# Diseño y Construcción de un Mecanismo para el uso de Invocación Implícita

MC Luis Felipe Fernández[1], MI Gabriel Bravo Martínez[2], Ing. Hada García Canale[3]

## Resumen

El principal propósito de este documento es mostrar la construcción de un mecanismo para uso de Invocación Implícita y contribuir a mejorar el diseño de software con respecto a atributos de cambiabilidad y reutilización de código. Un paradigma para mejorar el diseño de software es mantener bajo acoplamiento entre clases, y de esa manera obtener un diseño flexible. Cuando las necesidades de las compañías cambian y el software con el que cuenta tiene un diseño con un alto grado de acoplamiento, tiende a ser reemplazado por uno nuevo debido a la poca flexibilidad del diseño. Otra característica importante en el desarrollo de sistemas es la capacidad de que el software sea reutilizado. La invocación implícita es un paradigma que ha sido propuesto como una manera de lograr bajo acoplamiento y reutilización.

## Palabras Clave

Invocación Implícita, Diseño Orientado a Objetos, Bajo Acoplamiento, Alta Cohesión, Diseño de Software.

## 1. Introducción

Cuando se realiza diseño de software, generalmente hay clases implementadas con alta dependencia en otras clases, lo cual significa que si una clase necesita invocar el método de otra clase, es necesario tener visibilidad hacia dicha clase; por mencionar un tipo de visibilidad, encontramos la de tipo atributo.

Cuando el diseño de software muestra un alto acoplamiento se dificulta el mantenimiento porque cuando es necesario realizar modificaciones, las cuales pueden ser desde añadir un nuevo método a una clase o el cambio de un atributo entre otras cosas, son difíciles de llevar a cabo, ya que las clases son demasiado dependientes unas de otras, haciendo del proceso de mantenimiento una actividad compleja que requiere un alto esfuerzo en tiempo y de recurso humano.

Dos de los más importantes atributos de calidad que se abordan en este artículo son mantenimiento, en particular el de flexibilidad y la reutilización de código.

Para lograr un diseño modular y con bajo acoplamiento es posible utilizar un mecanismo el cual permita independizar las clases, y de esta manera reducir el acoplamiento. El modelo en el cual esta basado este mecanismo se conoce como Invocación Implícita (Shaw, 1996), y es utilizado como intermediario entre las clases para ayudar a minimizar la dependencia entre ellas.

Al diseñar un sistema con el uso de invocación implícita (Fernández et al., 1998) y después implementarlo como un mecanismo de invocación implícita, es posible obtener un diseño que conduzca a la mejora de atributos de mantenibilidad, además de la facilidad de reutilización de código. De esta manera podemos obtener un diseño con bajo acoplamiento y con menos complejidad al momento de la construcción.

Este documento esta distribuido de la siguiente manera. Sección 2 ofrece una clasificación de mensajes entre objetos. Sección 3 muestra la diferencia entre Invocación Implícita y Explicita. En la sección 4 serán analizados los requerimientos para el mecanismo. La sección 5 muestra el análisis y diseño del mecanismo con notación UML. Finalmente en la sección 6 se propone la construcción de este mecanismo.

[1] Universidad Autónoma de Ciudad Juárez. lfernand@uacj.mx

[2] Universidad Autónoma de Ciudad Juárez. gbravo@uacj.mx

[3] Infolink, S.A. hgarcia@infolinksa.com

## 2. Comunicación entre Objetos

En el paradigma de orientación a objetos la comunicación entre ellos es a través de mensajes, y estos mensajes pueden ser clasificados en dos formas, asíncronos y síncronos.

Cuando un objeto manda un mensaje a otro objeto y espera alguna información como respuesta es considerado un Mensaje Síncrono y cuando un objeto envía un mensaje a otro objeto sin espera de algún tipo de respuesta, es considerado un Mensaje Asíncrono. En la Figura 1 podemos ver a través de un Diagrama de Colaboración como se representa un Mensaje Síncrono.

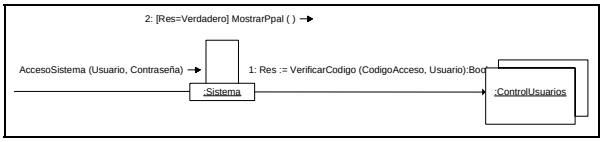


**Fig. 1 Representación de un Mensaje Síncrono.**

La representación de un mensaje Asíncrono es representada por el diagrama de colaboración, fig. 2.

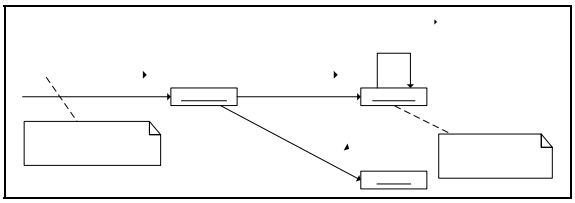

**Fig. 2 Representación de un Mensaje Asíncrono**

Los mensajes asíncronos son parte importante dentro de este documento, ya que en las siguientes secciones será usado de manera frecuente por la razón de que son la base de la propuesta para el diseño de sistemas con invocación implícita.

## 3. Invocación de Funciones

Cuando un objeto necesita que otro objeto modifique su información encapsulada o necesita el comportamiento de alguna de las funciones miembro, debe invocar alguna de esas funciones miembro. La manera en que esta función puede ser invocada puede dividirse en dos clasificaciones, Explicita e Implícita.

### 3.1 Invocación Explícita

Cuando un objeto desea que otro objeto ejecute una función, necesita mandar un mensaje indicando cual operación es la indicada y los argumentos que esta necesita, esto significa que un objeto A debe tener algún tipo de visibilidad hacia el objeto B. En la figura 3, el caso es una Visibilidad de tipo Atributo.

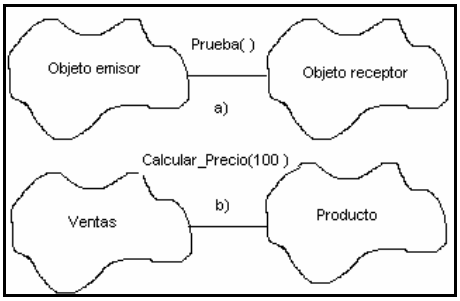


**Fig. 3 Objetos Emisor y Receptor**

En este caso, la clase Ventas debería tener una instancia de la clase Producto para tener visibilidad, justo como esta representado en el siguiente código en C (fig. 4).

```
#include "Product.cpp"

class Sales
{
    int total_sale;
    Product P1;

public:

    void Consult_Total(int x)
    {
        int quantity;
        quantity = x;
        total_sale = P1.Check_Price(quantity);
    }

};
```

**Fig. 4 Código en C representando el atributo de Visibilidad.**

Esto significa que la clase Venta tiene visible a la clase Producto por medio de crear una instancia dentro de si misma.

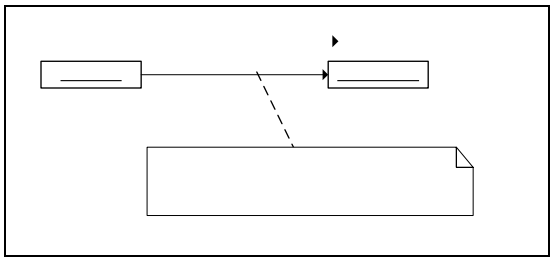

**Fig. 5 Diagrama de Colaboración en UML que representa el atributo de visibilidad entre las clases Venta y Producto.**

### 3.2 Invocación Implícita

La idea principal de la invocación implícita usada en sistemas de información es que en lugar de tener funciones invocadas explícitamente, un intermediario podría ser el que de alguna manera pueda buscar los

Pago Efectivo( )
1 Crear ( )
F1 FVenta
F2 FPago
3: Modificar (total)
Primer Mensaje Externo
S1 tota = tota

objetos e invocar esas funciones. La Figura 3 muestra a este intermediario.

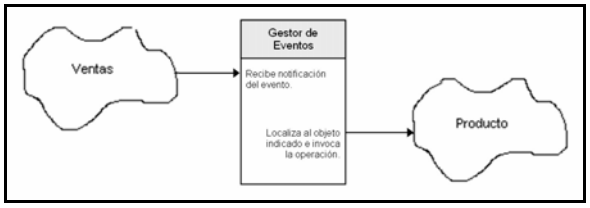


**Fig. 6 Objetos Emisor y Receptor con un Intermediario.**

La Invocación Implícita podría ser utilizada para obtener mayor independencia entre clases o módulos, y de la misma forma también tener un menor acoplamiento, y a la vez proveer un gran soporte para la reutilización de código. Además, esta podría ser considerada como un estilo de arquitectura (Shaw, op.cit.) ya que un sistema de información podría ser diseñado y modelado eliminando la interacción entre objetos lo mas posible y obtener el menor acoplamiento entre clases.

En otras palabras, Invocación Implícita pretende que un objeto A pueda ser comunicado con un objeto B sin tener la necesidad de ninguna manera de conocer B, esto significa que A no necesita tener una instancia de B para mandarle mensajes. Y el hecho de que A no necesite conocer a B implica que A no tenga dependencia con B (Dintel, 1997). Cuando una clase esta demasiado conectada o dependiente de otras clases es considerada como una clase que tiene un Alto Acoplamiento, lo cual significa que no tiene muchas posibilidades de ser reutilizada porque depende de cambios que puedan tomar efecto en otras clases, siento esto poco recomendable cuando se pretende la futura reutilización de código en su componente o clase.

## 4. Desarrollo del Mecanismo de Invocación Implícita

Se propone la implementación de un mecanismo que actué como un intermediario entre clases, así de esta manera la comunicación directa entre objetos puede ser minimizada o eliminada, el nombre de este mecanismo será Mecanismo de Invocación Implícita IIM (por sus siglas en inglés Implicit Invocation Mechanism) (Fernández et al. Op.cit.). La comunicación entre objetos será reemplazada por notificaciones de eventos.

Para definir la implementación del mecanismo es necesario hacer un análisis y reunión de conceptos que serán representados usando UML (Unified Modeling Language) (Larman, 2002), para establecer una perspectiva más amplia acerca de las funciones del mecanismo y como puede interactuar con un sistema de información.

El IIM será definido por los siguientes requerimientos y necesidades que serán implementadas. Básicamente hay tres funciones principales:

1. Los objetos, en lugar de invocar explícitamente operaciones de otros objetos, se limitan a notificar la ocurrencia de uno o más eventos al IIM.
2. Los objetos comunican al IIM su interés por un determinado evento, asociándole como mínimo una de sus operaciones.
3. El IIM acepta la comunicación del interés de un objeto por un evento y las operaciones asociadas. Además, recibe la notificación de una ocurrencia de un evento determinado e invoca las operaciones asociadas previamente al mismo.

Analicemos cuáles pueden ser las entradas y salidas del IIM, fig. 7.

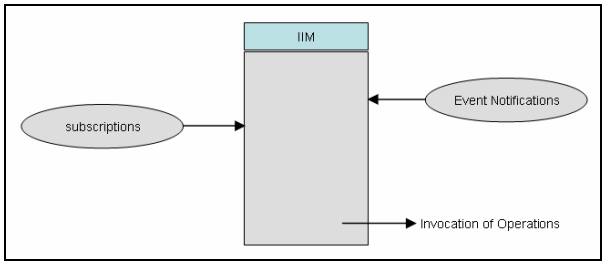


**Fig. 7 Entradas y Salidas del IIM.**

### 4.1 Eventos

Del punto de vista del IIM, un evento es cuando el usuario del mecanismo necesita que una acción sea ejecutada.

- Evento Externo

  Un evento externo proviene del exterior del sistema y requiere de una acción dentro del sistema.

- Evento Generado

  Un evento producido dentro del sistema en respuesta a un evento generado por la interfase de usuario o por otro sistema.

En caso del mecanismo, el evento externo será producido por su interfase y el sistema contenedor del mecanismo será aquel que envié el mensaje externo.

### 4.2 Notificación

Una notificación es cuando un objeto crea una instancia de tipo evento y esta es enviada al IIM. Siendo solamente un aviso de un evento que debe ser ejecutado, en lugar de mandar un mensaje directamente al objeto que tiene la función miembro, el IIM se encargará del problema.

- Notificación Síncrona

  Un evento síncrono esta esperando por una respuesta después de que la ejecución de una operación es solicitada para continuar su proceso. Cuando el IIM recibe una notificación como esta, este consulta su lista de eventos registrados y obtiene de ella el objeto interesado en el evento notificado, para invocar la operación necesaria.

- Notificación Asíncrona

  De la misma manera que la notificación síncrona, esta notificación es hecha por un objeto y es enviada al IIM informando la ocurrencia de un evento, la diferencia es que este tipo de notificación no esperara ningún tipo de respuesta para continuar su proceso.

### 4.3 Suscripción

El IIM debe de tener registrador para cada evento el objeto que esta interesado, y por lo menos asociar una de sus operaciones a dicho evento. La manera de permitir que el IIM sepa acerca de ese interés es por medio de una suscripción. En este documento solamente se trataran las subscripciones genéricas, es decir las subscripciones que envuelven la clase y por lo menos una de sus operaciones asociadas.

### 4.4 Operaciones

Una operación puede ser reconocida como una función miembro la cual es asociada a través de una subscripción cuando un objeto esta interesado en la ocurrencia de un evento. Y cuando uno de estos eventos es notificado en el IIM, este hace una búsqueda entre las subscripciones y ejecuta la operación asociada a ese evento en orden.

- Operaciones de Consulta

  Son operaciones que siempre esperaran un resultado el cual es la información necesaria, y este puede ser algún dato o un resultado, es por eso que este tipo de operaciones pueden solamente ser hechas en una forma sincronía.

- Operaciones de Actualización

  Ya que estas operaciones no esperan ningún tipo de resultado, pueden ser hechas en una forma asíncrona.

## 5. Diseño del IIM

Una vez que el análisis de los requerimientos para el componente esta finalizado, el siguiente paso es comenzar el diseño del componente. En este caso serán utilizadas las herramientas de UML, ya que es un estándar internacional que la mayoría de los diseñadores de software utiliza.

Uno de los primeros pasos es analizar bajo que ambiente de programación puede ser codificado el componente, y una de las opciones podría ser utilizar C++, ya que este lenguaje de programación ofrece una herramienta para crear componentes, aunque esta no es el único lenguaje que ofrece esta opción, es uno de los mas comunes.

La mayor dificultad que se tiene que enfrentar es que con la Invocación Implícita, una clase no necesita tener visibilidad hacia otra para ejecutar una operación, así que de esta manera se minimice el acoplamiento o dependencia entre clases.

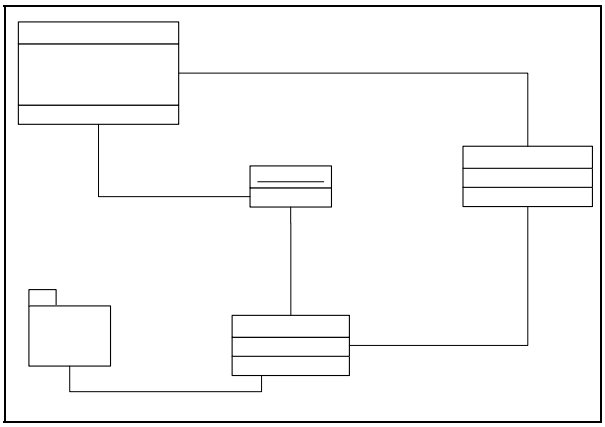

**Fig. 8 Modelo Conceptual del IIM**

En la Figura 8 el paquete “Sistema de Abarrotes” es la aplicación contenedora para el mecanismo y el primer contacto para el mecanismo es la clase Gestor de Eventos.

### 5.1 Análisis del Diagrama de Secuencias

El Diagrama de Secuencias es un artefacto que define el comportamiento del sistema, ya que este tipo de

diagramas muestra la secuencia de las funciones, viendo el sistema como una caja negra, definiendo solamente los eventos generados y recibidos por él.

Otro tipo de diagrama de secuencias, también se puede utilizar el diagrama de colaboración, el cual es considerado como de alto nivel muestra la secuencia de todas las funciones interactuando entre objetos, y este es el tipo de diagrama más favorable para mostrar la dependencia o independencia entre clases. En la fig. 9 esta un ejemplo del diagrama de secuencia de un caso de uso en un sistema, usando invocación explicita.

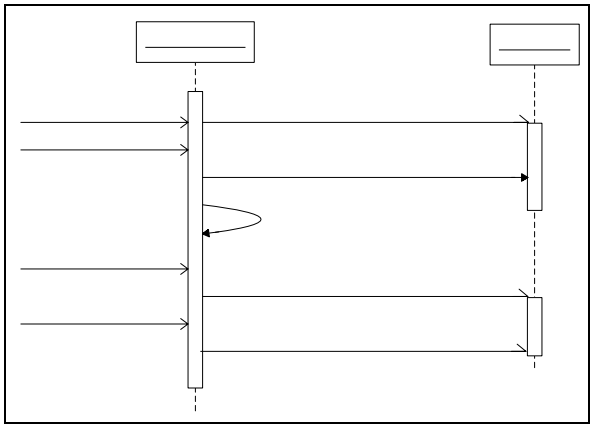

**Fig. 9 Diagrama de Secuencias con Invocación Explicita.**

En este diagrama se muestra la dependencia entre ambas clases, representada por dos objetos F1 y P1. Esto significa que las clases están altamente acopladas.

Ahora para la proposición de mejora nosotros tenemos un intermediario entre clases representado por un objeto G1, el cual es una instancia de IIM, porque el mecanismo será añadido en la aplicación contenedora como una clase. El siguiente diagrama de secuencias, fig. 10, muestra la interactividad con la instancia del IIM.

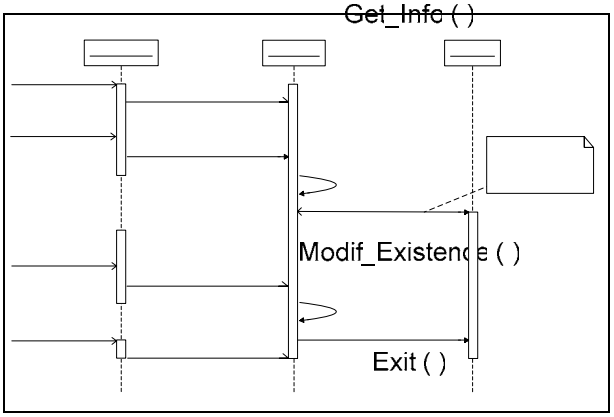


**Fig. 10 Diagrama de Secuencias con Invocación Implícita.**

En el diagrama mostrado en la figura 10 es posible ver que las clases FActualiza_Inv no tiene que tener visibilidad a la clase Producto para invocar la función Buscar_Cod(codigo), porque todo lo que ambas clases saben es que el IIM existe y que tiene eventos registrados a través de subscripciones, y solamente necesita saber que el evento "Buscar" es el que necesita llamar para obtener la información del producto para mostrar en pantalla.

Es posible que el diagrama en la Figura 10 parezca un poco más complejo, pero en realidad estamos obteniendo un nuevo diseño con un acoplamiento menor entre clases, creando un diseño menos susceptible a cambios, porque en caso de tener la necesidad de hacer un cambio a una clase, todo lo que el programador necesita hacer es modificar en dado caso la suscripción.

Para explicar de una mejor manera el procedimiento de invocar un mensaje asíncrono se muestra un diagrama de actividades en la figura 11.

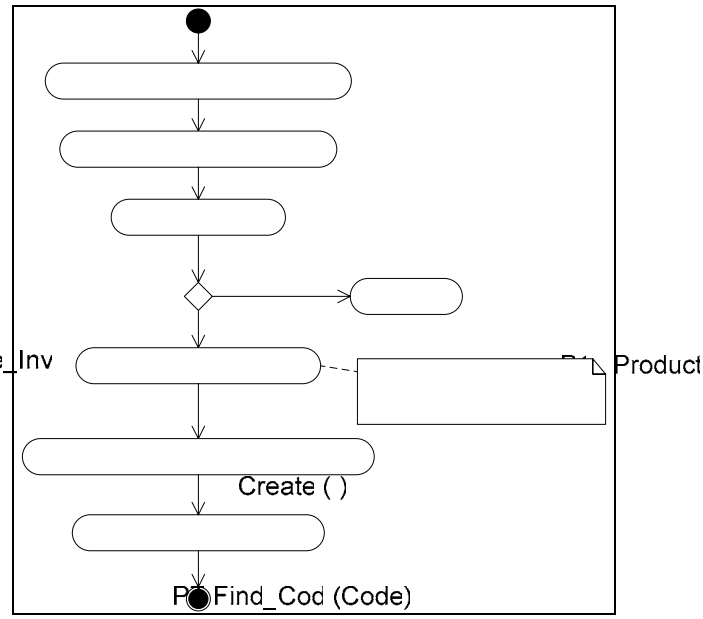


**Fig. 11 Diagrama de Actividades mostrando el procedimiento de llamar un mensaje asíncrono.**

Show_Product ( )

## 6. Construcción del IIM

El mecanismo fue codificado como un DLL en .NET porque es fácil de incrustar en la aplicación contenedora y fácil de crear una instancia o tantas como sean necesarias para manejar la Invocación Implícita en la aplicación (Microsoft, s/f).

Modif_Exist ( quantity )

Destroy ( )

## 6.1 Invocación Dinámica

Visual Studio .Net ha integrado una herramienta llamada Visor de Objetos, el cual permite la examinación interna de tipos de las librerías referenciadas.

Reflexión, hablando dentro del universo de .NET, puede ser comprendida como el proceso de reconocimiento de tipos en tiempo de ejecución, C# ofrece esta posibilidad usando los tipos definidos dentro del namespace System.Reflection, y usted puede cargar una librería en tiempo de ejecución y descubrir información acerca de ella, así como crear una instancia de ese tipo (Troelsen, 2001).

Para cargar una librería tiene que ser creada una instancia de tipo Assembly, entonces es posible obtener información tal como clases, atributos, métodos, y características así. Y también un objeto puede ser instanciado en tiempo de ejecución usando Late Binding lo cual es un aspecto importante de la interoperabilidad .NET/COM.

La dificultad a enfrentar, tal como se mencionó en la sección 5, acerca de la necesidad de que las clases no tengan visibilidad hacia otras clases para invocar sus operaciones, puede ser lograda de una manera con Invocación Dinámica, porque no es necesario que el IIM tenga todas las clases de la aplicación contenedora declaradas, porque el propósito principal es que pueda ser utilizable en todo tipo de sistemas de información, y tiene que ser general. Entonces de alguna manera el IIM necesita tener visibilidad a las clases que subscriben métodos a un evento, y permitirles conocer que una de sus operaciones ha sido requerida para invocar el método asociado.

La solución propuesta es que una clase suscriba uno de sus métodos y asociarlo a un evento, la manera de hacerlo será pasando toda su información al IIM, información tal como el nombre de la librería, nombre de clase, nombre del método y evento bajo el cual la subscripción será registrada, la codificación de la operación del IIM para llamar un mensaje asíncrono usando la Invocación Dinámica se muestra en la figura 12.

```
public void AsynchronousMsg (string Event, object[] Arg)
{
    int tam;
    size = Arg.Length;
    object[] NArg = new object[size];
    Subscripcion SubscTemp = new Subscripcion();

    NArg = Arg;

    ST = this.BuscarSusc(Evento);//Function to search in the subscription
                                 //catalog for the info associated to the
                                 //event.
        try
        {
            Assembly MyAssembly = null;
            //creating an instance to load the library
            MyAssembly = Assembly.Load(SubscTemp.NameSpace);
            //SubscTemp has the event info.
            Type CLASS = MyAssembly.GetType(SubscTemp.Clase);
            //Getting class info.
            object OBJ = Activator.CreateInstance(CLASS);
            //Creating new object using Late Binding.

            MethodInfo MyMethod = CLASS.GetMethod(ST.Metodo);
            MyMethod.Invoke(OBJ,NArg);//Invoking the new object´s method.
        }
        catch(Exception e)
        {
            Console.WriteLine(e.Message);
        }

}//End AsynchronousMsg()
```

**Fig. 12 Código para el Mensaje Asíncrono.**

El Nuevo objeto es creado usando “Late Binding”, y esto tiene que ser hecho porque el IIM no sabia exactamente cuándo un evento iba a ser solicitado, así que de esta manera al momento de que un evento ocurra inmediatamente busca a través del catálogo de suscripciones y encuentra información relacionada, entonces crea un objeto para llamar la operación asociada. Esta vez el IIM estará invocando explícitamente la operación asociada al evento.

## 7. Conclusiones

La independencia entre clases ha sido lograda utilizando invocación implícita, porque la clase “Actualiza_Inv” que antes necesitaba tener visibilidad hacia la clase “Producto” no es necesaria, ahora todo lo que ambas clases tienen que saber es que un objeto tipo IIM existe, y todas las clases necesitan mandar notificaciones hacia el IIM para invocar operaciones de manera implícita. Como trabajo futuro seria una buena idea crear un componente gráfico, para volver más manejable y fácil de incrustar en un sistema.

## Referencias


Dintel J, et al. 1997. *Towards a formal treatment of implicit invocation*. USA: Carnegie Mellon University.

http://citeseer.ist.psu.edu/cache/papers/cs/3410/http:zSzzSzreports-archive.adm.cs.cmu.eduzSzanonzSz1997zSzCMU-CS-97-153.pdf/towards-a-formal-treatment.pdf

Fernández LF, Cristina Gómez, Juan Ramón López y Antoni Olivé. 1998. *Invocación explícita vs. Invocación implícita: Análisis comparativo de dos enfoques de diseño de Sistemas de Información, III Jornadas de Ingeniería del Software*. Universidad de Murcia, España

Larman C. 2002. *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado*. Mex.: Prentice-Hall.

Microsoft ® MSDN Library. 2005. *Crear y utilizar archivos DLL de C#.* USA.

http://msdn.microsoft.com/library/spa/default.asp?url=/library/SPA/cscon/html/vclrfBuildingDynamicLinkingLibraryDLL.asp

Shaw M, Garlan D. 1996. *Software Architecture*. USA: Prentice Hall.

Troelsen A. 2001. *C# and the .NET Platform.* USA: The A Press.


Antigua Plaza de Toros de Ciudad Juárez. Foto: Betina.