fin de satisfacer nuevos requisitos (Rich & Waters, 1988; Parashar & Hariri, 2005). La naturaleza ofrece ejemplos de sistemas autónomos, como por ejemplo el cuerpo y ciertos órganos de los seres vivos que ajustan y modifican su funcionamiento para asegurar su supervivencia. Por ejemplo, considérese el corazón cuya función esencial es transportar la sangre a todo el cuerpo. Cuando el cuerpo aumenta su ritmo de funcionamiento, por ejemplo al correr, se necesita mayor cantidad de oxígeno que cuando se está en reposo; por lo tanto, el corazón aumenta su ritmo, bombea más sangre y así se suple esta necesidad. La piel es otro ejemplo: ésta reacciona y procura mantener una temperatura adecuada de acuerdo con el ambiente.

Un sistema autónomo puede ser útil en diversos dominios. Por ejemplo, considérese una aplicación de software. Usualmente, cuando surgen condiciones no consideradas por el programa se requiere de intervención humana para evitar fallos potenciales y satisfacer los nuevos requisitos. Sin embargo, si la aplicación fuese autónoma, ésta podría detectar y corregir las fallas por sí misma. Otros dominios donde se puede aplicar la programación autónoma son la robótica, domótica y en aplicaciones industriales y militares. En particular, en la robótica ya existen algunos trabajos (Fujita & Kitano, 1998; Massa *et al*, 2007; Sariff & Buniyamin, 2006) donde los robots exhiben un comportamiento autónomo: toman decisiones según las condiciones en las que se encuentren.

Las principales dificultades para desarrollar este tipo de sistemas son la representación del conocimiento y el razonamiento. Un sistema autónomo debe tomar y procesar los datos procedentes de su entorno (representación del conocimiento), tomar decisiones, distinguir entre un funcionamiento correcto y uno incorrecto, aprender de situaciones pasadas y plantear una solución si hay nuevos requisitos (razonamiento).

La programación reflexiva (Salavert & Pérez, 2000; c2.com, 2014; Sobel & Friedman, 1996) ofrece elementos que permiten la creación de programas autónomos. La reflexión es la capacidad de un programa para modificar su estado en tiempo de ejecución. Tiene dos aspectos: introspección e intercesión. La introspección es la habilidad de un programa para observar y razonar sobre su estado. La intercesión es la habilidad de un programa para modificar y agregarse nuevo comportamiento (código) (Demers & Malenfant, 1995). Lenguajes de programación como Java, C++, DotNet, Python y PHP entre otros, soportan aspectos esenciales de la reflexión (Martin & Sánchez, 1994; Albahari& Albahari, 2012; Schlossnagle, 2008; Lutz, 2013; Summerfield, 2009).

Dado el incremento de las capacidades de cómputo y las posibilidades de comunicación, los problemas a los que se enfrentaba el hombre han cambiado también y se han hecho más complejos, requiriendo soluciones más agiles y en las cuales no se hubiera podido pensar anteriormente. En la actualidad se requiere software que pueda ser concebido a partir de la necesidad, sistemas que puedan aprender de su entorno y tomar decisiones; sistemas que puedan operar de forma autónoma, razonar, adaptarse, corregir errores, modificarse y operar sin intervención humana.

Este ha sido el sueño humano, sistemas con requerimientos cambiantes que se aproximen al comportamiento de los sistemas biológicos (Parashar & Hariri, 2005; Rich & Waters1988), pero el diseño de la solución en sí para requisitos estáticos ya es un problema para el que se ha hecho necesario plantear metodologías y marcos de trabajo en la industria del software. Lo anterior ha hecho también pensar en formas y algoritmos de diseños para programas autónomos (Kant, 1985) al igual que formas de enfrentar el problema con técnicas de inteligencia artificial, como por ejemplo programación evolutiva, modelos de grafos, redes neuronales, entre otros (Kang, *et al*., 2006; Osella, *et al*., 2006)

Para llevar estas ideas a la realidad es necesario un modelo y un lenguaje de programación que permita materializar y poner en práctica el ideal propuesto. La programación reflexiva es quien tiene un enfoque más acercado al propósito de la

programación autónoma, permitiendo desarrollar sistemas que puedan conocerse a sí mismos, tomar acciones y ser adaptativos (Sonntag, 1994). Cantwell (1982) introdujo la noción de reflexión computacional en los lenguajes de programación como "sistemas que actúan sobre sí mismos" y explico cómo construir un sistema computacional que "razone" sobre sus propios procesos de inferencia (Smith, 1982). Para conseguirlo se requiere que el lenguaje de programación se apoye en una estructura de representación (Maes, 1987) la cual contiene la meta información de los componentes del software que se ejecuta, la cual debe ser accesible por el software.

Basado en lo anterior, en nuestro artículo se propone una clasificación que permite evaluar el grado de interacción entre los programas y la meta-información; es decir, desde formas restringidas de reflexión (mecanismos simples de inspección) hasta formas más libres (modificación de un programa por sí mismo sin intervención del programador). Esta clasificación puede ayudar a determinar el estado actual de la reflexión en algunos lenguajes de programación, lo mismo que a identificar los obstáculos que se deben superar. De hecho, en (Kephart & Chess, 2003) se pone de manifiesto que la computación autónoma jugará papel clave para afrontar la crisis de la complejidad del software (Horn, P. (2001); es decir, los sistemas llegarán a ser tan complejos que serán prácticamente imposible optimizarlos, mantenerlos y coordinarlos.

El artículo está distribuido de la siguiente manera: el segundo capítulo inicia presentando los antecedentes del tema para luego expresar la clasificación realizada, exponiendo cada uno de los cuatro niveles de reflexión propuestos. A su vez, se muestran programas para ejemplificar los tres primeros niveles, los cuales son soportados en los lenguajes de programación actuales. El cuarto nivel de reflexión es asociado a la capacidad de un programa de razonar y modificarse a sí mismo; para alcanzar este nivel, se presenta una propuesta con la que se pretende alcanzar tal objetivo. Al finalizar el segundo capítulo se realiza un análisis de algunos lenguajes de programación más usados, ubicándolos en los niveles de reflexión definidos. El capítulo tres enseña los experimentos realizados indicando los resultados obtenidos para cada una de las pruebas. Finalmente se muestran las conclusiones y las referencias bibliográficas.

2. Materiales y métodos

En este capítulo se presentan los antecedentes del tema y luego los cuatro niveles de reflexión existentes en la programación autónoma, indicando en cada uno de ellos, una serie de ejemplos. También se presentan algunos lenguajes de programación y el nivel de reflexión que soportan. La metodología para la clasificación de los niveles inició estudiando diferentes lenguajes de programación y comparándolos contra lo que se plantea como propósito de la programación reflexiva. Se encontró que no todos soportaban en completitud la teoría de la programación reflexiva, pero si características de esta, tales como la información que permitía conocer, en el momento que se podía obtener, en compilación o en ejecución, y los tipos de modificaciones que permitía realizar. Tomando estas características se propusieron los niveles de reflexión, pasando desde lo más simple, conocimiento de sí mismo, el primer nivel, hasta lo más complejo, aprendizaje y autocorrección, el cuarto nivel, el cual abarca la completitud de la programación reflexiva. Luego los lenguajes fueron organizados en dichos niveles según las características que soportaban de reflexión.

2.1 Antecedentes

En la revisión de la literatura realizada no se encontró información asociada a casos similares de "clasificación de los niveles" de reflexión. Los autores muestran a lo largo del trabajo, la clasificación realizada y en la Tabla 1 el nivel de reflexión los lenguajes más conocidos de acuerdo a la propuesta propia. Es bueno aclarar que por medio de BCEL, un proyecto desarrollado por Jakarta (BCEL, 2014), se puede desarrollar un programa como el presentado en el Ejemplo 6. BCEL aprovecha el siguiente aspecto de Java: en Java se crea un código llamado BYTECODE, el cual es un código intermedio entre el código propio de la máquina y el código fuente. El BYTECODE

es interpretado por la máquina virtual de cada sistema operativo. BCEL permite ingresar código al BYTECODE así: BCEL toma el archivo con extensión .class (archivo que genera Java, donde está el BYTECODE) e incorpora allí el código nuevo. Algunas de las clases de BCEL que se pueden usar para desarrollar el programa son ClassGen, MethodGen, ConstantPoolGen, InstructionList, InstructionFactory y LocalVariableGen. Estas clases permiten modificar el BYTECODE, de esta forma se puede lograr el tercer nivel de reflexión.

2.2 Niveles de reflexión

2.2.1 Primer nivel de reflexión

El primer nivel de reflexión se refiere a la capacidad de un programa de obtener información de sí mismo en tiempo de ejecución. Por ejemplo, en tiempo de ejecución un programa puede obtener información sobre los atributos y métodos de una clase. El programa también puede crear (instanciar) objetos de dicha clase e invocar sus métodos. Considérese inicialmente el Ejemplo 1 que no usa reflexión.

Ejemplo 1.

```
1. class Base
2.  {
3.          public String s;
4. }
5. class Main
6. {
7.          public static void main(String args[])
8.          {
9.                    Base b;
10.         }
11. }
```

Donde la clase Base solo tiene un atributo s y en la clase Main solo se declara el método main donde se crea una instancia de la clase Base llamada b. Nótese que en el programa se declara la clase Base antes de ser compilada. Por otro lado, por medio de la API (Application Programming Interface) Reflection de Java (JAVATM, 2014), es posible obtener en tiempo de ejecución información sobre una clase y crear instancias de esta. Esta capacidad se considera como un primer nivel de reflexión tal y como lo ilustra el Ejemplo 2.

Ejemplo 2.

```
1. import java.util.Scanner;
2. class Ejemplo_Primer_Nivel_Reflexion{
3.   public static void main(String[] args) {
//Se lee el nombre de la clase que se desea instanciar
4.      String nombreClase = new
Scanner(System.in).nextLine();
//Se carga la definición de la clase especificada
5.  Class clase = Class.forName(nombreClase);
6.                Object b;
// Se crea una instancia de la clase especificada
7.          b = clase.newInstance(); }
8. }
```

En el Ejemplo 2 se declara una clase Ejemplo_ Primer_Nivel_Reflexion con su método main. En la línea 4 se solicita al usuario que ingrese el nombre de la clase que desea instanciar y a continuación, en la línea 5, el método forName carga en tiempo de ejecución la definición de la clase ingresada. Si la definición no se encuentra, se producirá un error. De lo contrario, se crea una instancia de la clase en la línea 7. Con el fin de ilustrar una posible utilidad de este tipo de programación considérese el Ejemplo 3 donde no se usa reflexión.

Ejemplo 3.

```
1.  class Clase1{…};
2.  class Clase2{…};
3.  class Clase3{…};
4.  class Clase4{…};
5.  class Clase5{…};
6.    import java.util.Scanner;
7.    class Main{
8.        public static void main(String args[]){
9.                String opcion =
                  new Scanner(System.in).nextLine();
10.               if(opcion.equals('1') == 1) Clase1 c1;
    // Se instancia la clase Clase1
11.               else  if(opcion.equals('2')  ==  2)
    Clase1 c2; // Se instancia la clase Clase2
12.               else  if(opcion.equals('3')  ==  3)
    Clase2 c3; // Se instancia la clase Clase3
13.               else  if(opcion.equals('4')  ==  4)
    Clase3 c4; // Se instancia la clase Clase4
14.               else  if(opcion.equals('5')  ==  5)
    Clase4 c5; // Se instancia la clase Clase5
15.               else       System.out.Println("Opción
    incorrecta");
16.               }
17.       }
```

En este ejemplo, dependiendo de la opción ingresada por el usuario, se instanciará una de las cinco clases definidas. Nótese que con el uso de reflexión, ver Ejemplo 2, se logra el mismo resultado.

La API Reflection de Java también permite obtener la información de los métodos y atributos de una clase, obtener y asignar valores a sus atributos, invocar sus métodos, entre otras acciones. Sin embargo, el desempeño de un programa de este tipo puede ser más costoso que su correspondiente versión no reflexiva. Además, este es un tipo de reflexión bajo por que el programa no modifica su código (aunque las definiciones de las clases a instanciar sí podrían cambiar).

2.2.2 Segundo nivel de reflexión

El segundo nivel de reflexión se refiere a la capacidad de un programa de recibir y ejecutar código en tiempo de ejecución. Por ejemplo, el usuario puede ingresar la definición de una clase y esta ser instanciada por el programa. Considérese el programa en pseudocódigo del Ejemplo 4 que resume los pasos para ingresar en tiempo de ejecución la definición de una clase, instanciar un objeto de ésta e invocar uno de sus métodos.

Ejemplo 4.

Pseudo código: Ejemplo_Segundo_Nivel_Reflexion

1. BEGIN
2. Ingresar nombre de la clase
3. Ingresar especificación de la clase: atributos y métodos
4. Ingresar cuerpo (implementación) de los métodos
5. Crear la clase
6. Instanciar objeto de la clase
7. Invocar método del objeto
8. END

Aunque el Ejemplo 4 también se puede llegar a desarrollar en Java por medio de la API mencionada, por facilidades de implementación e ilustración se presenta en el lenguaje de programación PL/SQL de Oracle en el Ejemplo 5.

Ejemplo 5.

```
1. CREATE OR REPLACE PROCEDURE
Ejemplo_Segundo_Nivel_Reflexion(
   nombreClase VARCHAR2,
   especificacionClase VARCHAR2,
   implementacionClase VARCHAR2,
   ejecucionMetodo VARCHAR2) IS
2.          creacionClase  VARCHAR2(200);
3.          creacionMetodos VARCHAR2(400);
4.          BEGIN
5.                    creacionClase  := CONCAT('CREATE
OR REPLACE TYPE ', CONCAT(nombreClase, CONCAT('
AS OBJECT( ', CONCAT( especificacionClase ,' )') ) ) );
6.                    creacionMetodos :=
CONCAT('CREATE OR REPLACE TYPE BODY ',
CONCAT(nombreClase, CONCAT(' AS ',
CONCAT(implementacionClase , ' END;'))));
7.                    EXECUTE IMMEDIATE
creacionClase ;
8.                    EXECUTE IMMEDIATE
creacionMetodos;
9.                    EXECUTE IMMEDIATE
ejecucionMetodo;
10.         END;
```

En el Ejemplo 5 se crea un procedimiento en PL/SQL que recibe como parámetros el nombre de la clase a crear nombreClase, la especificación especificacionClase (atributos y prototipos de los métodos), la implementación de los métodos implementacionClase y el código de ejecución ejecucionMetodo. En este último parámetro, se instancia un objeto de la clase y se invoca uno de sus métodos (esta acción se puede ejecutar en PL/SQL gracias a la sentencia EXECUTE IMMEDIATE).

Un ejemplo del contenido de los parámetros se muestra a continuación.

- nombreClase = 'class1'
- especificacionClase = 'id NUMBER(7), nombre VARCHAR(20), edad NUMBER(10), MEMBER FUNCTION numeroId RETURN NUMBER'
- implementacionClase = 'MEMBER FUNCTION getNumeroId RETURN NUMBER IS BEGIN RETURN id; END getNumeroId;'
- ejecucionMetodo = 'DECLARE claseA class1; id NUMBER; BEGIN claseA := class1(1025854,' 'Juan' ',20); id := claseA.getNumeroId; DBMS_OUTPUT.PUT_LINE(CONCAT(' 'El ID es: ' ', id)); END;'

Al ejecutar el procedimiento Ejemplo_Segundo_Nivel_Reflexion del Ejemplo 5 con los parámetros anteriores, el código *generado y ejecutado* es el siguiente.

```
CREATE OR REPLACE TYPE class1 AS OBJECT(
  id NUMBER(7),
  nombre VARCHAR(20),
  edad NUMBER(10),
  MEMBER FUNCTION numeroId RETURN NUMBER)

CREATE OR REPLACE TYPE BODY class1 AS
MEMBER FUNCTION getNumeroId
RETURN NUMBER IS
BEGIN
  RETURN id;
END getNumeroId;
END;

DECLARE
  claseA class1;
  id NUMBER;
BEGIN
  claseA := class1(1025854,'Juan',20);
  id := claseA.numeroId;
  DBMS_OUTPUT.PUT_LINE(CONCAT('El ID es: ', id));
END:
```

Se crea la clase (tipo) class1.

Se crea el cuerpo de la clase

Se crea un objeto de la clase class1 y se invoca uno de sus métodos.

2.2.3 Tercer nivel de reflexión

El tercer nivel de reflexión se refiere a la capacidad de un programa de modificar su propio código en tiempo de ejecución. La diferencia con el segundo nivel de reflexión es que el código del programa que está en ejecución se modifica a sí mismo a raíz de la intervención del usuario.

En el Ejemplo 6 se ilustra el tercer nivel de reflexión: el usuario puede insertar, modificar o eliminar código de un programa en tiempo de ejecución.

Ejemplo 6.

Pseudo código: Ejemplo_Tercer_Nivel_Reflexion

```
1. BEGIN
2.        IF el usuario desea modificar el programa THEN
3.                 Ingresar, modificar o eliminar
                   instrucciones del programa
4.        END IF
5.        Ejecutar el programa
6. END
```

Considérese un ejemplo aplicado a la programación orientada a objetos. En este ejemplo, el usuario adicionará atributos y métodos a la especificación de la clase del programa. Sea una clase Empleado así:

```
Clase Empleado
        Atributos:
                nombre
                identificación
                edad
                salario
        Métodos:
                asignar(nombre, identificación, edad)
                asignar(nombre, identificación, edad, salario)
```

Supóngase que cuando se definió esta clase no se consideraron determinados atributos. Por medio de un programa como el publicado en el Ejemplo 7 (correspondiente al pseudocódigo del Ejemplo 6), el usuario puede en tiempo de ejecución incluir en la especificación de la clase Empleado los atributos sexo y dirección y los métodos asignar (el de siete parámetros) y actualizar Salario.

Ejemplo 7

```
1.        Class Empleado{
2.        Atributos: nombre, identificación, edad y salario
3.        Métodos: asignar(nombre, identificación, edad)
4.                 asignar(nombre, identificación, edad,
salario)
5.      }
6.        BEGIN
7.         IF el usuario desea modificar la especificación de
la clase THEN
8.              Ingresar atributos: sexo y dirección
          Ingresar métodos: asignar(nombre, identificación,
edad, salario, sexo, dirección) y actualizarSalario(salario)

9.         END IF
10.       Ejecutar el programa
11.       END
```

Luego de ejecutar el programa anterior, la clase Empleado que conforma el programa quedaría así:

```
Clase Empleado
Atributos:
  nombre
  identificacion
  edad
  salario
  sexo                 } Nuevos atributos
  direccion            }
Métodos:
  asignar(nombre, identificacion, edad)
  asignar(nombre, identificacion, edad, salario)
  asignar(nombre, identificacion, edad, salario,   } Nuevos
  sexo, direccion)                                 } métodos
  actualizarSalario(salario)                       }
```

Y el programa alterado queda así:

```
1.  Class Empleado{
2.  Atributos: nombre, Identificación, edad, salario, sexo, dirección
3.  Métodos: asignar(nombre, identificación, edad)
4.           asignar(nombre, identificación, edad, salario)
5.           asignar(nombre, identificación, edad, salario, sexo, dirección)
6.           actualizarSalario(salario)
7.  }
8.  BEGIN
9.  IF el usuario desea modificar la especificación de la clase THEN
```

```
9.   IF el usuario desea modificar la especificación de la clase THEN
10.          Ingresar atributos y métodos a la clase Empleado
11.  END IF
12.  Ejecutar el programa
13.  END
```

2.2.4 Cuarto nivel de reflexión

El cuarto nivel de reflexión se refiere a la capacidad de un programa de modificarse a sí mismo de acuerdo con los cambios de su entorno. Aunque en el tercer nivel de reflexión es posible modificar en tiempo de ejecución el código del programa gracias a la intervención del usuario, en el cuarto nivel de reflexión se espera que el programa tome las decisiones necesarias por sí mismo para afrontar los cambios que eventualmente surjan (nuevos requisitos).

Para lograr un sistema como el anterior se propone la siguiente arquitectura, véase la Figura 1(a), la cual incluye los siguientes módulos:

• *Módulo de razonamiento*: toma las decisiones en el sistema. Decide cuándo, cómo y cuáles modificaciones se deben llevar a cabo sobre el sistema.

• *Módulo de conocimiento*: almacena y suministra el conocimiento al módulo de razonamiento. El conocimiento puede ser adquirido por experiencia, por la información suministrada por el usuario o por otros medios.

• *Módulo de Ejecución:* ejecuta el sistema principal, es decir, la función principal para la cual fue desarrollado el sistema, por ejemplo un sistema de vigilancia, un sistema de control automático para un robot o una aplicación de escritorio, entre otras.

Figura 1. (a) Módulos para un sistema que soporta el cuarto nivel de reflexión. (b) Arquitectura

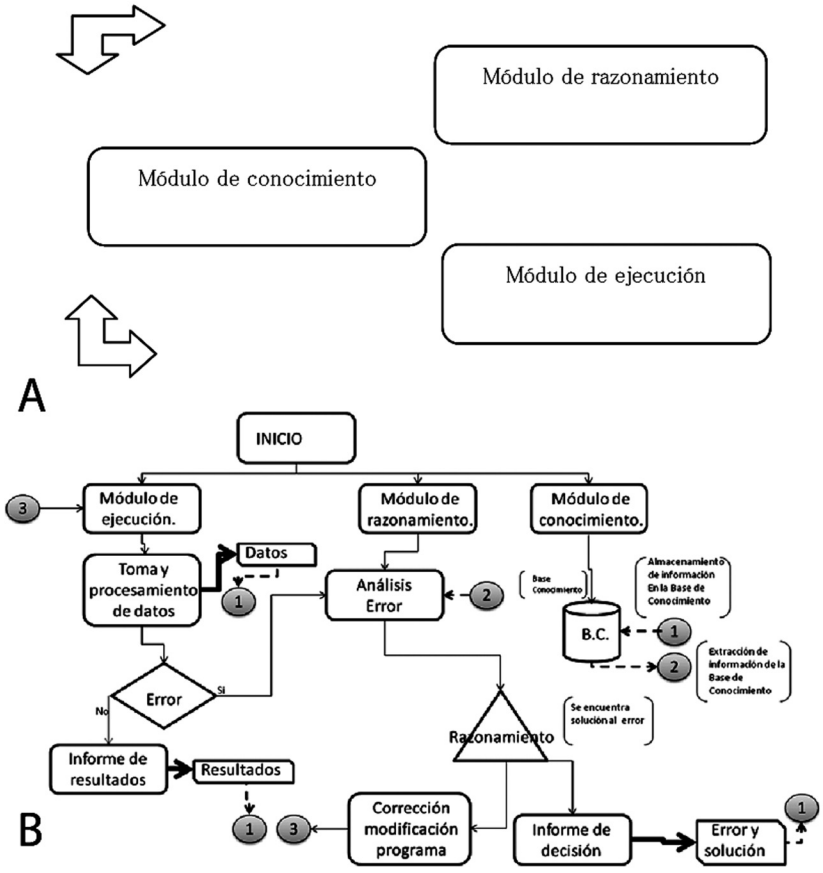

Figura 1. *(a) Módulos para un sistema que soporta el cuarto nivel de reflexión. (b) Arquitectura detallada de un sistema que soporta el cuarto nivel de reflexión. Construcción propia*

detallada de un sistema que soporta el cuarto nivel de reflexión. Construcción propia

En la Figura 1(b) se muestra un diagrama del funcionamiento más detallado del sistema. Los tres módulos se representan luego del bloque de inicio del programa. Las flechas continuas delgadas representan el flujo del programa, las flechas continuas gruesas representan accesos a la base de conocimiento y las flechas punteadas representan flujos de datos.

El módulo de ejecución inicia con la toma y procesamiento de datos, los cuales se almacenan en la base de conocimiento, para su posterior análisis. Luego se pregunta si han ocurrido errores, si no los hay, los resultados se almacenan en la base de conocimiento y se asocian con los datos tomados. Si hubo errores, estos se analizan en el módulo de razonamiento, junto con los datos tomados y se procede a buscar la solución al problema. Cuando se encuentra la solución, se hace un informe el cual incluye el error, los datos que lo generaron, el análisis realizado y la solución. Este informe se almacena en la base de conocimiento, se modifica el programa y se continúa con su ejecución.

Ejemplo 8.

Considérese un robot con un sistema de locomoción de tipo insecto (n extremidades) que va a explorar un ambiente del cual se desconocen todas las propiedades de su superficie.

La programación base del robot, contiene los tres módulos anteriores, donde el módulo de ejecución tiene la información esencial para explorar una superficie plana sin obstáculos. Sin embargo, la superficie podría ser irregular y tener diferentes tipos de obstáculos y al robot no se le ha indicado la forma de eludir los obstáculos que se presentarán en el camino.

Cuando el robot se libera en el ambiente descrito, el módulo de conocimiento adquiere la información del entorno y la transmite al módulo de razonamiento el cual debe determinar la forma adecuada de desplazamiento, según la superficie que identifique, y encontrar la forma de sortear los obstáculos que se presenten en el camino con el fin de cumplir con el objetivo asignado. Incluso si el robot perdiese una de sus extremidades, el mismo solucionaría el problema, encontrando la forma de desplazarse con las extremidades restantes. Usando el modelo planteado, el robot podría solucionar varios problemas para los cuales no fue programado inicialmente, modificando su comportamiento y adaptándose a las condiciones del medio.

2.2.5 Lenguajes de programación

En la Tabla 1 se presentan algunos lenguajes de programación y se describe el nivel de reflexión que soportan.

3. Resultados de los experimentos y discusión

Las pruebas se realizaron en un equipo con procesador AMD Phenom x3 de 2.3 GHz y 2 GB de memoria RAM y con sistema operativo Windows XP. Para los experimentos se trabajó con el lenguaje Java, JDK 1.6.0.26, sistema operativo Linux Ubuntu 10.10. Se usó la librería java.lang.reflect (librería estándar del lenguaje Java para reflexión). Se consideraron las siguientes pruebas. En la Prueba 1 se implementaron los programas de los ejemplos 2 y 3; y gradualmente se aumentó el número de clases posibles a instanciar en ambos programas. Esto implicó para el Ejemplo 2 aumentar el número de clases en el paquete donde se encontraba la clase Ejemplo_Primer_Nivel_Reflexion y para el Ejemplo 3 incorporar la definición de las nuevas clases (antes de la línea 6) y aumentar en forma correspondiente el número de opciones en la estructura if … else if. Cada una de las clases incorporadas contenía solo un método sin parámetros. Los resultados se muestran en la Tabla 2(a). Nótese que el programa no reflexivo (Ejemplo 3) se ejecutó en promedio 1.5 veces más rápido que el programa reflexivo (Ejemplo 2). Sin embargo, ambos programas mostraron un comportamiento escalable, es decir, el aumento del número de clases no aumentó significativamente su tiempo de ejecución.

***Tabla 1**. Lenguajes de programación y nivel de reflexión soportado. Construcción propia*

Lenguaje	***Nivel de reflexión soportado***	***Observación***
ActionScript 3	*Primero*	*Flash ofrece el paquete flash.utils el cual provee funciones de reflexión esenciales. Este paquete devuelve un archivo XML con información del objeto (RIA Solutions, 2012; Adobe, 2012).*
C#	*Primero*	*C# tiene de forma nativa métodos como Assembly.GetExecutingAssembly() e Invoke(), entre otros; que permiten obtener información de los objetos instanciados (Microsoft-a, 2005).*
C++	*Tercero*	*En forma similar al API BCEL de Java, se puede incorporar código a los archivos generados luego de compilado el código por medio de clases especializadas desarrolladas por el programador. También está el API Reflection y librerías como LibReflection (Achilleas, 2004) y C++ reflection (Roiser, 2005).*
Java	*Tercero*	*Se puede lograr por medio de la API BCEL (BCEL, 2013; APACHE-a, 2013; APACHE-b, 2013) la cual permite incorporar código al BYTECODE.*
Perl	*Primero*	*Perl ofrece de forma nativa métodos de reflexión esenciales (Warden, 2012).*
PHP	*Segundo*	*PHP incluye una API para el manejo de reflexión. Se puede lograr que el usuario ingrese código en tiempo de ejecución (PHP, 2013).*
PL/SQL	*Segundo*	*PL/SQL de Oracle permite ejecutar código enviado en tiempo de ejecución mediante la sentencia EXECUTE IMMEDIATE (Oracle-a, 2013; Oracle-b).*
Ruby	*Primero*	*Ruby ofrece de forma nativa métodos de reflexión esenciales (Thomas, et al., 2004).*
Smalltalk	*Tercero*	*Smalltalk tiene meta-objetos, los cuales permiten que los objetos modifiquen su código (Black, et al., 2009).*
Visual Basic	*Primero*	*Visual Basic tiene de forma nativa métodos como GetType(), que permiten obtener el tipo de las variables declaradas (Microsoft-b, 2012; Pattison, 2013).*

En la Prueba 2 se analizaron de nuevo los programas de los ejemplos 2 y 3, pero esta vez solo se consideró una clase a instanciar a la cual se le aumentó gradualmente el número de métodos posibles a invocar. Por simplicidad, cada uno de los métodos incorporados a la clase carecía de parámetros. Los resultados se muestran en la Tabla 2(b). El programa no reflexivo (Ejemplo 3) se ejecutó en promedio 1.7 veces más rápido que el programa reflexivo (Ejemplo 2) y se mantuvo la propiedad de escalabilidad en ambos programas.

En la Prueba 3 se analizaron de nuevo los programas de los ejemplos 2 y 3, pero esta vez se analizó como afectaba el número de parámetros al invocar un método. Solo se consideró una clase a instanciar con un solo método al cual se le aumentó gradualmente el número de parámetros. Los resultados se muestran en la Tabla 3(a). El programa no reflexivo (Ejemplo 3) ejecutó en promedio 2.8 veces más rápido que el programa reflexivo (Ejemplo 2) y se mantuvo de nuevo la propiedad de escalablidad en ambos programas.

Tabla 3. (a) Resultados para la Prueba 3: Invocación de un método de una clase variando el número de parámetros. (b) Resultados para la Prueba 4: Instanciamiento de una clase e invocación de uno de sus métodos. Valores en ns. Construcción propia

En la Prueba 4 se combinaron las pruebas 1 y 2, es decir, se aumentó gradualmente el número de clases y el número de métodos de cada clase y se instanció una clase y se invocó uno de sus métodos. Los resultados se muestran en la Tabla 3(b). El programa no reflexivo (Ejemplo 3) ejecutó en promedio 2.4 veces más rápido que el programa reflexivo (Ejemplo 2) y se mantuvo de nuevo la propiedad de escalabilidad en ambos programas.

Tabla 2. *(a) Resultados para la Prueba 1. Instanciamiento de una clase.*

Número posible de clases a instanciar	***1***	***20***	***40***	***60***	***80***	***100***	***Promedio***
Tiempo de ejecución programa no reflexivo (ns)	*875179*	*827368*	*849074*	*847928*	*854944*	*851472*	*850994*
Tiempo de ejecución programa reflexivo (ns)	*1357474*	*1341081*	*1415647*	*1497070*	*1466793*	*1403937*	*1413667*

Tabla 2. *(b) Resultados para la Prueba 2: Invocación de un método de una clase. Construcción propia*

Número posible de clases a instanciar	***1***	***20***	***40***	***60***	***80***	***100***	***Promedio***
Tiempo de ejecución programa no reflexivo (ns)	*868450*	*800236*	*808859*	*837310*	*834343*	*823084,5*	*828714*
Tiempo de ejecución programa reflexivo (ns)	*1468648*	*1423630*	*1424982*	*1407833*	*1416407,5*	*1446139*	*1431273*

En la Figura 2(a) se resumen los resultados de las cuatro pruebas.

La prueba 5 correspondió al Ejemplo 5 y se desarrolló en Oracle 11g mediante su lenguaje de programación PL/SQL. Se comparó el programa reflexivo del Ejemplo 5 con el no reflexivo instanciando una clase e invocando su único método. Las instrucciones del programa no reflexivo se muestran a continuación. Los resultados se muestran en la Figura 2(b).

```
CREATE OR REPLACE TYPE class1 AS OBJECT(
  id NUMBER(7),
  nombre VARCHAR(20),
  edad NUMBER(10),
  MEMBER FUNCTION getNumeroId RETURN NUMBER
);
CREATE OR REPLACE TYPE BODY class1 AS
  MEMBER FUNCTION getNumeroId RETURN NUMBER IS
  BEGIN
    RETURN id;
  END getNumeroId;
END;
DECLARE
  claseA class1;
  id NUMBER;
BEGIN
  claseA := class1(1025854,'Juan',20);
  id := claseA.getNumeroId;
  DBMS_OUTPUT.PUT_LINE(CONCAT('El ID es: ', id));
END;
```

Al igual que en las pruebas anteriores, se observó que el programa no reflexivo fue más rápido que el reflexivo.

4. Conclusiones

Con el fin de alcanzar el objetivo de desarrollar sistemas autónomos y llevarlos a la realidad, se realizó un estudio de las herramientas existentes al día de hoy que permitieran implementar un sistema autónomo. En la búsqueda, se encontró la programación reflexiva, que por sus características y teoría se asemeja al propósito de la programación autónoma, por lo que se tomó como base para la realización de sistemas autónomos, la programación reflexiva. Luego, se procedió a

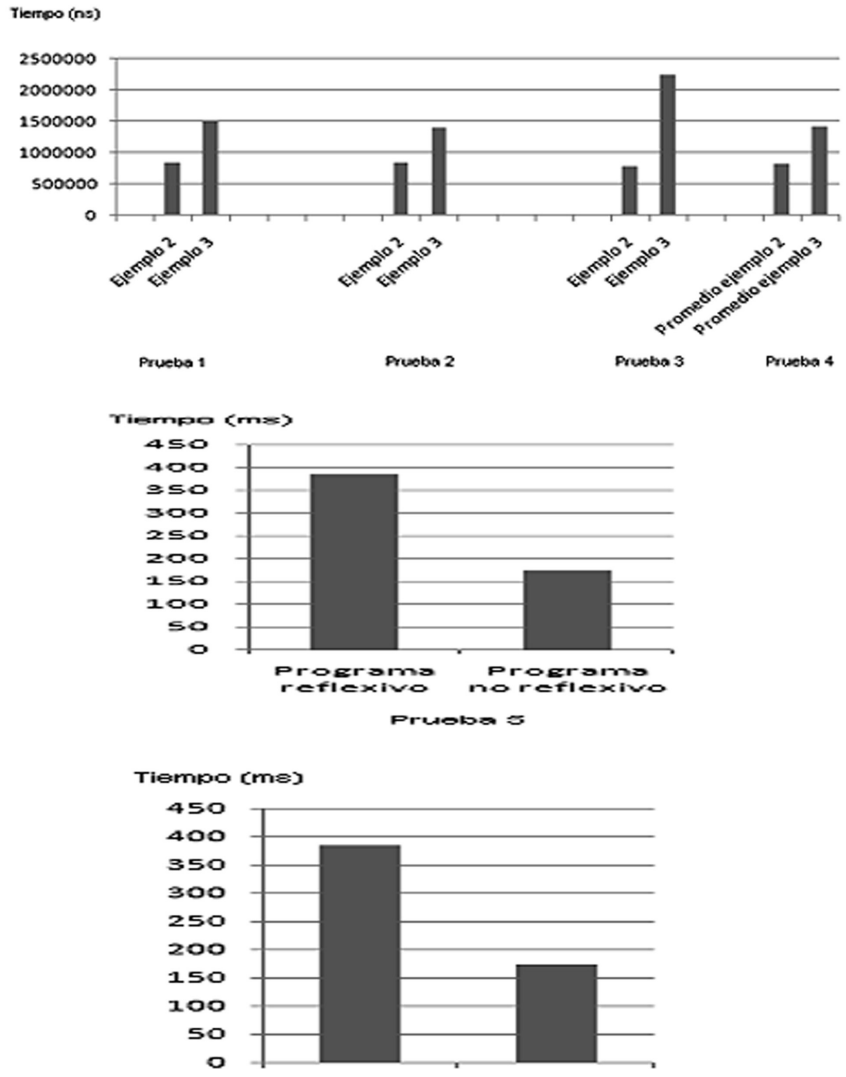

Figura 2. *(a) Resultado pruebas. (b) Resultados para la Prueba 5: Instanciamiento de una clase e invocación de su método. (c) Resultados para la Prueba 5: Instanciamiento de una clase e invocación de su método. Construcción propia*

realizar una investigación de su alcance en diversos lenguajes de programación para el momento actual. Encontrando que muchos lenguajes la implementaban, pero con restricciones y solo cierta parte de la filosofía de la reflexión. Por lo que se procedió a realizar una clasificación de la reflexión en niveles, y luego los lenguajes organizados en estos niveles, acentuando la capacidad que se tiene en la actualidad para desarrollar un sistema autónomo haciendo uso de los lenguajes de programación existentes.

Para la comprensión de cada nivel planteado, se realizaron ejemplos, de cómo se puede usar, implementar o ver reflejado tal nivel en un lenguaje y finalmente, se realizaron pruebas de rendimiento de un programa construido de forma tradicional contra un programa construido usando reflexión, en el que el resultado fue el esperado, la ejecución de la aplicación desarrollada de forma tradicional aventajo la aplicación desarrollada usando reflexión.

Sin embargo y pese al rendimiento, el ver los ejemplos y entendiendo las características que presenta la reflexión, se abren un sinfín de posibilidades en el diseño y desarrollo de software, que permiten solucionar problemas que difícilmente se pueden

resolver, si fuese posible, con el uso de programación tradicional. Pero se debe prestar especial atención a su uso indiscriminado, pues como tal es una gran herramienta, pero si es usada en problemas los cuales pueden ser solucionados con programación tradicional o su uso en el futuro del software no presenta mayor ventaja, puede convertirse en una mala práctica hacer abuso de esta, pues como se vio en los ejemplos y en las comparativas, la implementación es compleja y aumenta con el aumento de nivel, haciendo más complejo el código y difícil de mantener y su rendimiento es bajo respecto al programa no reflexivo.

Se determinó dentro del estudio realizado que dentro de los objetivos planteados por la programación reflexiva, ninguno de los lenguajes existentes permitían implementar la totalidad de las características necesarias, y que el máximo nivel alcanzado, fue el tercero. Con el fin de alcanzar la reflexión completa, se realizó un modelo y propuesta de cómo podría ser implementado el cuarto nivel de reflexión, permitiendo sobre este llevar a cabo la totalidad del desarrollo de un sistema autónomo.

Se quedan planteados como trabajos futuros, la formalización del cuarto nivel de reflexión como arquitectura sobre la que se pueda cumplir la filosofía completa de la programación reflexiva. Analizar si tal estructura se puede implementar sobre uno de los lenguajes existentes hoy en día, sea por la realización de un API, Framework o modificación del núcleo del lenguaje y que características debe cumplir para tales modificaciones, o si es necesario la implementación de un lenguaje nuevo. Y finalmente desarrollar el cuarto nivel de reflexión, con el cual se estaría muy cerca de alcanzar la programación autónoma completa. También queda el trabajo de implementar un sistema autónomo en un proceso evolutivo, haciendo uso de los recursos que se tienen hoy en día en cuanto a software y hardware y sobre el cuarto nivel de reflexión.

5. Referencias bibliográficas

Albahari, J., & Albahari, B. (2012). C# 5.0 in a Nutshell: The Definitive Reference. O'Reilly Media, Inc..

Achilleas, M. (2004). AGM::LibReflection: A reflection library for C++. - CodeProject. [Online]. Available: http://www.codeproject.com/Articles/8712/AGM-LibReflection-A-reflection-library-for-C [Accessed: 26-May-2014].

Adobe (2012). ActionScript® 3.0 Reference for the Adobe® Flash® Platform. [Online]. Available: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/utils/package.html. [Accessed: 26-May-2014].

APACHE-a. Apache Commons BCELTM - Byte Code Engineering Library (BCEL). [Online]. Available:http://commons.apache.org/bcel/manual.html. [Accessed: 26-May-2014].

APACHE-b. Overview (Commons BCEL 6.0-SNAPSHOT API). [Online]. Available: http://commons.apache.org/bcel/apidocs/index.html. [Accessed: 26-May-2014].

BCEL. BCEL API Documentation. [Online]. Available: http://bcel.sourceforge.net/docs/index.html. [Accessed: 26-May-2014].

Black, A., Ducasse, S., Nierstrasz, O., & Pollet, D. (2009). Reflection. NewYork: Square Bracket Associates Press.

c2.com. Automated Code Generation. [Online].Available:http://c2.com/cgi/wiki?AutomatedCodeGeneration. [Accessed: 26-May-2014].

Demers, F. N., & Malenfant, J. (1995). Reflection in Logic, Functional and Object-Oriented Programming. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (ICAI), Montreal, Quebec, Canada, 29-39.

Fujita, M., & Kitano, H. (1998). Development of an autonomous quadruped robot for robot entertainment. *Autonomous Robots*, 5(1), 7-18.

Horn, P. (2001). Autonomic Computing: IBM's Perspective on the State of Information Technology. New York: IBM Corporation.

JAVATM. Trail: The Reflection API (The JavaTM Tutorials). [Online]. Available: http://docs.oracle.com/javase/tutorial/reflect/index.html. [Accessed: 26-May-2014].

Kang, Z., Li, Y., & Kang, L. S. (2006). *Automatic programming methodology for program reuse.* In Proceeding of the 2006 IEEE International Conference on Computational Intelligence and Security, Guangzhou, China, 208-214.

Kant, E. (1985). Understanding and automating algorithm design. Software Engineering, *IEEE Transactions on*, 11, 1361-1374

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1), 41-50.

Lutz, M. (2013). Learning python. New York: O'Reilly Media, Inc., 983-1021.

Maes, P. (1987). Concepts and experiments in computational reflection. ACM Sigplan Notices 22(12), pp. 147-155).

Martin, J., Sánchez, J. (1994) API Reflection en los lenguajes Java y C. Tesis de Maestría en Ciencias de la Compuación e Inteligencia Artificial. Departamento de Informática y Automática. Universidad de Salamanca, España.

Massa, G. L. O., Vinuesa, H., & Lanzarini, L. (2006, October). Modular Creation of Neuronal Networks for Autonomous Robot Control. In Robotics Symposium, 2006. LARS'06. IEEE 3rd Latin American (pp. 66-73). IEEE.

Microsoft-a (2005) Reflexión (Guía de programación de C#) - MSDN [Online]. Available: http://msdn.microsoft.com/es-es/library/ms173183%28v=vs.80%29.aspx. [Accessed: 26-May-2014].

Microsoft-b (2012). Reflection (C# and Visual Basic) - MSDN [Online]. Available: http://msdn.microsoft.com/en-us/library/ms173183.aspx#Y0. [Accessed: 24-January-2013].

Oracle-a. PL/SQL User's Guide and Reference 10g Release 1 (10.1) [Online]. Available: http://docs.oracle.com/cd/B12037_01/appdev.101/b10807/13_elems017.htm. [Accessed: 26-May-2014].

Oracle-b. Oracle® Database PL/SQL Language Reference 11g Release 1 (11.1) [Online]. Available: http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/dynamic.htm. [Accessed: 26-May-2014].

Osella Massa, G. L., Vinuesa, H., & Lanzarini, L. C. (2006). Modular creation of neuronal networks for autonomous robot control. In XII Congreso Argentino de Ciencias de la Computación.

Parashar, M.; Hariri, S. (2005) Autonomic computing: An overview. In: Banâtre, J. P.; Fradet, P.; Giavitto, J. L.; Michel, O. (Ed.). Unconventional Programming Paradigms. Springer Berlin / Heidelberg.

Pattison, T. (2013). Basic Instincts: Reflection in Visual Basic .NET. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc163750.aspx [Accessed: 26-May-2014].

PHP (2013). Reflection - Manual" [Online]. Available: http://php.net/manual/en/book.reflection.php. [Accessed: 26-May-2014].

RIA Solutions (2012) Reflection en ActionScript 3. [Online]. Available: http://www.madeinflex.com/2007/09/25/reflection-en-actionscript-3/. [Accessed: 26-May-2014].

Rich, C., & Waters, R. C. (1988). Automatic programming: Myths and prospects. *Computer*, 21(8), 40-51.

Roiser, S. (2005). SEAL C++ Reflection. [Online]. Available: http://seal.web.cern.ch/seal/workbook/reflection.html. [Accessed: 24-January-2013].

Salavert, I., & Pérez, M. (2000). Ingeniería del software y bases de datos. Tendencias actuales, 28, 39-52.

Sariff, N., & Buniyamin, N. (2006, June). An overview of autonomous mobile robot

path planning algorithms. In Research and Development, 2006. SCOReD 2006. 4th Student Conference on (pp. 183-188). IEEE.

Schlossnagle, G. (2008). Advanced Php Programming: Developing Large-scale Web Applications With Php 5. Sams.

Smith, B. C. (1982). Procedural Reflection in Programming Languages. Doctoral Thesis. Department of Electrical Engineering and Computer Science. Massachusetts Institute of Technology, USA.

Sobel, J. M. & Friedman, D. P. (1996). An Introduction to Reflection-Oriented Programming. In Reflection´96 Conference, USA (pp. 1-20).

Sonntag, S., Hartig, H., Kowalski, O., Kuhnhauser, W., & Lux, W. (1994, January). Adaptability using reflection. In System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on (Vol. 2, pp. 383-392). IEEE.

Summerfield, M. (2009). Advanced Python 3 Programming Techniques. Pearson Education.

Thomas, D., Fowler, Ch. & Hunt, A. (2004) Reflection, ObjectSpace, and Distributed Ruby. Programming Ruby: The Pragmatic Programmer's Guide, 403-419.

Warden, S. (2012). Reflection. Modern Perl. Chromatic, 125-126. Onyx NEon Press.