

# Specifikace ročníkového projektu Patterns4Net

Štěpán Šindelář

29. srpna 2010

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
1.1	Účel dokumentu . . . . .	3
1.2	Stručný popis . . . . .	3
1.2.1	Softwarové požadavky . . . . .	3
1.2.2	Hardwarové požadavky . . . . .	3
<b>2</b>	<b>Sada nástrojů Patterns4Net</b>	<b>4</b>
2.1	Pattern Enforcer . . . . .	4
2.1.1	Úvod . . . . .	4
2.1.2	Funkční požadavky . . . . .	4
2.1.3	Uživatelské rozhraní . . . . .	6
2.2	Architecture explorer . . . . .	6
2.2.1	Úvod . . . . .	6
2.2.2	Funkční požadavky . . . . .	7
2.2.3	Uživatelské rozhraní . . . . .	7
2.3	DocGenerator . . . . .	9
2.4	Pattern Recognizer . . . . .	9
2.5	Společné požadavky pro všechny nástroje . . . . .	10
2.5.1	Uživatelské rozhraní . . . . .	10
2.5.2	Konfigurační soubory . . . . .	10
2.5.3	Instalátor . . . . .	10
<b>3</b>	<b>Doplňující materiály</b>	<b>11</b>
3.1	Existující software . . . . .	11
3.1.1	Pattern Enforcing Compiler For Java . . . . .	11
3.1.2	NDepend . . . . .	11

# 1 Úvod

## 1.1 Účel dokumentu

Tento dokument slouží jako specifikace požadavků na ročníkový projekt s názvem Patterns4Net.

## 1.2 Stručný popis

Patterns4Net bude sada nástrojů určená pro platformu .NET usnadňující návrh a vývoj aplikací pomocí vzorů. Pojmem vzory jsou zde myšleny klasické návrhové vzory popsané v [1] a vzory popsané v [2], dále rozvinuté v [3] jako součást postupu nazvaného *Domain-driven design*.

### 1.2.1 Softwarové požadavky

Patterns4Net bude vyvíjen primárně pro Microsoft .NET Framework 4. Pro součásti s grafickým uživatelským rozhraním bude použita technologie Windows Presentation Foundation. Požadavky na cílový systém jsou následující: kromě nezbytného Microsoft .NET Frameworku 4, který implikuje OS Windows XP SP3 a vyšší, je doporučeno i Visual Studio 2010.

### 1.2.2 Hardwarové požadavky

Hardwarové požadavky se odvíjí od požadavků samotného běhového prostředí .NET 4, tedy Pentium 1 GHz nebo vyšší a alespoň 512MB RAM.

## 2 Sada nástrojů Patterns4Net

Hlavní myšlenkou, která stojí za Patterns4Net, je, že vývojář bude mít možnost přímo v kódu pomocí syntaktických prostředků přidávat k jednotlivým třídám informace o jejich vztazích a implementovaných vzorech. Tyto informace pak budou moci snadno automatizovaně zpracovat nástroje, které budou součástí Patterns4Net.

Zmíněné syntaktické prostředky jazyka C# budou především atributy. Pro zjednodušení výsledného kódu budou další možností i jmenné konvence a uživatelsky definované konvence. Uživatelsky definovaná konvence bude metoda v C#, která na základě parametru typu *System.Type* rozhodne, zda třída odpovídá dané konvenci.

### 2.1 Pattern Enforcer

#### 2.1.1 Úvod

Pomocí klíčového slova *static* u deklarace třídy v C# může programátor instruovat kompilátor, aby přeložil třídu bez chyb pokud obsahuje pouze statické metody, přitom samotné klíčové slovo *static* se pak do výsledného MSIL kódu nepromítne. Na podobném principu bude založen Pattern Enforcer, jež bude kontrolovat, zda konkrétní implementace dané třídy odpovídá vzorům, které by podle meta informací měla implementovat. Ne všechny aspekty spjaté s nějakým vzorem lze syntakticky kontrolovat, například rozdíl mezi *Bridge* a *Adapter* je pouze sémantický.

#### 2.1.2 Funkční požadavky

Pattern Enforcer bude poskytovat API, pomocí kterého bude moci i sám uživatel definovat dodatečné vzory. Jeho návrh bude vycházet z potřeb pro konfiguraci vzorů, které budou v Pattern Enforceru vestavěné. Toto API bude podobné CQL (viz sekce 3.1). Od CQL se bude lišit tím, že bude více orientované na potřeby kontrolování správné implementace vzorů a bude se jednat pouze o C# API bez jakéhokoliv grafického rozhraní.

Kontrolu Pattern Enforcerem půjde vypnout pro jednotlivé třídy nebo členy pomocí speciálního atributu, který bude mít jako povinný parametr „Message“, kde by měl programátor uvést důvod, proč se rozhodl atribut použít. Tato možnost bude dostupná proto, že implementace jednoho vzoru může mít mnoho podob a je možné, že nějaký méně obvyklý způsob by Pattern Enforcer označil za chybný, přesto může mít pro programátora smysl označit třídu odpovídající meta informací o implementovaném vzoru a to z důvodů dokumentačních.

**Vestavěné vzory.** Následuje seznam vestavěných vzorů a u každého z nich popis toho, co bude Pattern Enforcer kontrolovat. Pattern Enforcer bude kontrolovat výsledný CIL kód, takže není třeba ošetřovat speciální jazykové konstrukce jako např. automaticky implementované vlastnosti (auto-properties).

- Immutable.
  - Veškerá data, která třída nese, jsou uložena pouze v privátních členech nebo ve veřejných read-only členech.
  - Přiřazovací příkaz s privátním členem na levé straně se nesmí objevit v kódu třídy, který je dosažitelný zvenčí (přes veřejnou nebo protected metodu až na konstruktor).
  - V případě, že třída není označena jako „mělce“ imutabilní, pak: imutabilita všech tříd, jejichž instance daná třída obsahuje jako svá data.
- Null Object.
  - Třída dědí od nějaké třídy nebo implementuje nějaký interface.

- Všechny veřejné metody, které lze překrýt (override), jsou překryty.
- Překryté void metody mají prázdné tělo.
- Překryté ne-void metody obsahují pouze příkaz return s konstantní hodnotou.
- Composite. *Atribut pro Composite bude obsahovat i nepovinný parametr, kterým půjde specifikovat, jaké rozhraní daná třída „skládá“ (dále budeme používat označení „komponenta“). Pokud rozhraní není explicitně uvedeno, použije se (v tomto pořadí) rozhraní rodičovské třídy, pokud to není object, implementovaný interface, pokud je právě jeden, jinak rozhraní samotné třídy.*
  - Třída obsahuje alespoň jednu položku se seznamem (implementuje IEnumerable) dětí (objekty typu *komponenta*).
  - Všechny metody (včetně kompilátorem vygenerovaných jako jsou get a set metody) buďto v cyklu přistupují k položkám ve všech seznamech dětí, nebo jsou označeny atributem *NotComposite*.
- Template Method. *Atributu dostupný pouze pro metody.*
  - Metoda musí obsahovat volání jiných metod, které jsou virtuální nebo abstraktní.
- Simple Factory Method. *Atributu dostupný pouze pro metody.*
  - Metoda nesmí být void.
- Factory Method. *Atributu dostupný pouze pro metody.*
  - Metoda nesmí být void.
  - Metoda musí být virtuální nebo abstraktní.
- Prototype.
  - Třída musí implementovat rozhraní (interface) ICloneable.
- Singleton. *Patterns4Net bude v první verzi podporovat vzor Singleton bez dědičnosti. Zjistit, jestli má třída vždy nejvýše jednu instanci, pokud povolíme libovolný způsob dosažení tohoto cíle, nelze. Patterns4Net tedy bude podporovat nejčastější způsoby implementace vzoru Singleton v jazyce C# viz [4].*
  - Třída má privátní konstruktor.
  - Třída T obsahuje statickou členskou proměnnou typu T, která je inicializovaná ve statickém konstrukturu, nebo<sup>1</sup>
  - obsahuje privátní vložnou (nested) třídu L, jež obsahuje statickou členskou proměnnou typu T, která je inicializovaná ve statickém konstrukturu L, nebo
  - obsahuje statickou metodu, která vrací instanci typu T a ve svém těle má následující konstrukci:

```
// ...
if (instance == null)
{
    instance = new T();
}
// ...
return instance;
```

kde *instance* je privátní statická členská proměnná a volání konstrukturu T se nevyskytuje na žádném jiném místě v kódu třídy.

---

<sup>1</sup>Následující použití spojky „nebo“ má logický význam vylučovací.

- Adapter, Brigde, Proxy, Decorator. *Všechny tyto atributy budou obsahovat nepovinný parametr udávající typ rozhraní, které třída adaptuje, abstrahuje, zastupuje nebo dekoruje.*
  - Pokud je toto rozhraní uvedeno, pak třída musí obsahovat alespoň jednu datovou položku daného typu.
- Strategy.
  - Pokud se jedná o třídu, musí obsahovat právě jednu metodu, která je virtuální nebo abstraktní.
  - Pokud se jedná o rozhraní (interface), musí obsahovat právě jednu metodu.
- Visitor. *Visitor bude obsahovat povinný parametr udávající společného předka všech elementů (označme jej jako kořen), které může Visitor „navštívit“.*
  - Visitor musí obsahovat void metodu, která akceptuje jeden parametr typu object (implementace vzoru Visitor pomocí reflexe), nebo sadu void metod, které akceptují jeden parametr typu T, kde musí být jako T zastoupeny všechny podtřídy kořene (i kořen sám pokud to není interface).
  - Kořen definuje void metodu pojmenovanou jako Accept nebo opatřenou atributem VisitorAccept, která bere jako parametr Visitora. Pokud se nejedná o implementaci Visitora pomocí reflexe, pak musí každý potomek kořene překrýt tuto metodu a v jejím těle zavolat odpovídající metodu Visitora s this jako parametrem.

### 2.1.3 Uživatelské rozhraní

Pattern Enforcer bude poskytovat uživatelské rozhraní v podobě utility pro příkazovou řádku a úkolu (task) pro MSBuild. V obou případech bude možné zadat následující parametry

- Jednu nebo více .NET sestav (assembly) na prověření.
- Cestu k souboru s konfigurací.
- Cestu k souboru, do kterého bude zapsán výstup ve formátu xml.

Pokud nebude uveden výstupní soubor, vypíše program výstup v textové podobě na konzoli. Návrátová hodnota programu bude 0, pokud nenalezl v zadaných .NET sestavách (assembly) chyby v implementaci vzorů, jinak 1.

V konfiguračním souboru bude uživatel moci vypnout kontrolu specifických vzorů a bude moci uvést cestu k sestavám s vlastními vzory a rozšířeními pro Pattern Enforcer.

Výstupní soubor bude obsahovat informace o tom, jaké všechny třídy byly prověřeny a případně na jaké chyby Pattern Enforcer při kontrole narazil.

**Unit testy.** Kromě klasického uživatelského rozhraní bude Pattern Enforcer nabízet i metody vhodné pro zavolání z unit testů. Bude možné tak zavolat kompletní kontrolu přímo ze C# nebo například zkontrolovat, jestli je nějaká konkrétní třída neměnná (implementuje vzor *Immutable*).

## 2.2 Architecture explorer

### 2.2.1 Úvod

Přidané meta informace dávají možnost generovat z kódu přesnější třídní diagram a navíc rozlišovat třídy podle jejich důležitosti v kontextu celého programu. Například *Helper class* patří k třídám, které nejsou důležité, pokud chceme pochopit základní principy daného objektového návrhu na vyšší úrovni abstrakce. Na druhou stranu *Entity*, jakožto pojem z *Domain-driven design*, označuje poměrně důležitou třídu.

Architecture explorer, jak název napovídá, umožní extrahovat a zobrazit z existující .NET assembly diagram podobný třídnímu diagramu UML, který ale bude interaktivní a bude umožňovat „přibližovat“ a „oddalovat“, což bude prakticky znamenat, že při malém „přiblížení“ bude zobrazovat pouze třídy významné z hlediska vyšší abstrakce, naopak při velkém přiblížení už bude diagram zobrazovat detailnější informace.

Významnou roli mohou sehrát uživatelsky definované vzory, jejichž součástí bude i konfigurace toho, jak se mají zobrazovat v Architecture Exploreru.

### 2.2.2 Funkční požadavky

#### Další meta informace

**Dokumentace vztahů.** Kromě implementovaných vzorů bude Patterns4Net umožňovat pomocí atributů dokumentovat vztahy mezi třídami. Některé vztahy, jako například Adaptér – Adaptovaný (Adapter – Adaptee), vyplývají z uvedených vzorů, ale ne každá třída implementuje nějaký vzor. Při obecné vztahu dvou tříd, kdy jedna obsahuje referenci na druhou, není jasné, jestli se jedná o vztah kompozice, agregace nebo dokonce pouze vztah používá (uses). Patterns4Net tedy bude v takové situaci defaultně vnímat referenci jako vztah agregace, ale uživatel bude mít možnost odekorovat danou položku atributem *Composition* nebo *Uses*, a tak uvést typ vztahu explicitně. Další věc, kterou nelze odvodit bez dodatečné dokumentace, je četnost vztahů, konkrétně vztah 1:N versus N:M. Defaultní možností v tomto případě bude 1:N a vztah N:M bude možné dokumentovat atributem *ManyToMany*.

**Vrstvy.** Architecture Explorer by měl umožnit zobrazit třídní diagram na různých úrovních abstrakce, proto bude podporovat koncept *vrstev*, což je další úroveň balíčku (packages) z třídních diagramů UML. Uživatel bude mít možnost pomocí atributu na úrovni sestavy (assembly), označit danou sestavu nebo více sestav jako *vrstvu*.

#### Vstupy a výstupy

Vstupem Architecture Exploreru bude jedna nebo více .NET sestav (assembly). Architecture Explorer ze vstupních sestav vyextrahuje meta informace o třídách, implementovaných vzorech, vztazích mezi třídami a vrstvách. Dále ze vstupních sestav získá informace o vztahu „používá“.

Třída A používá třídu B, pokud se někde v kódu metod třídy A objeví volání statického členu třídy B nebo třída A obsahuje metodu, jejíž parametrem je třída A, nebo třída A instancuje B pomocí operátoru *new*. Z toho plyne, že Architecture Explorer nemusí umět odhalit vztah „používá“, pokud je skryt ve volání pomocí reflexe (reflection) nebo pomocí datového typu *dynamic*.

Dalším možným vstupem pro Architecture Explorer bude Visual Studio 2010 solution (soubor \*.sln). V takovém případě Architecture Explorer z daného souboru získá seznam projektů a u každého projektu zjistí lokaci výsledné sestavy (assembly). Všechny tyto sestavy potom Architecture Explorer načte.

### 2.2.3 Uživatelské rozhraní

Architecture Explorer bude program s grafickým uživatelským rozhraním. Bude se jednat o tzv. „jedno-okenní“ aplikaci s nástrojovou lištou ve stylu office 2007 (tzv. ribbon) a postranními panely pro zobrazení dodatečných informací, které bude možné skrýt nebo libovolně přemístit (dockovat). Hlavním obsahem aplikačního okna bude vygenerovaný diagram.

#### Výsledný diagram

Výsledný diagram bude umožňovat měnit úroveň abstrakce („přibližovat/oddalovat“) a podle úrovně abstrakce bude zobrazovat všechny nebo pouze některé třídy. U zobrazených tříd potom všechny, pouze některé nebo žádné členy. Pokud budou v diagramu zobrazeny nějaké dvě třídy, které jsou ve vztahu, pak bude jejich vztah vždy zobrazen.

Architecture Explorer bude zobrazovat vztahy dědičnosti, implementace, agregace, kompozice a používá (uses) a u vzorů (vestavěné vzory nebo uživatelsky definované), které obsahují nějaký pojmenovaný vztah (např. *Composite* – *Component*), bude tento vztah nějakým způsobem graficky odlišovat. Ve většině případů pouze pomocí poznámky u spojení znázorňující vztah.

U tříd, které implementují nějaký vzor, bude tento fakt také nějakým způsobem graficky znázorněn. Opět ve většině případů pouze pomocí textové poznámky vedle jména třídy.

**Terminologie.** Třídy implementující vzory *Entity*, *AggregateRoot* nebo *ValueObject* z *Domain-driven design* označíme jako třídy 0. úrovně. Třídy implementující vzory pomocného charakteru jako je *Helper Class*, *Null Object* a pod. označíme jako třídy 2. úrovně. Ostatní třídy jsou 1. úrovně. Uživatel bude mít možnost definovat si vlastní vzory a nastavit jim libovolnou úroveň. Definice úrovně vestavěných vzorů bude načítána z konfiguračního souboru ve formátu xml. Pojmem element je myšlena třída, libovolný člen třídy, balíček nebo vrstva.

**Úrovně abstrakce.** Úrovně abstrakce, které bude interaktivní diagram nabízet, jsou následující:

- Nejvyšší úroveň – diagram zobrazuje vrstvy jako čtverce, v každé vrstvě potom balíčky (odpovídají jmenným prostorům). Pokud ve vrstvě A existuje třída která je ve vztahu s třídou z vrstvy B, pak je tento vztah znázorněn mezi vrstvami A a B.
- Úroveň vrstvy – diagram zobrazuje elementy z vybrané vrstvy a to třídy 0. úrovně, pokud v dané vrstvě existuje alespoň jedna taková třída, nebo třídy 1. úrovně v opačném případě. U tříd jsou zobrazeny pouze názvy, členy nejsou zobrazeny. Dále jsou zobrazeny balíčky jako čtverce, ve kterých jsou umístěny třídy. Pokud má nějaká zobrazená třída A vztah s třídou B v jiné vrstvě, pak je tento vztah znázorněn, ale graficky odlišen.
- Úroveň balíčku – diagram zobrazuje elementy z vybraného balíčku a to třídy 0. a 1. úrovně, pokud existují třídy 0. úrovně, nebo třídy 1. a 2. úrovně, potom ovšem u tříd 2. úrovně nezobrazuje členy, ale pouze název třídy. Pokud má nějaká zobrazená třída A vztah s třídou B v jiném balíčku nebo vrstvě, pak je tento vztah znázorněn, ale graficky odlišen.
- Úroveň třídy – diagram zobrazuje vybranou třídu, všechny její členy a všechny třídy se kterými je ve vztahu. U těchto tříd zobrazuje pouze členy, které se vztahu nějak účastní.

## Postranní panely s doplňujícími informacemi

Architecture Explorer bude poskytovat následující panely:

- Hierarchický seznam všech elementů v diagramu.
- Pro aktuální element (pouze pokud je to třída nebo člen třídy):
  - Dokumentační komentáře extrahované z kódu.
  - Dokumentace implementovaného vzoru (případně více vzorů), pokud je daný element součástí implementace nějakého vzoru.
  - (Pouze pro třídy.) Detailní seznam všech členů.

## Ovládání



**Výběr aktuálního elementu.** Aktuální element půjde vybrat kliknutím na grafický prvek, který jej znázorňuje, nebo kliknutím či označením odpovídajícího uzlu v postranním panelu s hierarchickým seznamem.

**Zobrazení detailu elementu.** Zobrazení detailu elementu (diagram bude zobrazen v odpovídající míře astrakce) bude možné provést dvojklikem na grafický prvek, který jej znázorňuje, dvojklikem na odpovídající uzlu v postranním panelu s hierarchickým seznamem nebo stisknutím klávesy enter, když je daný element *aktuálním elementem*.

**Nástrojová lišta.** Nástrojová lišta bude poskytovat ovládací prvky pro následující akce:

- Načíst Visual Studio 2010 solution.
- Začít nový diagram.
- Načíst sestavu (assembly).
- Vybrat aktuální vrstvu.
- O úroveň výš (v aktuálním diagramu).
- Spustit Pattern Recognizer (viz 2.4).
- Spustit Pattern Enforcer (viz 2.1).
- Spustit DocGenerator (viz 2.3).
- Nápověda – o programu.
- Nápověda – uživatelská.

## 2.3 DocGenerator

Jazyk C# podporuje tzv. dokumentační komentáře, pomocí kterých může programátor přímo v kódu vytvářet dokumentaci k jednotlivým elementům jako například třídám nebo metodám. Proces generování výsledné API dokumentace ve formátu html nebo jiných formátech funguje tak, že C# kompilátor vygeneruje xml soubor, který obsahuje API dokumentaci, a ten pak lze snadno převést na požadovaný formát pomocí xslt nebo jiných specializovaných nástrojů jako je Microsoft Sandcastle.

Patterns4Net bude poskytovat nástroj nazvaný DocGenerator, který podle Patterns4Net meta informací přidá informace o implementovaných vzorech do kompilátorem vygenerovaného xml souboru s API dokumentací.

DocGenerator bude utilita pro příkazovou řádku. Vstupem bude cesta k souboru s xml komentáři a cesta k odpovídající .NET sestavě (assembly) s meta daty. Výstupem programu bude modifikovaný soubor s xml komentáři.

## 2.4 Pattern Recognizer

Protože nástroj Pattern Enforcer bude schopen ověřovat, zda třída implementuje nějaký konkrétní vzor, bude možné jej jednoduše použít i na rozeznání vzorů. Výsledky potom zobrazí Architecture Explorer společně s informací o tom, na kolik procent je pravděpodobné, že daná třída opravdu implementuje rozeznávaný vzor.

Algoritmus rozeznávání vzorů bude jednoduchý, cílem je využít již existující nástroje Pattern Enforcer a Architecture Explorer. Rozeznávání bude fungovat tak, že pro každou třídu a pro každý vzor zkusí Pattern Recognizer zavolat Pattern Enforcer a podle počtu chyb určí procentuální pravděpodobnost vztahu mezi prověřovanou třídou a vzorem.

Pattern Recognized bude podporovat vzory *Immutable*, *Template Method*, *Singleton*, *Composite* a *Null Object*.

## **2.5 Společné požadavky pro všechny nástroje**

### **2.5.1 Uživatelské rozhraní**

Uživatelské rozhraní a uživatelská dokumentace pro všechny nástroje, které budou součástí Patterns4Net, budou v anglickém jazyce.

### **2.5.2 Konfigurační soubory**

Pokud nějaký nástroj Patterns4Net bude používat konfigurační soubor, musí být schopen fungovat, i když daný soubor nebude existovat – doplní si sám defaultní konfigurační hodnoty.

### **2.5.3 Instalátor**

Patterns4Net jako celek bude poskytovat instalační soubor, který nainstaluje všechny nástroje Patterns4Net, potřebný software třetích stran, nápovědu a ukázkové projekty.

## 3 Doplnující materiály

### 3.1 Existující software

#### 3.1.1 Pattern Enforcing Compiler For Java

Pattern Enforcing Compiler (PEC) For Java [5] je software velice podobný nástroji Pattern Enforcer. PEC umožňuje kontrolovat správnou implementaci vzorů, místo javovských anotací, používá prázdné rozhraní (interface, tzv. Marker interface pattern), jako jeden z důvodů pro tento postup uvádí autor fakt, že implementované rozhraní (interface) se zobrazuje ve vygenerované API dokumentaci. Kromě ověření správné implementace vzorů umožňuje PEC i generování kódu, například umí vygenerovat rekurzivní volání v metodách pro třídu implementující vzor *Composite*.

#### 3.1.2 NDepend

NDepend [6] je nástroj pro analýzu .NET sestav (assembly), umožňuje generovat různé diagramy jako je matice závislostí, počítat metriky jako je cyklomatická složitost, unikátní funkcionalitou NDepend je tzv. Code Query Language (CQL), který umožňuje pokládat dotazy typu „zobraz všechny metody, jejichž kód je delší jak X řádků nebo jejich cyklomatická složitost je větší jak Y“ nebo „zobraz metody, které používají třídu Z“.

## Reference

- [1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.
- [2] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.
- [3] E. Evans, *Domain-driven design: tackling complexity in the heart of software*. Longman, 2004.
- [4] J. Skeet, “C# in depth: Implementing the singleton pattern.” <http://csharpindepth.com/Articles/General/Singleton.aspx>, July 2010.
- [5] H. Lovatt, “Pattern enforcing compiler for java.” <https://pec.dev.java.net/>, May 2010.
- [6] P. Smacchia, “Ndepend.” <http://www.ndepend.com/Default.aspx>, July 2010.