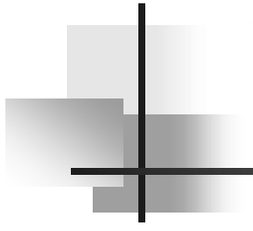
Propuesta de un modelo de reflectividad en C/C++

Iker Jamardo Zugaza
Jorge García Ochoa de Aspuru
ijamardo@eside.deusto.es



Índice

- ¿Qué es la reflectividad?
- ¿Para qué sirve la reflectividad?
- Tipos de reflectividad.
- Propiedades de un sistema reflectivo sobre una plataforma orientada a objetos.
- Reflectividad en C/C++
  - Modelos anteriores.
  - El modelo propuesto por MIRA.
- Conclusiones.
- Referencias.

¿Qué es la reflectividad?

- "***La mayor sabiduría que existe es conocerse a uno mismo***." Galileo Galilei (1564-1642).
- Definición en el ámbito computacional
  - La reflectividad es la actividad desarrollada por un sistema computacional cuando realiza computación sobre su propia computación [Maes 1987]. Esta propiedad le permite razonar y actuar sobre sí mismo y ajustar su comportamiento en función de ciertas condiciones [Ortín 2001].

¿Qué es la reflectividad?

- Sistema computacional
  - un sistema basado en computadora cuyo propósito es dar una respuesta y realizar acciones sobre un dominio en concreto. [Maes 1987]
  - Ejemplo: un programa en ejecución.

¿Qué es la reflectividad?

- En un sistema reflectivo existen dos sistemas computacionales
  - Sistema base
    - El sistema computacional en el que se ejecuta un programa.
  - Meta sistema
    - Un sistema computacional que tiene como dominio un sistema base. El meta sistema interpreta al intérprete del sistema base, es decir, el meta sistema conoce la estructura interna de un sistema computacional, de un programa [Golm 1997].

¿Qué es la reflectividad?

- Conectividad causal
  - Se produce cuando el sistema computacional y el dominio al que representa están conectados de tal forma que un cambio en cualquiera de los dos implica un cambio correspondiente en el otro.

¿Qué es la reflectividad?

- Reflexión = la capacidad de un sistema de conocerse e incluso de modificarse a sí mismo siendo esta auto-representación siempre coherente o acorde con el propio sistema.
  - Permitir a todo sistema el acceso a las estructuras que lo definen.
  - Permitir que la conexión causal entre el sistema base (programa) y el meta sistema (información reflectiva) se mantenga siempre.

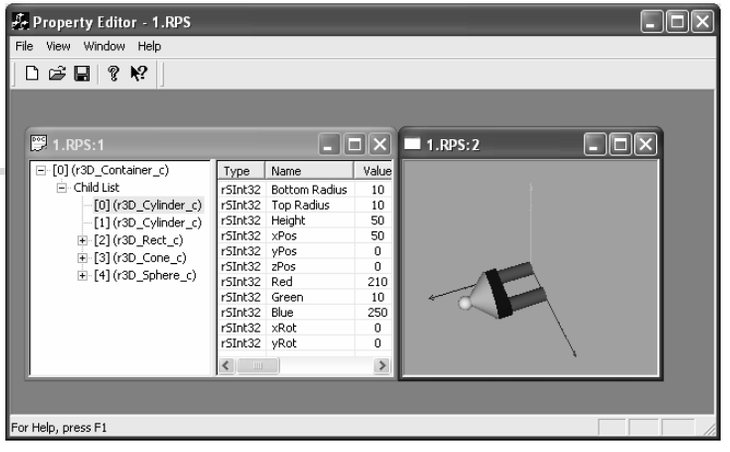
¿Para qué sirve la reflectividad?

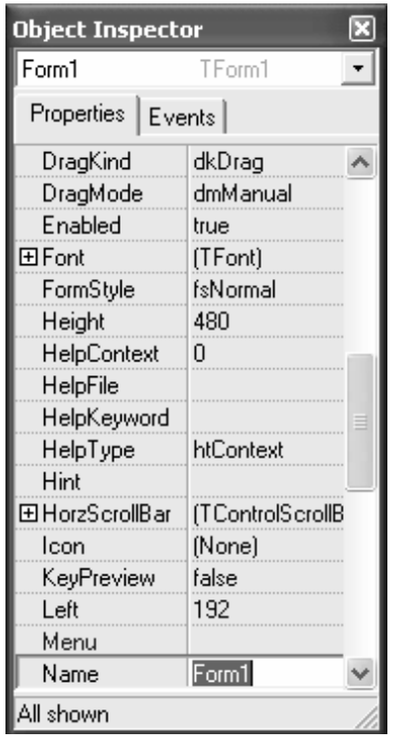
- Sistemas de serialización/persistencia.
  - La persistencia es la propiedad de poder almacenar el estado de un valor de un tipo de datos en un momento concreto para poder restaurar dicho estado posteriormente.
  - Pero no sólo eso [Barczikay 2003]:
    - Poder almacenar y restaurar las conexiones (referencias) entre valores.
    - Tener algún sistema de gestión de posibles errores y control de versiones de datos almacenados.

¿Para qué sirve la reflectividad?

- Sistemas de computación distribuida
  - En cierta manera, está ligado con la persistencia.
  - Publicación de meta información y protocolo de comunicación para acceso/ejecución.
  - Ejemplos:
    - CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture): http://www.corba.org.
    - RM**I** (**R**emote **M**ethod **I**nvocation): http://java.sun.com/products/jdk/rmi/
    - Remoting.

¿Para qué sirve la reflectividad?





- Ayuda en el desarrollo de...
  - ...depuradores en tiempo de ejecución.
  - ...plataformas de testeo.
  - ...generadores de aplicaciones.
  - ...inspectores de propiedades.
  - ...sistemas de conexiones de propiedades.
  - ...sistemas genéricos de búsqueda.

Tipos de reflectividad

- Según...
  - ... lo que se refleja:
    - Introspección.
    - Reflectividad estructural.
    - Reflectividad computacional.
    - Reflectividad lingüística.
  - ... cuándo se refleja:
    - En tiempo de compilación.
    - En tiempo de ejecución.

Tipos de reflectividad

- Según...
  - ... cómo se expresa el acceso al meta sistema:
    - Reflectividad procedural.
    - Reflectividad declarativa.
  - ... desde dónde se puede modificar el sistema:
    - Reflectividad con acceso interno.
    - Reflectividad con acceso externo.
  - ... cómo se ejecuta el sistema:
    - Ejecución interpretada.
    - Ejecución nativa.

Propiedades de un sistema reflectivo sobre una plataforma orientada a objetos.

- Propiedad 1: **Separación entre el nivel de objeto y el nivel de meta objeto.**
- Propiedad 2: **Ofrecer una representación reflectiva uniforme.**
- Propiedad 3: **Ofrecer una representación reflectiva completa.**
- Propiedad 4: **Ofrecer una representación reflectiva consistente.**
- Propiedad 5: **Las estructuras reflectivas pueden cambiarse en tiempo de ejecución.**

Reflectividad en C/C++: Modelos anteriores

- En C/C++ existe un sistema básico de reflectividad en tiempo de ejecución.
  - RTTI (**R**un-**T**ime **T**ype **I**dentification).
    - La palabra reservada del lenguaje `typeid` permite obtener un objeto de una clase denominada `tipo_info` a partir de una variable o tipo de datos.

```
class type_info
{
private:
  type_info(const type_info&);
  type_info& operator=(const type_info&);
public:
  virtual ~type_info();
  bool operator==(const type_info& rhs) const;
  bool operator!=(const type_info& rhs) const;
  int before(const type_info& rhs) const;
  const char* name() const;
  const char* raw_name() const;
};
```

Reflectividad en C/C++: Modelos anteriores

Ejemplo 1:

```
const type_info& intTypeInfo = typeid(int);
```

Ejemplo 2:

```
int i;
const type_info& iTypeInfo = typeid(i);
```

Ejemplo 3:

```
bool equals = intTypeInfo == iTypeInfo; // true
```

Reflectividad en C/C++: Modelos anteriores

- Haciendo uso de la palabra reservada del languaje `dynamic_cast<>` se puede conocer el tipo con el que se instanció un puntero.
  - Similar al `instanceof` de Java. Permite cast-s (conversiones de tipo) seguros.

```
class Parent
{
public:
  virtual ~Parent() { } // Obligatorio un método virtual.
};

class Child: public Parent
{
};

void main()
{
  Child c;
  Parent* pp = &c;
  std::cout << (dynamic_cast<Child*>(pp) != NULL) << std::endl; // true
}
```

Reflectividad en C/C++: Modelos anteriores

- Estas utilidades son claramente insuficientes a la hora de ofrecer funcionalidad reflectiva avanzada a los desarrolladores.
- Muchos desarrolladores han hecho sus propias implementaciones para solucionar problemas concretos.
- Existen distintas propuestas de propósito general desde diferentes perspectivas para conseguir distintos objetivos.
- Factor común: Hay que definir una nueva metodología/estructura propia haciendo uso del lenguaje.

Reflectividad en C/C++: Modelos anteriores

- 3 aproximaciones [Knizhnik 2003]:
  - Obtener la meta-información a partir de información de debugueo.
    - Ventajas:
      - Extracción de todo tipo de información.
      - Detección de cambios automática.
    - Desventajas:
      - El programa debe crearse con información de debugueo.
      - No hay un formato estándar para información de debugueo.
      - Complicación del proceso final:
        - Codificar.
        - Compilar con información de debugueo.
        - Parsear información de debugueo para generar estructuras reflectivas.
        - Compilar y linkar el programa final.

Reflectividad en C/C++: Modelos anteriores

- Crear un compilador especializado que genere la meta-información
  - Ventajas:
    - Simplifica la labor del desarrollador.
    - Se puede optimizar el código generado.
  - Desventajas:
    - Crear un compilador de C/C++ = tarea nada trivial.
    - Los desarrolladores prefieren usar compiladores específicos existentes.
    - La aplicación puede no podrá soportar cambios en tiempo de ejecución.

Reflectividad en C/C++: Modelos anteriores

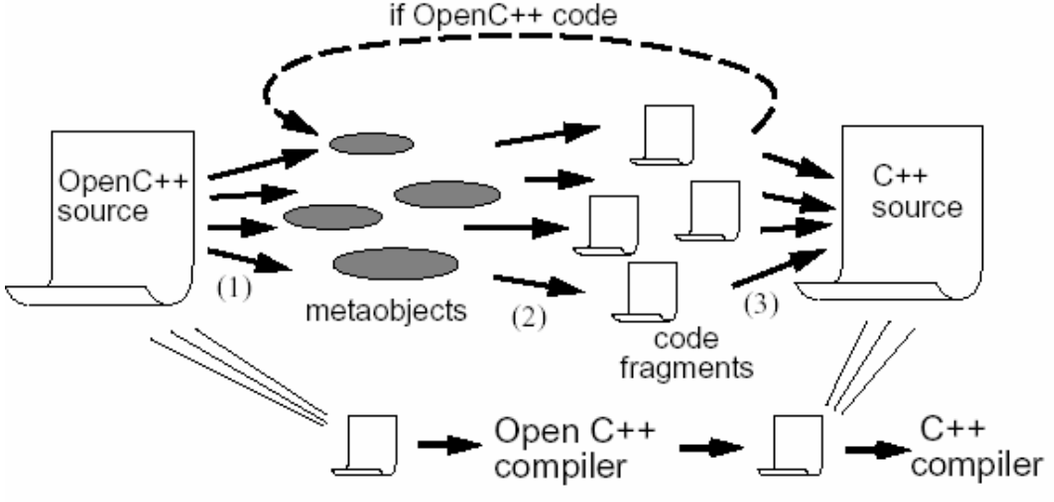
- El desarrollador aporta la meta-información
  - Ventajas:
    - La opción más portable. Se basa en estructuras del propio lenguaje que se compilan junto con el programa.
    - La opción más sencilla de implementar.
    - La opción más flexible. Se puede deshabilitar, se pueden indicar sólo los elementos que se necesiten, etc.
  - Desventajas:
    - Cambios manuales = posibilidad de error = no tiene por qué reflejar los cambios (conectividad causal).
    - Más trabajo para el desarrollador.

Reflectividad en C/C++: Modelos anteriores

- OpenC++:
  - Open C++ es un protocolo de metaobjetos (MOP **M**eta **O**bject **P**rotocol), una interfaz de programación orientada a objetos para poder personalizar el comportamiento y la implementación de los lenguajes de programación así como otros sistemas software de computación [Chiba 1995].

Reflectividad en C/C++: Modelos anteriores

- Permite que los programadores puedan implementar extensiones al lenguaje C++ como la persistencia, computación de objetos de forma distribuida o incluso optimizaciones en tiempo de compilación.



Reflectividad en C/C++: Modelos anteriores

- Metaclasses and reflection for C++ [Vollmann 2000]
  - Se define una API con la que se construye tanto el sistema base como el meta sistema.

```
ClassDef* product = new ClassDef(0, "Product");
product->addAttribute(Attribute("Product Number", Type::intT));
product->addAttribute(Attribute("Name", Type::stringT));
product->addAttribute(Attribute("Price", Type::doubleT));
product->addAttribute(Attribute("Weight", Type::doubleT));
list<Attribute> attrL;
attrL.push_back(Attribute("Author", Type::stringT));
attrL.push_back(Attribute("Title", Type::stringT));
attrL.push_back(Attribute("ISBN", Type::intT));
ClassDef* book = new ClassDef(product, “Book", attrL.begin(), attrL.end());
Object *bscpp = book->newObject();
```

Reflectividad en C/C++: Modelos anteriores

- Ventajas:
  - Sistema reflectivo en tiempo de ejecución.
  - Mentenimiento de la conectividad causal.
- Desventajas:
  - El programador debe usar la API para construir su aplicación. Vollmann propone soluciones para poder integrar el sistema con aplicaciones desarrolladas de la forma tradicional.
  - Sobrecarga en memoria y en tiempo de ejecución.
  - Él código resultante es más oscuro ya que no existe una visión directa de los tipos de datos usados.

Reflectividad en C/C++: Modelos anteriores

- Advanced Run Time Type Identification in C++ [Barczikay 2003].
  - Permite acceso tanto a datos como a métodos.
  - Diferencia entre distintos tipos de datos.
    - Primitivos.
    - Compuestos.
    - Contenedores.
  - Utilizado en la librería ROOPS (**R**eflective **O**bject **O**riented **P**roperty **S**tream).
  - Estupendos ejemplos.

Reflectividad en C/C++: Modelos anteriores

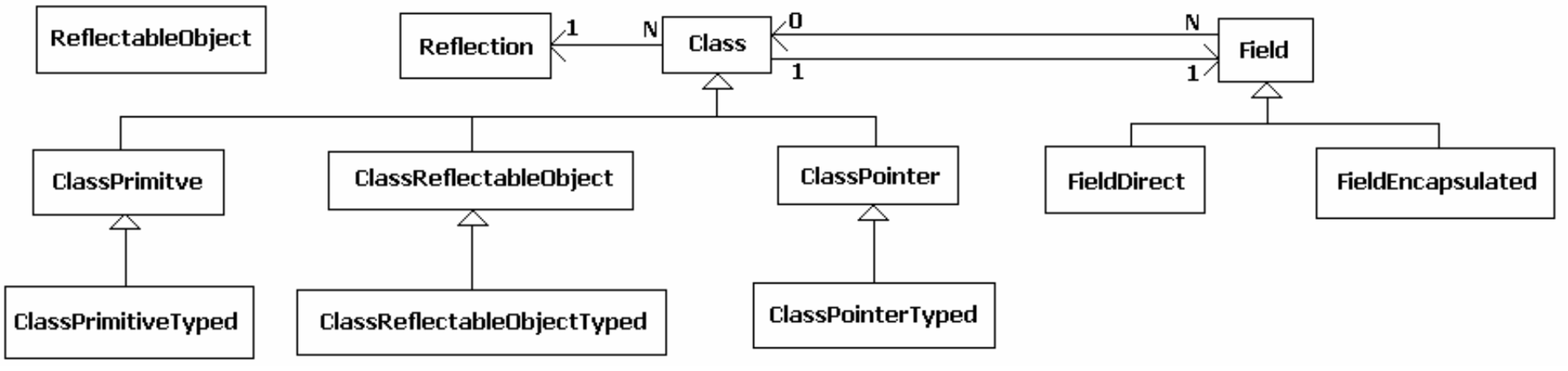
- Reflex C++:
  - Propuesta del CERN.
  - Soporte para TODAS las posibles estructuras del lenguaje.
  - Necesidad de un preproceso mediante una herramienta que parsea el código y genera la meta información (gccxml).

http://seal-reflex.web.cern.ch/seal-reflex/

Reflectividad en C/C++: El modelo propuesto por MIRA.

- Motivación
  - Sistema sencillo de aprender y usar.
  - Sistema no intrusivo:
    - El desarrollador no tiene por qué cambiar su forma de programar.
    - Se puede aplicar a programas existentes.
  - Sistema opcional.
  - Sistema parecido al de Java en estructura.
  - Aprender a diseñar e implementar un sistema reflectivo útil: (templates, punteros, especialización de tipos, uso de sistema propio de serialización, etc.)

Reflectividad en C/C++: El modelo propuesto por MIRA.



Reflectividad en C/C++: El modelo propuesto por MIRA.

- Tipo de reflectividad:
  - Según lo que se refleja: **Introspección**.
  - Según cuándo se refleja: **En tiempo de ejecución**.
  - Según cómo se expresa el acceso al meta sistema: Reflectividad **declarativa**.
  - Según desde dónde se puede modificar el sistema: Reflectividad **con acceso interno**.
  - Según cómo se ejecuta el sistema: **Ejecución nativa**.

Reflectividad en C/C++: El modelo propuesto por MIRA.

- Propiedades que cumple:
  - Propiedad 1 (separación entre el nivel de objeto y el nivel de meta objeto): **SI**
  - Propiedad 2 (ofrecer una representación reflectiva uniforme): **SI**
  - Propiedad 3 (Ofrecer una representación reflectiva completa): **SI** pero no al completo (faltan principalmente métodos).
    - Es posible conseguirlo.
  - Propiedad 4 (ofrecer una representación reflectiva consistente): **SI** pero en manos del programador.
    - Solución: Uso de precompilador.
  - Propiedad 5 (las estructuras reflectivas pueden cambiarse en tiempo de ejecución): **NO**

Conclusiones

- La reflectividad es un componente importante en las plataformas y tecnologías actuales (algunas no existirían sin ella).
- Tener en cuenta las capacidades que ofrece la reflectividad a la hora de diseñar una solución puede ofrecer flexibilidad frente al cambio.
- Las soluciones en el lenguaje C/C++ son variadas.

Referencias

- [Maes 1987] Pattie Maes. “Comcepts and Experiments in Computational Reflection”. Pattie Maes. OOPSLA ’87 Proceedings. Bruselas, Bélgica. Octubre de 1987.
- [Ortín 2001] Francisco Ortín. “Sistema Computacional de Programación Flexible diseñado sobre una Máquina Abstracta Reflectiva no Restrictiva”. PhD. Thesis. Universidad de Oviedo. Diciembre 2001.
- [Golm 1997] Michael Golm. “Design and Implementation of a Meta Architecture for Java”. Friedrich-Alexander-Universität. Computer Science Department. Erlangen-Nürnberg, Alemania. Enero de 1997.
- [Barczikay 2003] Peter Barczikay, Andras Tantos. “Advanced Run Time Type Identification in C++”. http://www.rcs.hu. Robot Control Software Ltd. 2003.
- [Chiba 1995] Shigeru Chiba. “A Metaobject Protocol for C++”. In 10th Conference on Object-Oriented Programming Systems, Languages, and Applications. OOPSLA’95. 1995.
- [Knizhnik 2003] Konstantin Knizhnik. “Reflection for C++”, http://www.garret.ru/~knizhnik/cppreflection/docs/reflect.html. 2003.
- [Vollmann 2000] Detlef Vollmann. “Metaclasses and Reflection in C++”. http://www.vollmann.com/en/pubs/meta/meta/meta.html. 2000.