Castle Windsor

Tuna Toksoz

March 24, 2010

Ben Neyim?

- Boğaziçi Üniv. Bilgisayar Müh. 4. sınıf öğrencisi
- NHibernate ve Castle da (pasif) geliştirici
- Kisisel blogunda ve Devlicio.us'ta (pasif) yazar
- Robotik konusuna meraklı

DI Nedir?

- Martin Fowler'in makalesindeki bir patterndır.
- Nesne bağımlılıklarının dışarıdan sağlanması esasına dayanır.
- 3 alt yapıdan oluşur
  - Bağımlı
  - Bağlılık
  - Bağımlılığı sağlayan

Neden DI kullanmalıyız?

- Gevşek bağlı bileşenler
- Artan test edilebilirlik
- Çalışma zamanlı değişikliklerde kolaylık

Neden DI kullanmamalıyız?

- ...

Dependency Injection Yöntemleri

- Constructor Injection
- Property Injection
- Method Injection

Dependency Injection Yöntemleri - Örnekler

- Constructor Injection

```
public CurrentBatteryLevelStatisticsCollector(IObjectSource objectSource, IEventAggregator eventAggregator)
    : base(objectSource)
{
    this.eventAggregator = eventAggregator;
    this.batteryLevels = new Dictionary<ObjectBase, float>();
}
```

- Property Injection

```
public class BasicEnvironment
{
    public IObjectSource ObjectSource { get; set; }
}
```

- Method Injection

Inversion of Control Container

- Tüm servislerin kayıt altına alındığı ve erişilebildiği nokta.
- Bağımlılık çözümlemesini otomatik yapan yazılım bileşeni
- Yazılım geliştirmenin ilerleyen sürecinde bağımlılıkların kolay değiştirilebilmesi

Neden Castle Windsor?

- Çok kullanılan bir framework
- Aktif geliştirme
  - Ekim 2009 - Subat 2010 döneminde 118 commit.
  - 2. versiyon
- Frameworkü genişletme konusunda esneklik

Castle Windsor Konfigurasyonu

- XML Konfigurasyon
- Fluent/Programatik Konfigurasyon
- Binsor/Boo Konfigurasyonu

XML Konfigurasyonu

Dezavantajlar

- Eski
- Hataya acık

Avantajlar

- Derlemeden değişiklik yapılabilme

```
<castle>
  <components>
    <component id="HtmlTitleRetriever" type="WindsorSample.HtmlTitleRetriever, WindsorSample"/>
    <component id="StringParsingTitleScraper" service="WindsorSample.ITitleScraper, WindsorSample"
      type="WindsorSample.StringParsingTitleScraper, WindsorSample"/>
    <component id="HttpFileDownloader" service="WindsorSample.IFileDownloader, WindsorSample"
     type="WindsorSample.HttpFileDownloader, WindsorSample"/>
  </components>
</castle>
```

Fluent/Programatik Konfigurasyon

Dezavantajlar

- Derleme sonrasında değişiklik yapılması imkansız/zor

Avantajlar

- Derleme zamanlı kontrol
- Intellisense
- AllTypes Of
- Convention over Configuration

Fluent/Programatik Konfigurasyon - Cont'd

```
public void Install(IWindsorContainer container, Castle.MicroKernel.IConfigurationStore store)
{
    container
        .Register(Component.For<ICatalogService>()
            .ImplementedBy<MyCatalogService>().LifeStyle.Singleton)
        .Register(Component.For<IPriceService>()
            .ImplementedBy<PriceService>()
            .Named("priceService")
            .DependsOn(new {taxRate=0.18f})
            .OnCreate((kernel,service)=>service.Name="priceService"))
        .Register(AllTypes.Of<IConsoleCommandInterpreter>()
                    .FromAssembly(typeof(IConsoleCommandInterpreter).Assembly)
                    .WithService.FirstInterface());
}
```

Boo/Binsor ile Konfigurasyon

- Derleme/Runtime zamanlı kontrol
- Intellisense (MonoDevelop)
- Derleme sonrasında değişiklik yapılması kolay
- Boo dilinin esnekliği ile Konfigurasyon genişletilmesi

```
component mycompfactory, MyCompFactory
component mycomp, MyComp:
    createUsing @mycompfactory.Creat
```

Genişletme Noktaları

- Facility yapısı
- Eventler
- Bağımlılık Çözümü Kontrol mekanizmaları
  - Subdependency Resolver
  - Handler Selector
  - Interceptor Selector
- Yaşam döngüsü kontrol mekanizmaları
- Bileşen yaratımını kontrol mekanizmaları

Facility Yapısı

- MK/Windsor'un ayarlanmasini saglayan yapilar
- Belli bir amaca yönelik işlemlerin tümünün toplandığı yer

Mevcut Facility'ler

- Active Record Integration
- Automatic Transaction Management
- Batch Registration - Obselete
- Event Wiring
- Factory Support
- Nhibernate Integration
- Synchronize
- WCF Facility

Eventler

- ComponentRegistered
- ComponentUnregistered
- ComponentModelCreated
- ComponentCreated
- ComponentDestroyed
- DependencyResolving
- ve diğerleri

Eventler - Code

```
public class EnrichWithFacility : AbstractFacility
{
    public delegate void ExtendComponentDelegate(IKernel kernel, object instance);
    public const string ExtendWithPropertyKey = "extendwith";
    protected override void Init()
    {
        Kernel.ComponentCreated += Kernel_ComponentCreated;
    }
    void Kernel_ComponentCreated(ComponentModel model, object instance)
    {
        if (model.ExtendedProperties.Contains(ExtendWithPropertyKey))
        {
            var action = model.ExtendedProperties[ExtendWithPropertyKey] as ExtendComponentDelegate;
            action(this.Kernel, instance);
        }
    }
}
```

Bağımlılık Çözümünü Kontrol Mekanizmaları

- Subdependency Resolver
- Handler Selector
- Interceptor Selector

Subdependency Resolver

- Bir bileşenin herhangi bir bağımlılığının nasıl çözülmesi gerektiğini anlatır.
- Mevcut bir bileşen ile cevap verebilir ya da yeni bir nesne ile donebiliriz
- Daha önceden çözümlenmiş bağımlılıklarda geçerli değil (MEF?)

Subdependency Resolver - Code

```
public class ServiceIdResolver : ISubDependencyResolver
{
    #region ISubDependencyResolver Members
    public bool CanResolve(CreationContext context, ISubDependencyResolver parentResolver,
                           ComponentModel model, DependencyModel dependency)
    {
        return dependency.DependencyKey.ToLowerInvariant().Equals("serviceid") &&
               dependency.TargetType == typeof(string);

    }
    public object Resolve(CreationContext context, ISubDependencyResolver parentResolver,
                          ComponentModel model, DependencyModel dependency)
    {
        return model.Name;
    }
    #endregion
}
```

Subdependency Resolver - Code 2

```
public class ArrayResolver : ISubDependencyResolver
{
    private readonly IKernel kernel;
    public ArrayResolver(IKernel kernel)
    {
        this.kernel = kernel;
    }

    public object Resolve(CreationContext context, ISubDependencyResolver contextHandlerResolver,
                          ComponentModel model, DependencyModel dependency)
    {
        return kernel.ResolveAll(dependency.TargetType.GetElementType(), null);
    }

    public bool CanResolve(CreationContext context, ISubDependencyResolver contextHandlerResolver,
                           ComponentModel model, DependencyModel dependency)
    {
        return dependency.TargetType != null && dependency.TargetType.IsArray &&
               kernel.HasComponent(dependency.TargetType.GetElementType());
    }
}
```

Potansiyel sorun?

Handler Selector

- Bir bileşenin nasıl çözümlenmesi gerektiğini belirtir
- Mevcut duruma göre bağımlılıkların değiştirilmesini sağlar
- Daha önceden çözümlenmiş bileşenlerde geçerli değil

Handler Selector - Code

```
public class DataAccessHandlerSelector : IHandlerSelector
{
    bool databaseIsDown = false;

    public DataAccessHandlerSelector()
    {
        DatabaseMonitor.OnChangedState +=
            state => databaseIsDown = state == DatabaseState.Down;
    }

    public bool HasOpinionAbout(string key, Type service)
    {
        return databaseIsDown && service == typeof(IRepository);
    }

    public IHandler SelectHandler(string key, Type service, IHandler[] handlers)
    {
        return handlers.Where(x => x.ComponentModel.Implementation == typeof(CacheOnlyRepository)).First();
    }
}
```

Interceptor Selector/Interceptor Model Selector/IProxyGeneration Hook

- Bir bileşenle eşleştirilmiş cross-cutting concern'lerin runtime da değiştirilebilmesi
- Mevcut duruma göre bu interceptorlerin hangisinin seçileceğine karar verir
- Hangi metodların intercept edilip edilmeyeceğine karar verilebilir
- Daha önceden çözümlenmiş bağımlılıklarda geçerli değil

Yaşam Döngüsü Kontrol Mekanizmaları

Bileşenlerin ne zaman yaratılmaları gerektiğine dair karar vericidirler.

- Singleton
- PerThread
- PerWebRequest
- Transient
- Poolable
- Özel

Mevcut Yaşam Döngüleri - Singleton

```
public class SingletonLifestyleManager : AbstractLifestyleManager
{
    private volatile Object instance;

    public override void Dispose()
    {
        if (instance != null) base.Release( instance );
    }
    public override object Resolve(CreationContext context)
    {
        if (instance == null)
            lock (ComponentActivator)
                if (instance == null)
                    instance = base.Resolve(context);
        return instance;
    }
    public override bool Release(object instance)
    {
        return false;
    }
}
```

Bileşen Yaratımını Kontrol Mekanizmaları

Bileşenlerin nasıl yaratılmaları gerektiğine dair mantıpı içerirler. Castle literaturunde Activator olarak geçerler.

- Default Activator (Esas injection işinin yapıldığı Activator tipi)
- Accessor/Factory Activator (Factory Support Facility'de kullanılan Activator tipleri)

Bileşen Yaratımını Kontrol Mekanizmaları - Accessor Activator

```
public class AccessorActivator : DefaultComponentActivator
{
    public AccessorActivator(ComponentModel model, IKernel kernel,
        ComponentInstanceDelegate onCreation, ComponentInstanceDelegate onDestruction)
        : base(model, kernel, onCreation, onDestruction)
    {
    }

    protected override object Instantiate(CreationContext context)
    {
        String accessor = (String)Model.ExtendedProperties["instance.accessor"];

        PropertyInfo pi = Model.Implementation.GetProperty(accessor, BindingFlags.Public | BindingFlags.Static);

        return pi.GetValue(null, new object[0]);
    }
}
```

DI Avantajları

- Uygulamada değişiklik yapmada çeviklik kazandırır
- Artan test edilebilirlik
- Yazılımda componentler seviyesinde düşünmemizi sağlar

Windsor

- Karşılaşılan sorunlara çözüm olarak geliştirilmiş bir framework
- Çeşitli diğer frameworklerle kolay entegrasyon
- Aktif geliştirme grubu

Kaynaklar

- http://castleproject.org
- http://groups.google.com/group/castle-project-users/
- http://ayende.com