DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

Implementación de Reflexión y Metaprogramación en C++ y C#

Febrero 2025

Asignatura: Programación Avanzada

Alumnos: Evans Balcázar Veizaga
Jorge B. Mostajo Pedraza
Shirley E. Pérez Delgadillo
Rubén Espinoza Orosco

Tutor Académico: PhD. Luis Roberto Pérez Ríos

Tutor Auxiliar:

Observaciones Generales	Nota

CONTENIDO

Implementación de Reflexión y Metaprogramación en C++ y C#

Evans Balcázar Veizaga
Jorge B. Mostajo Pedraza
Shirley E. Pérez Delgadillo
Rubén Espinoza Orosco

1. Introducción

En este proyecto, se implementó un sistema de reflexión y metaprogramación en C++ para la creación dinámica de clases y la invocación de métodos sin conocerlos en tiempo de compilación. La elección de C++ como lenguaje se debe a su capacidad para manejar técnicas avanzadas de metaprogramación, como plantillas (templates), SFINAE y macros del preprocesador, permitiendo una implementación eficiente sin sacrificar el rendimiento.
El paradigma orientado a objetos fue clave para estructurar la solución, mientras que la metaprogramación con templates permitió un sistema de reflexión estático y flexible. Además, se incluyeron pruebas unitarias con Google Test para garantizar la robustez del diseño.

También se implementó en C# un proyecto para la actualización automática de las versiones de sus librerías, utilizando *T4 Templates* como alternativa para la generación de código. El objetivo es generar automáticamente los archivos *Assembly.cs* durante la compilación, los cuales incorporan los metadatos de versión en una librería de clases de .NET Framework 4.8. Esto permite actualizar el cuarto dígito de la versión de la librería, siguiendo el formato `{Major version}.{Minor version}.{Hotfix version}.{MDBuild number}` (por ejemplo, `4.3.1.281`), donde *M* representa el mes y *D* el día de la compilación.

2. Descripción Técnica

Se realizó de la siguiente manera:

2.1 Generación y Transformación del Código:

La generación de código se basa en el uso de plantillas y macros para automatizar el registro de clases y métodos en una fábrica centralizada. Se utilizó una estructura de clases donde:

- **Fabrica** actúa como el registro global de clases y métodos.
- **ObjetoBase** es la clase base de todas las clases registrables.
- Se emplea **std::unordered_map** para almacenar constructores y punteros a funciones de los métodos registrados.

- Macros cómo **REGISTRAR_CLASE y REGISTRAR_METODO** facilitan el registro automático de clases y métodos sin afectar el código fuente de cada clase.

El código resultante permite que nuevas clases y métodos sean registrados sin modificar el sistema central, garantizando flexibilidad y escalabilidad.

"Fabrica.hpp"(Sistema de Reflexión)

```
#ifndef FABRICA_HPP
#define FABRICA_HPP

#include <iostream>
#include <unordered_map>
#include <functional>
#include <memory>
#include <type_traits>

// ==========================
// SISTEMA DE REGISTRO DE CLASES Y MÉTODOS CON TEMPLATES Y SFINAE
// ==========================

class ObjetoBase {
public:
    virtual ~ObjetoBase() = default;
};

// Registro global de clases y métodos
class Fabrica {
private:
    using Constructor = std::function<std::shared_ptr<ObjetoBase>()>;
    using Metodo = std::function<void(ObjetoBase*)>;

    std::unordered_map<std::string, Constructor> clases;
    std::unordered_map<std::string, Metodo> metodos;

public:
    static Fabrica& instancia() {
        static Fabrica instancia;
        return instancia;
    }

    // Registrar una clase
    template <typename T>
    void registrarClase(const std::string& nombre) {
            static_assert(std::is_base_of<ObjetoBase, T>::value, "T debe heredar de
ObjetoBase");
        clases[nombre] = []() { return std::make_shared<T>(); };
    }

    // Crear una instancia de una clase
    std::shared_ptr<ObjetoBase> crear(const std::string& nombre) {
        if (clases.find(nombre) != clases.end()) {
            return clases[nombre]();
        }
        throw std::runtime_error("Clase no registrada: " + nombre);
    }

    // Registrar un método si existe en la clase (SFINAE)
    template <typename T, typename = decltype(&T::mostrarInfo)>
    void registrarMetodo(const std::string& nombre) {
        metodos[nombre] = [](ObjetoBase* obj) {
            static_cast<T*>(obj)->mostrarInfo();
```

```
        };
    }

    // Invocar un método
    void invocarMetodo(const std::string& nombre, ObjetoBase* objeto) {
        if (metodos.find(nombre) != metodos.end()) {
            metodos[nombre](objeto);
        } else {
            throw std::runtime_error("Método no registrado: " + nombre);
        }
    }
};

// ========================
// MACROS PARA REGISTRO AUTOMÁTICO
// ========================

#define REGISTRAR_CLASE(T) Fabrica::instancia().registrarClase<T>(#T)
#define  REGISTRAR_METODO(T,  METODO)  Fabrica::instancia().registrarMetodo<T>(#T  "::"
#METODO)
#endif
```

2.2 Funcionamiento en Runtime:

La reflexión en tiempo de ejecución se logra mediante:

- **Mapas de funciones y constructores:** Se almacenan funciones lambda que permiten la creación de instancias dinámicas.
- **Uso de SFINAE:** Se registra automáticamente un método si está presente en la clase.
- **Invocación Dinámica:** Un objeto puede ser creado con Fabrica::instancia().crear("Clase") y su método invocado con Fabrica::instancia().invocarMetodo("Clase::metodo", objeto), sin conocerlos en tiempo de compilación.

Ejemplo :
"clases.hpp" (Clases de Ejemplo)

```
#ifndef CLASES_HPP
#define CLASES_HPP

#include "fabrica.hpp"

class Usuario : public ObjetoBase {
public:
    void mostrarInfo() {
        std::cout << "Soy un Usuario registrado en el sistema.\n";
    }
};

class Administrador : public ObjetoBase {
public:
    void mostrarInfo() {
        std::cout << "Soy un Administrador con permisos avanzados.\n";
    }
};
```

```
// =========================
// REGISTRO AUTOMÁTICO
// =========================

struct Registro {
    Registro() {
        REGISTRAR_CLASE(Usuario);
        REGISTRAR_CLASE(Administrador);
        REGISTRAR_METODO(Usuario, mostrarInfo);
        REGISTRAR_METODO(Administrador, mostrarInfo);
    }
} registro;

#endif
```

"main.cpp" (Ejecución Principal)

```
#include "clases.hpp"

int main() {
    try {
       // Crear objetos dinámicamente
       std::shared_ptr<ObjetoBase> usuario = Fabrica::instancia().crear("Usuario");
                                          std::shared_ptr<ObjetoBase>     admin     =
Fabrica::instancia().crear("Administrador");

        // Invocar métodos reflejados
        Fabrica::instancia().invocarMetodo("Usuario::mostrarInfo", usuario.get());
        Fabrica::instancia().invocarMetodo("Administrador::mostrarInfo", admin.get());

    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << '\n';
    }

    return 0;
}
```

Ejecución del programa Principal:

```
vficct@fedora:~/DCC_M5/A4$ ./main
Soy un Usuario registrado en el sistema.
Soy un Administrador con permisos avanzados.
```

debes instalar la librería :

```
vficct@fedora:~$ sudo dnf install gtest-devel
Actualizando y cargando repositorios:
Repositorios cargados.
Package                   Arch   Version                          Repository
Size
Installing:
 gtest-devel              x86_64   1.14.0-5.fc41                  fedora
1.0 MiB
Installing dependencies:
 gmock                    x86_64   1.14.0-5.fc41                  fedora       138.9
KiB
 gtest                    x86_64   1.14.0-5.fc41                  fedora       489.6
KiB
```

y crear el Test:

“test_fabrica.cpp” (Pruebas Unitarias)

```
#include <gtest/gtest.h>
#include "clases.hpp"

// Test: Creación de clases registradas
TEST(FabricaTest, CrearClasesRegistradas) {
    EXPECT_NO_THROW(Fabrica::instancia().crear("Usuario"));
    EXPECT_NO_THROW(Fabrica::instancia().crear("Administrador"));
}

// Test: Creación de clases no registradas
TEST(FabricaTest, CrearClaseNoRegistrada) {
    EXPECT_THROW(Fabrica::instancia().crear("Inexistente"), std::runtime_error);
}

// Test: Invocar métodos registrados
TEST(FabricaTest, InvocarMetodosRegistrados) {
    std::shared_ptr<ObjetoBase> usuario = Fabrica::instancia().crear("Usuario");
    std::shared_ptr<ObjetoBase> admin = Fabrica::instancia().crear("Administrador");

             EXPECT_NO_THROW(Fabrica::instancia().invocarMetodo("Usuario::mostrarInfo",
usuario.get()));
       EXPECT_NO_THROW(Fabrica::instancia().invocarMetodo("Administrador::mostrarInfo",
admin.get()));
}

// Test: Invocar métodos no registrados
TEST(FabricaTest, InvocarMetodoNoRegistrado) {
    std::shared_ptr<ObjetoBase> usuario = Fabrica::instancia().crear("Usuario");
                    EXPECT_THROW(Fabrica::instancia().invocarMetodo("Usuario::noExiste",
usuario.get()), std::runtime_error);
}

// Test: Registro seguro de clases y métodos
TEST(FabricaTest, RegistroSeguro) {
    // Prueba que el compilador no permita registrar clases inválidas
       // static_assert falla si intentamos registrar una clase que no herede de
ObjetoBase
    class ClaseInvalida {};

    // Esta línea generaría un error de compilación si se descomenta
    // Fabrica::instancia().registrarClase<ClaseInvalida>("ClaseInvalida");

    // Aseguramos que las clases registradas no fallen
    EXPECT_NO_THROW(REGISTRAR_CLASE(Usuario));
    EXPECT_NO_THROW(REGISTRAR_CLASE(Administrador));
    EXPECT_NO_THROW(REGISTRAR_METODO(Usuario, mostrarInfo));
    EXPECT_NO_THROW(REGISTRAR_METODO(Administrador, mostrarInfo));
}
```

Crear “CMakeLists.txt” (para compilar con Google Test)

```
cmake_minimum_required(VERSION 3.10)

project(ReflexionCXX)

set(CMAKE_CXX_STANDARD 17)

# Agregar Google Test
enable_testing()
find_package(GTest REQUIRED)
include_directories(${GTEST_INCLUDE_DIRS})
```

```
# Archivos fuente
add_executable(main main.cpp)
add_executable(test_fabrica test_fabrica.cpp)

# Link Google Test
target_link_libraries(test_fabrica GTest::GTest GTest::Main)

# Agregar pruebas
add_test(NAME FabricaTest COMMAND test_fabrica)
```

Ejecucion de Pruebas en Shell (BASH):

```
mkdir build && cd build
cmake ..
make
./test_fabrica
```

Respuestas del test:

```
vficct@fedora:~/DCC_M5/A4/build$ ./test_fabrica
Running                                    main()                                    from
/builddir/build/BUILD/gtest-1.14.0-build/googletest-1.14.0/googletest/src/gtest_main.cc
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from FabricaTest
[ RUN      ] FabricaTest.CrearClasesRegistradas
[       OK ] FabricaTest.CrearClasesRegistradas (0 ms)
[ RUN      ] FabricaTest.CrearClaseNoRegistrada
[       OK ] FabricaTest.CrearClaseNoRegistrada (0 ms)
[ RUN      ] FabricaTest.InvocarMetodosRegistrados
Soy un Usuario registrado en el sistema.
Soy un Administrador con permisos avanzados.
[       OK ] FabricaTest.InvocarMetodosRegistrados (0 ms)
[ RUN      ] FabricaTest.InvocarMetodoNoRegistrado
[       OK ] FabricaTest.InvocarMetodoNoRegistrado (0 ms)
[ RUN      ] FabricaTest.RegistroSeguro
[       OK ] FabricaTest.RegistroSeguro (0 ms)
[----------] 5 tests from FabricaTest (0 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 5 tests.
```

2.3 Generación y transformación del código en el proyecto de C#

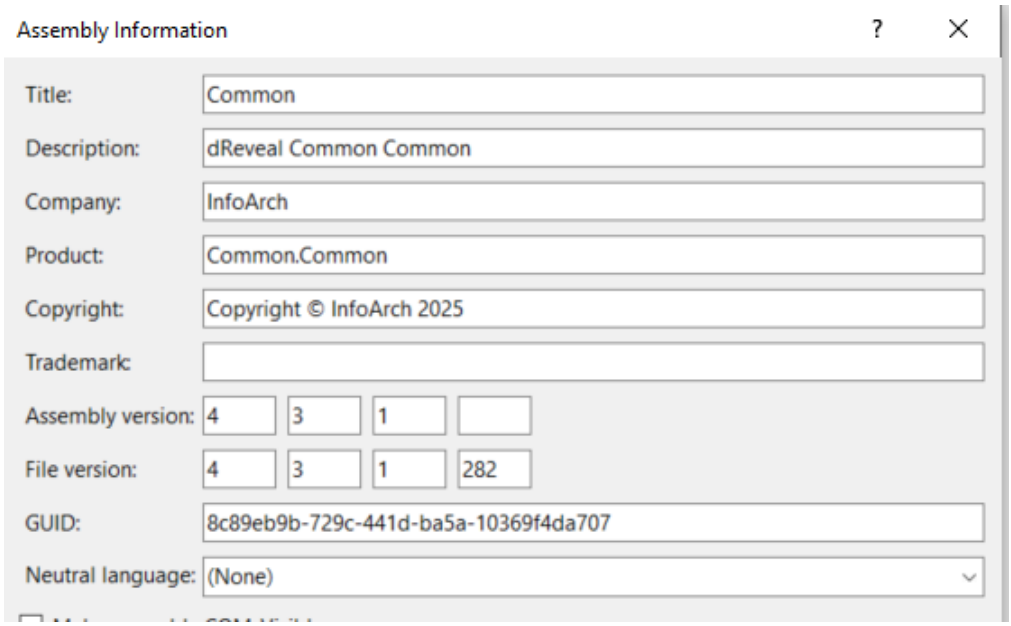
La lógica principal del **"versionado automático"** se encuentra en el archivo T4 llamado ***AssemblyTemplate.t4*** visualizado a continuación:

```
<#@ assembly name="System.Windows.Forms" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.Text.RegularExpressions" #>

<#
    int incBuild = 1;
    int incIndex = 1;

    try {
        string currentDirectory = Path.GetDirectoryName(Host.TemplateFile);
        string assemblyInfo = File.ReadAllText(Path.Combine(currentDirectory,"AssemblyInfo.cs"));
        Regex pattern = new Regex("AssemblyFileVersion\\(\"\\d+\\.\\d+\\.(?<revision>\\d+)\\.(?<build>\\d+)\"\\)");
        MatchCollection matches = pattern.Matches(assemblyInfo);
        incBuild = Convert.ToInt32(matches[0].Groups["build"].Value);
        int build2 = System.Convert.ToInt32(string.Format("{0}{1}", System.DateTime.Now.Month.ToString(), System.DateTime.Now.Day.ToString()));
        if (incBuild.ToString().Trim().StartsWith(build2.ToString().Trim()))
        {
           incIndex = System.Convert.ToInt32(incBuild.ToString().Trim().Replace(build2.ToString().Trim(), "")) + 1;
        }
        build = System.Convert.ToInt32(string.Format("{0}{1}{2}", System.DateTime.Now.Month.ToString(), System.DateTime.Now.Day.ToString(), incIndex.ToString()));
    }
    catch( Exception e)
    {
        throw new Exception(e.Message);
    };
#>
```

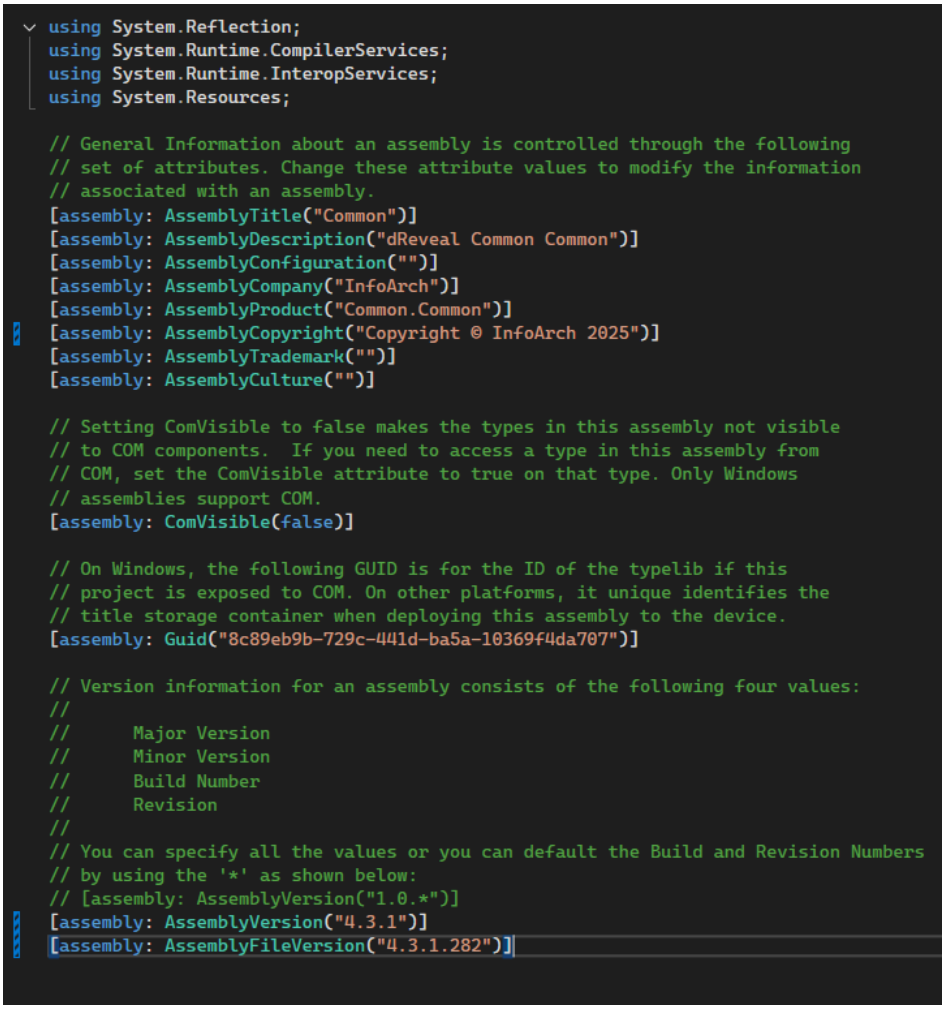
Este código es apoyado con el archivo ***Assembly.tt***(*Runtime Text Template*).

```
<#@ template debug="true" hostspecific="true" language="C#" #>
<#@ output extension=".cs" #>
<#@ assembly name="System.Windows.Forms" #>
<#@ assembly name="InfoArch.Common.Resources.v4.3.dll" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.Text.RegularExpressions" #>
<#@ import namespace="InfoArch.Common.Resources" #>
<#@ include file="..\..\..\Templates\AssemblyTemplate.t4" #>
<#+
    //RESOURCES
    string title = Resources.Common_Common_Title;
    string description = Resources.Common_Common_Description;
    string company = Resources._CurrentdRevealCompany;
    string configuration = Resources.Common_Common_Configuration;
    string copyright = Resources._CurrentdRevealCopyright;
    string product = Resources.Common_Common_Product;
    string trademark = "";
    string guid = Resources.Common_Common_Guid;
    string year = System.DateTime.Now.Year.ToString();
    int build = 0;
#>
```

Este archivo .tt se encarga de modificar los elementos del Assembly Information para su posterior liberación de la librería.



Al momento de ejecutarse mediante el evento de pre-compilación genera un nuevo archivo Assembly.cs que viene en las propiedades de los proyectos de librería de clases de .Net Framework con la nueva información de la versión.



```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Resources;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Common")]
[assembly: AssemblyDescription("dReveal Common Common")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("InfoArch")]
[assembly: AssemblyProduct("Common.Common")]
[assembly: AssemblyCopyright("Copyright © InfoArch 2025")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type. Only Windows
// assemblies support COM.
[assembly: ComVisible(false)]

// On Windows, the following GUID is for the ID of the typelib if this
// project is exposed to COM. On other platforms, it unique identifies the
// title storage container when deploying this assembly to the device.
[assembly: Guid("8c89eb9b-729c-441d-ba5a-10369f4da707")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("4.3.1")]
[assembly: AssemblyFileVersion("4.3.1.282")]
```

3. Resultados

3.1. Ventajas Obtenidas:

- **Menos código repetitivo:** Gracias a la automatización del registro de clases y métodos mediante macros y plantillas.
- **Flexibilidad:** Se pueden agregar nuevas clases y métodos sin modificar la estructura central.
- **Modularidad:** Separación clara entre la fábrica, las clases y el sistema de reflexión.
- **Pruebas unitarias robustas:** Se validó el correcto funcionamiento del sistema con Google Test.

3.2. Dificultades Encontradas:

- **Depuración:** Debido a la naturaleza dinámica de la reflexión, errores de tipo en la invocación de métodos pueden ser difíciles de rastrear.

- **Compatibilidad con Google Test:** En Fedora Linux 41, hubo problemas con la detección de GTest por parte de CMake, que fueron resueltos con configuraciones manuales.

- **Documentación:** Explicar el funcionamiento de SFINAE y macros avanzadas puede ser desafiante para nuevos desarrolladores.

3.3. Proyecto en C# - Ventajas:

- **Automatización del versionado:** Reduce la carga manual al actualizar versiones, asegurando consistencia en cada compilación.
- **Generación en tiempo de compilación:** No requiere procesos adicionales en tiempo de ejecución, lo que mejora el rendimiento y evita sobrecarga.
- **Flexibilidad en el formato de versión:** Se puede personalizar la lógica de numeración, como incluir fechas, números de compilación, etc.
- **Integración con .NET Framework 4.8:** Compatible con proyectos que aún dependen de esta versión, sin necesidad de herramientas externas.
- **Mayor control sobre los metadatos de ensamblado:** Permite definir y actualizar automáticamente los valores de AssemblyVersion y AssemblyFileVersion.

3.4. Proyecto en C# - Desventajas:

- **Curva de aprendizaje:** *T4 Templates* pueden ser complejas de configurar y depurar, especialmente para quienes no están familiarizados con su sintaxis y funcionamiento.
- **Generación de archivos adicionales:** Puede hacer que la estructura del proyecto se vuelva más compleja, con archivos generados que deben ser gestionados correctamente.
- **Dependencia del proceso de compilación:** Si hay problemas con la generación del *T4 Template*, la compilación puede fallar, afectando la entrega del software.
- **Dificultad para integración con CI/CD:** Aunque es posible automatizarlo, puede ser complicado al integrarlo con ciertos entornos de CI/CD que gestionan versiones de forma diferente.
- **Menos soporte en versiones modernas de .NET:** En .NET Core y .NET 5+, *T4 Templates* tienen menos soporte oficial, por lo que no es una solución a largo plazo para proyectos en evolución.

4. Conclusiones

El uso de reflexión y metaprogramación en C++ permitió la creación de un sistema flexible para la generación y ejecución dinámica de clases y métodos. La combinación de templates, SFINAE y macros avanzadas permitió lograr una implementación eficiente y modular.

Posibles Mejoras

- **Soporte para tipos de retorno en la reflexión:** Actualmente, los métodos reflejados son void; podría extenderse para permitir retornos dinámicos.
- **Mejor manejo de errores en runtime:** Implementar validaciones más detalladas y mensajes de error más claros.
- **Integración con serialización:** Para permitir exportar y cargar objetos reflejados desde JSON u otros formatos.

Este proyecto muestra el potencial de la metaprogramación en C++ para reducir el acoplamiento y mejorar la extensibilidad en sistemas complejos. Con ajustes adicionales, podría utilizarse en sistemas de plugins o arquitecturas orientadas a datos.

Por otro lado el proyecto en C# relacionado al versionado automático con *T4 Templates* en .NET Framework 4.8 ofrece una solución eficiente para actualizar versiones sin intervención manual, asegurando coherencia y automatización en el proceso de compilación. Sin embargo, su uso implica una curva de aprendizaje, mayor complejidad en la gestión de archivos generados y posibles dificultades en la integración con CI/CD. Aunque es una opción viable para proyectos heredados, su compatibilidad limitada con versiones modernas de .NET sugiere considerar alternativas más sostenibles a largo plazo.

5. Bibliografía

- Alexandrescu, A. (2001). *Modern C++ Design: Generic Programming and Design Patterns Applied.* Addison-Wesley.
- Meyers, S. (2014). *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14.* O'Reilly Media.
- Stroustrup, B. (2013). *The C++ Programming Language (4th Edition).* Addison-Wesley.
- Vandevoorde, D., Josuttis, N., & Gregor, D. (2018). *C++ Templates: The Complete Guide (2nd Edition).* Addison-Wesley.