

# Inyección de dependencias y contenedores livianos en .NET

Por Nicolás Paez, mayo 2006, npaez at fi.uba.ar

Esta nota presenta el patrón de Inyección de dependencias y los proyectos más relevantes de contenedores livianos disponibles en la actualidad para plataforma .NET. Adicionalmente a los aspectos técnicos de cada proyecto también se cubren las generalidades de las comunidades creadas en torno de cada uno.

## Introducción conceptual

Podríamos decir sin temor a equivocarnos, que el auge del patrón de inyección de dependencias, se produjo a partir de un libro escrito por Rod Johnson en el año 2004, titulado “J2EE Development without EJB”. Si bien la idea de inversión de control que dió origen al patrón de inyección de dependencias venía debatiéndose en la comunidad con anterioridad, fue el mencionado libro, el que de alguna forma popularizó el patrón marcando el hito fundacional. En este libro Johnson propone la utilización de contenedores livianos en conjunto con técnicas de programación orientada a aspectos para reemplazar el uso de contenedores EJB.

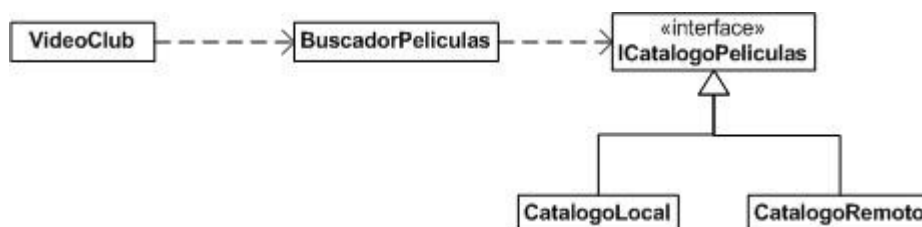


Figura 1: Planteo inicial del problema

El patrón inyección de dependencias, es una técnica que como su nombre lo indica busca facilitar la resolución de dependencias entre objetos.

Veamos un ejemplo ilustrativo para entender que es lo que se propone. Tal como muestra el diagrama de la figura 1, supongamos que tenemos una clase que tiene como responsabilidad realizar búsquedas de películas. Para poder cumplir con su cometido, cada instancia de esta clase, requiere de un objeto que implemente la interface ICatalogoPeliculas. Es aquí donde debemos comenzar a tomar decisiones, ¿Quién es el encargado de resolver esta dependencia? Una primera alternativa sería que la propia clase BuscadorPeliculas, se encargara de instanciar al catálogo, pero esto agregaría una responsabilidad extra a la clase BuscadorPeliculas incrementando al mismo tiempo el acoplamiento. Otra alternativa podría ser que sea la clase VideoClub la encargada de instanciar tanto al buscador como al catálogo y luego resolver la dependencia, el problema aquí es similar al caso anterior, pero aquí la responsabilidad adicional la estaría tomando clase VideoClub.

El patrón en cuestión propone introducir una nueva clase (Inyector<sup>1</sup>), que funcionando en forma similar a un Factory [factory], se encargue de resolver las dependencias, permitiéndonos adicionalmente separar interface de implementación.

<sup>1</sup> El nombre de esta clase no está estandarizado, cada implementación le ha dado un nombre distinto.

Si quisiéramos dar un paso más en nuestro ejemplo, podríamos plantear la existencia de varios tipos de buscadores e independizar al VideoClub del buscador específico a utilizar mediante la utilización del Inyector. Esto nos daría como resultado la situación ilustrada en la figura 2.

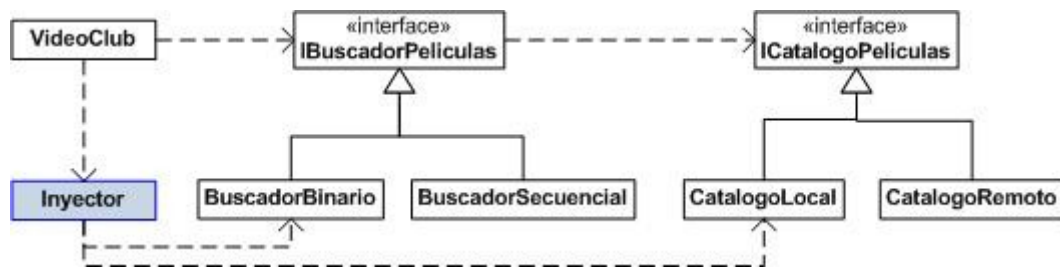


Figura 2: Solución con IoC

## Generalidades de los frameworks

Los denominados framework de inyección de dependencias, a grandes rasgos proveen alguna clase con funcionalidad análoga a la del Inyector. En general todos los frameworks ofrecen dos formas de resolución de dependencias, por constructores o por setters. En cuanto a la configuración también existen dos alternativas, por archivos de configuración (XML) o por medio de atributos. Una funcionalidad comúnmente provista es estos frameworks, es la posibilidad de personalizar el proceso de creación de instancias.

Por su parte los contenedores livianos, parten de la inyección de dependencias y ofrecen un conjunto de funcionalidades adicionales. Entre las funcionalidades más comunes ofrecidas por los contenedores livianos además de la inyección de dependencias cabe mencionar: programación orientada a aspectos, manejo del ciclo de vida de objetos, propagación de eventos e integración con otros frameworks, entre otros. Para quienes quieran ahondar en estos temas les recomiendo el artículo escrito por Martín Fowler titulado “Inversion of control container and Dependency Injection Patter” [fowler].

En las siguientes secciones se analizarán los dos proyectos de contenedores livianos más utilizados en plataforma .NET: SpringFramework y Proyecto Castle.

## Frameworks de Inyección de dependencias

Si uno busca un componente que le ayude puntualmente resolver las dependencias de su aplicación y nada más, puede optar por utilizar el ObjectBuilder. El Object Builder es un framework (o application block, según la jerga de Microsoft) de inyección de dependencias desarrollado por el Grupo de Prácticas y Patrones de Microsoft para resolver las dependencias entre los distintos componentes que conforman la Enterprise Library [entlib] y el Composite UI App. Block. Por el momento Microsoft no ha lanzado el ObjectBuilder como un application block independiente, por lo cual si uno desea utilizarlo deberá descargar alguno de los mencionados componentes que lo utilizan.

Otros frameworks de inyección de dependencias disponibles para .NET son StructureMap [structure], Pico [pico] y NFactory [nfactory].

Martín Fowler en su sitio personal escribió un artículo [fowler] en el que propuso reemplazar el término “inversión de control” por “inyección de dependencias”.

## SpringFramework

Spring es un contenedor liviano desarrollado por la empresa Interface 21. El mismo surgió originalmente en Java, donde fue muy bien recibido por la comunidad (llegando a tener cierta influencia en la especificación de EJB 3.0).

La versión .NET está basada en la versión Java y aunque aún no cuenta con toda la funcionalidad de esta última, tiene un futuro prometedor.

Spring.NET está formado por los siguientes módulos:

- ◆ Core: es la parte central del framework sobre la que están montados la mayoría de los módulos restantes. Incluye la interface `IObjectFactory`, la cual funciona como una fábrica de objetos, con la funcionalidad extra de la resolución de dependencias y algunos otros servicios. Otra interface de este módulo es `IApplicationContext`, la cual extiende `IObjectFactory`, proveyendo algunas funcionalidades adicionales para desarrollo enterprise, como ser localización de texto.
- ◆ AOP: este módulo permite hacer programación orientada a aspectos, para lo cual implementa las interfaces definidas por `AOPAlliance`. Este módulo complementa la funcionalidad del Core y representa un punto importante de extensión.
- ◆ Services: este módulo permite exponer cualquier objeto como servicio para acceso remoto vía Enterprise Services, Remoting o Web Services, sin necesidad que el objeto en cuestión deba heredar de una clase predeterminada, ni implementar interface alguna.
- ◆ Data: provee una capa de acceso a datos que permite la abstracción del proveedor de datos y ofrece manejo de transacciones declarativamente.
- ◆ ORM: provee integración con ORMs como Nhibernate [nhibernate] e iBatis [ibatis].
- ◆ Web: extiende la funcionalidad de ASP.NET proveyendo inyección de dependencias para páginas (`System.Web.Page`) y databinding bidireccional entre otros.

De todos estos módulos, actualmente solo dos han sido liberados: Core y AOP. El resto aún se encuentra en desarrollo, pero a pesar de eso ya es posible descargar versiones funcionales. Para comienzos del año próximo se prevé la liberación de la versión 1.1 de Spring.NET, la cual incluiría varios de los módulos que se encuentran actualmente en desarrollo.

Un punto para resaltar de Spring.NET, es que los módulos son lo suficientemente independientes (aunque como es de esperar casi todos los módulos dependen del Core).

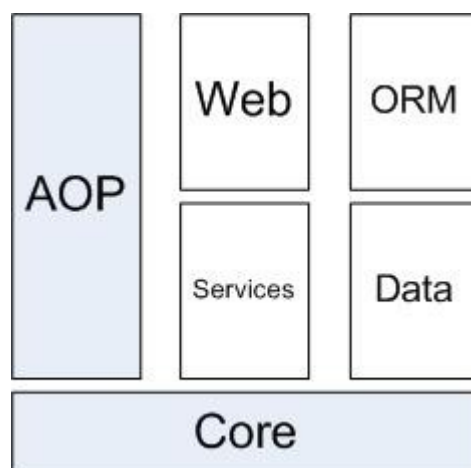


Figura 3: Componentes de Spring

## Proyecto Castle

Este proyecto nació a partir del proyecto Apache Avalon [avalon], con la intención de desarrollar un simple framework de inversión de control. A pesar de su humilde objetivo original, hoy día el proyecto ha crecido en múltiples direcciones, con la misión de generar un conjunto de herramientas que aceleren el desarrollo de aplicaciones enterprise.

Mientras que Spring es un proyecto con varios módulos, Castle consta de varios proyectos, los cuales persiguen objetivos más diversos.

Castle cuenta con un contenedor liviano llamado Windsor. Este está montado sobre un componente core, denominado Microkernel que implementa la inversión de control y que puede ser utilizado independientemente. Básicamente Windsor, enriquece el Microkernel, agregando ciertas funcionalidades y proveyendo una interface simplificada. En Microkernel provee varios puntos de extensibilidad, entre los que se destacan:

- ◆ Handlers: posibilitan la modificación del proceso de creación de instancias.
- ◆ Facilities: proveen integración con otros frameworks.
- ◆ Interceptors: permiten asociar interceptores a los métodos de los objetos creados por el contenedor.
- ◆ Entre los otros proyectos que forman parte de Castle, los más importantes son:
- ◆ Dynamic Proxy: es una librería que permite la generación de Proxies transparentes en tiempo de ejecución, de forma no intrusiva. Es actualmente utilizado por otros proyectos incluso fuera del ámbito de Castle, como ser NHibernate, Retina.NET [retina] y Ibatis.NET.
- ◆ Active Record: es una implementación del conocido patrón para persistencia de objetos, basada en NHibernate.
- ◆ Monorail: es una implementación para .NET del difundido framework Ruby On Rails [ror].



Figura 4: WindsorCastle

## Spring.NET vs. Windsor Castle

Característica	Spring	Castle
Url	<a href="http://www.springframework.net">http://www.springframework.net</a>	<a href="http://www.castleproject.org">http://www.castleproject.org</a>

Licencia	Apache Licence 2.0	Apache License 2.0
Sponsor Principal	Interface 21	Hamilton Verissimo
Estado	versión 1.0.2 (core, aop)	Release Candidate 2
Fundación (registración en sourceforge)	10/04/2004	17/11/2004
Repositorio	CVS	SVN
Documentación	En desarrollo, pero muy completa a pesar de ello. Múltiples formatos	Escasa. Online
Pruebas unitarias Build Tool Issue Tracker Foro	Si, Nunit NAnt JIRA phpBB, 694 usuarios, 2983 posts	Si, Nunit NAnt JIRA vBulletin, 1099 usuarios, 1859 posts

## Referencias

- ◆ [avalon] <http://avalon.apache.org>
- ◆ [ibatis] <http://ibatis.apache.org/index.html>
- ◆ [castle] <http://www.castleproject.org>
- ◆ [entlib] <http://msdn.microsoft.com/library/?url=/library/en-us/dnpag2/html/EntLib2.asp>
- ◆ [fowler] <http://www.martinfowler.com/articles/injection.html>
- ◆ [factory] <http://gsraj.tripod.com/design/creational/factory/factory.html>
- ◆ [jmiller] <http://codebetter.com/blogs/jeremy.miller/default.aspx>
- ◆ [nfactory] <http://www.puzzleframework.com/WikiEngine/WikiPageViewer.aspx?ID=84>
- ◆ [nhibernate] <http://www.hibernate.org/343.html>
- ◆ [pico] <http://picocontainer.org/>
- ◆ [retina] <http://www.gotdotnet.com/workspaces/workspace.aspx?id=fd081831-5a33-45cd-9f23-a828dd1f3fd1>
- ◆ [ror] <http://ap.rubyonrails.org>
- ◆ [spring] <http://www.springframework.net>
- ◆ [structure] <http://structuremap.sourceforge.net/Default.htm>