File Manager – Programátorská dokumentace

David Burian

Obsah

1 Popis aplikace

File Manager je okenní Windows Forms aplikace napsaná v programovacím jazyce C# pro .NET platformu verze 4.7.2. Je to minimalistická aplikace s jednoduchým designem. Její cíl je být rychlý pomocník při manipulaci s NTFS file systémem. K dosažení co největší efektivity používá více vláken.

Aplikace je složena ze dvou hlavních částí: z okenní aplikace WinForms obsahující všechny GUI prvky a s nimi spjatou logiku a z knihovny, která vykonává a řídí operace nad file systémem.

2 Co aplikace umí vykonat

- ***Zobrazovat obsah složek spolu s dalšími užitečnými informacemi***

  Aplikace uživateli zobrazuje obsah adresářů, na které má čtecí práva. Umožňuje uživateli označovat více souborů a používat je jako operandy v různých operacích.

- ***Reagovat na klávesový vstup***

  Program zachytává všechny stisknuté klávesy a dokáže na ně reagovat. Vnitřní logika aplikace přesně řídí, která část aplikace bude událost stisknuté klávesy zpracovávat.

- ***Měnit vzhled aplikace***

  Aplikace má modulární design, který jí umožňuje měnit rozpoložení vnitřních částí okna za běhu programu.

- ***Zpracovávat příkazy od uživatele***

  File Manager disponuje příkazovým řádkem, který umožňuje uživatelům zadat různé příkazy z předdefinované množiny operací.

- ***Asynchronně vykonávat operace***

  Aplikace je navržená tak, aby dokázala zpracovávat příkaz od uživatele a i přesto zůstávala responzivní. Zatímco jedno vlákno vykonává uživatelem zadanou operaci, druhé, hlavní vlákno, ovládá GUI a vykonává další uživatelovy žádosti.

- ***Sledovat status operací napříč více vláken***

  Program navíc dokáže zobrazovat průběh operace v reálném čase. Tedy synchronizuje více vláken a dokáže uživatelovi zobrazit aktuální stav operací.

- ***Rušit operace napříč více vláken***

  File Manager umožňuje uživateli rušit operace, které čekají na spuštění i které jsou již v běhu. Navíc je zaručena správná dealokace zdrojů i při náhlém skončení aplikace.

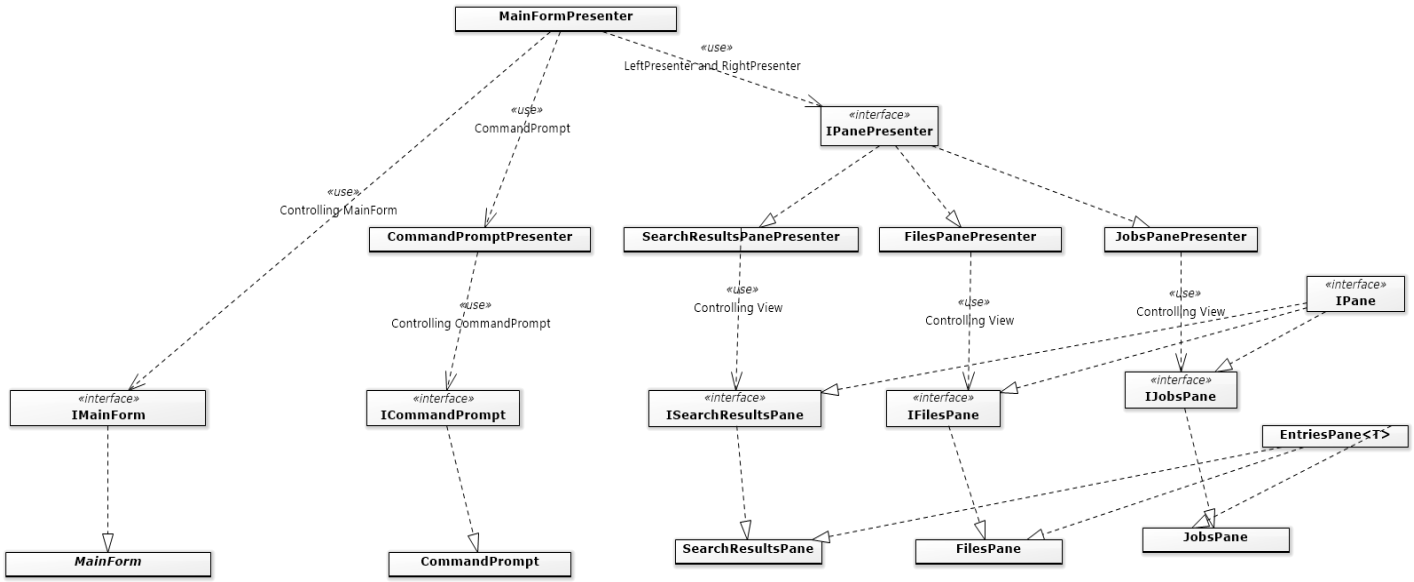
3 Architektura programu

Aplikace je rozdělena do dvou hlavních částí. První část – FileManager – se stará o vykreslení grafických prvků uživateli, o jejich správné fungování a zpracování uživatelem vyvolaných událostí. Druhá část – MultithreadedFileSystemOperations (zkráceně MFSO) – vykonává operace nad file systémem, stará se o jejich synchronizaci a poskytuje API na jednoduchou synchronizaci stavů operací.

3.1 FileManager

Návrh FileManageru kopíruje designový vzor Model-View-Presenter[1]. Implementuje si navíc své vlastní kontrolní prvky. Výsledná architektura je tak rozřazená do dvou částí, kdy v jedné leží pouze hloupé grafické komponenty (views) a v druhé leží kód, který ovládá celou aplikaci (presenter). Model pak představuje samotný file systém, v některých případech knihovna.

FileManager je jedno-okenní aplikace jejíž hlavní okno (MainForm) je rozdělené na tři části. Úplně dole, po celé šířce okna leží příkazový řádek (CommandPrompt). Nad ním je okno rozděleno na dvě poloviny. V každé polovině může může být jiné okno (IPane), které je ovládáno skrz odpovídajícího presentera.



Obrázek 1. Hlavní třídy a rozhraní FileManageru. Použití je vyznačeno nevyplněnou šipkou, zatímco implementace rozhraní či zdědění třídy je naznačeno šipkou plnou.

3.1.1 MainFormPresenter

MainFormPresenter je hlavní ovládací třída celé aplikace. Má několik rolí:

- ***Zachycuje stisknuté klávesy***

MainForm zachycuje všechny stisknuté klávesy a posílá informaci do MainFormPresenteru. Ten rozhoduje jakou část aplikace nechá událost stisknuté klávesy zpracovat.

1 File Manager je moje první aplikace se složitějším GUI a navíc můj první pokus o MVP. Očekávám tedy, že návrh FileManageru nebude úplně kopírovat MVP, ale že tu a tam se objeví nějaké odchylky.

- ***Zpracovává příkazy***

Jak CommandPromptPresenter tak i všechny IPanePresentery mají možnost nahlásit pomocí události nový příkaz ke zpracování. Tím mohou interagovat s hlavní logikou celé aplikace. Tím se příkazy stávají univerzální formou, jak části aplikace mohou požádat hlavní logiku o nějakou změnu celé aplikace. V tuto chvíli příkazy generuje pouze CommandPromptPresenter a to na základě vstupu od uživatele. V příštích verzích by ale příkazy mohly generovat i IPanePresentery, například na základě stisknuté klávesy.

- ***Mění vzhled aplikace***

Na základě příkazů a stisknutých kláves může hlavní stavební moduly aplikace (IPanePresentery) měnit a škálovat. Může například místo pravého presenteru vložit jiný, nebo levý presenter zvětšit na celou obrazovku.

3.1.2 FilesPanePresenter, JobsPanePresenter, SearchResultsPanePresenter

Hlavní funkce IPanePresenterů je zobrazovat množinu záznamů. Používají pro to SortedEntriesHolder a UnsortedEntriesHolder. Obě třídy fungují jako komunikační brána mezi EntriesPane a IPanePresenter. EntriesPane je abstraktní třída která zastřešuje všechny views, které nějakým způsobem zobrazují záznamy.

I přes viditelné podobnosti mezi IPanePresentery, JobsPanePresenter a SearchResultPanePresenter fungují naprosto odlišně než FilesPanePresenter. Zatímco FilesPanePresenter používá SortedEntriesHolder a mění své zobrazení synchronně z hlavního vlákna aplikace, JobsPanePresenter a SearchResultPanePresenter musí reagovat na asynchronní žádosti o přidání či odebrání záznamu. Proto používají UnsortedEntriesHolder, který dovoluje změny ihned propagovat k EntriesPane a tedy je ihned zobrazit. Princip asynchronního přidávání a odebírání záznamů funguje u obou presenterů na bázi událostí. Při inicializaci se presenter zaregistruje u třídy z MFSO a ta pak vyvolává události při každé změně. Při vyvolání změny z vlákna MFSO pak presenter pouze musí pomocí synchronizačního kontextu WinForms zavolat svůj handler na danou událost, ve které přidá nebo vymaže nějaký záznam.

3.1.3 Těžké problémy při implementaci

Jediný větší problém, na který jsem při implementaci FileManageru narazil byla rychlost reagování na asynchronní události. Je to z části problém složité architektury grafických prvků, ale také pravděpodobně tím, že jsem program testoval na starším stroji. Byl jsem nucený změnit chování aplikace tak, aby se asynchronní události neděly tak často a optimalizoval jsem handlery událostí, aby běžely minimální čas. Výsledkem pak je například implementace SearchWrapperu v MFSO, který vyvolává událost až při několika nálezech a neaktualizuje SearchResultsPanePresenter při každém nálezu. Každopádně po mých optimalizacích je aplikace responzivní a i nadále běhá dostatečně rychle.

3.2 MultithredFileSystemOperations

Architektura celé knihovny se točí okolo operací. Objekt, který se vyskytuje prakticky v každé části knihovny je jednotka práce (job), který přesně popisuje operaci i její kontext. Job se nejdříve na základě žádosti od uživatele knihovny vytvoří, poté se předá do fronty a poté se provede. Po provedení je job dealokován a z fronty se vybere job další.

3.2.1 Joby a JobWrapper

Předdefinované joby jsou FileTransferJob, DirectoryTransferJob a DeleteJob. Jak je z názvů patrné FileTransferJob přemisťuje soubor, DirectoryTransferJob přemisťuje složku a DeleteJob maže jak složky tak soubory.

Základní charakteristika jobu je, že je přerušitelný, sám hlásí svůj procentuální průběh, má typ, argumenty a dá se spustit.

Jelikož job samotný se stará pouze o vykonávání operace, bylo nutné vytvořit třídu, která obalí job a poskytne příjemnější API pro práci s vnějším prostředím – JobWrapper. Díky tomu má jednotka práce identifikátor a pamatuje si například svůj stav. S JobWrapperem se nikdy nepracuje přímo. Existují dvě rozhraní, které JobWrapper implementuje: IJobHandle a IJobView. IJobHandle reprezentuje přímý ukazatel na jednotku práce ale také její vlastnictví. IJobView slouží pouze k pozorování jobu a neumožňuje jeho změnu (až na metodu Cancel(), která je thread-safe). Důvod proč se s JobWrapperem nikdy nepracuje přímo je ten, že by mělo být jasné, kdo job vlastní a kdo se na něj pouze dívá. Rozhraní toto jednoduše rozliší. Navíc použití dvou rozhraní umožňuje vymezit práva na pracování s jobem. Protože IJobHandle i JobWrapper jsou interní, jakýkoliv kód, který by chtěl vlastnit job nebo ho zásadně měnit, musí ležet uvnitř knihovny MFSO.

3.2.2 Operations

Statická třída Operations, slouží jako sbírka všech operací, které MFSO dokáže provést. V zásadě to je přemísťování, mazání a hledání souborů či složek. Každá operace je reprezentována jednou nebo více metodami nabízející různé nastavení provedení operace.

Většina metod ve třídě Operations nejdříve vytvoří na míru odpovídající jednotku práce, kterou pak pošle ke zpracování. Je důležité zmínit, že ne nutně se pro každé zavolání metody ve třídě Operations vytvoří jeden job. To jak se práce distribuuje mezi joby si rozhoduje metoda sama.

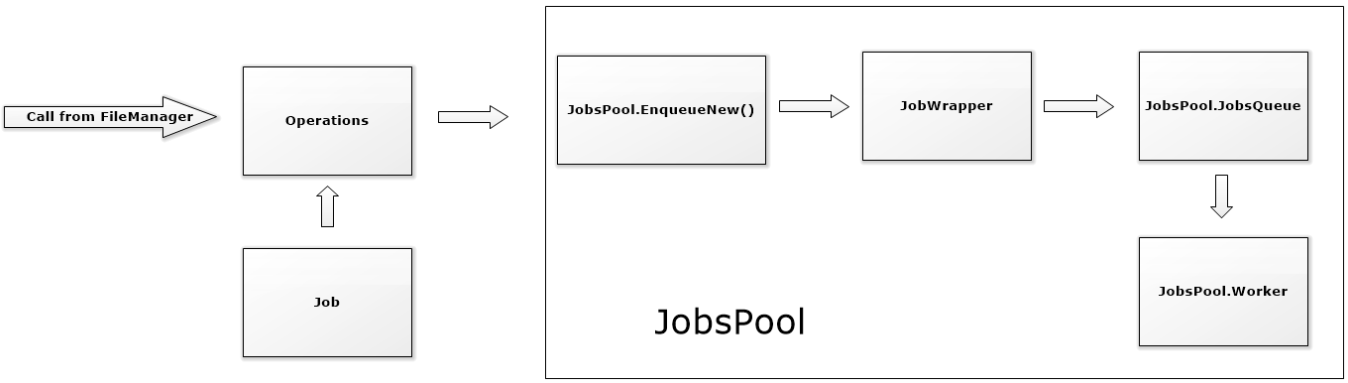
3.2.3 JobsPool

JobsPool je statická třída, která se stará o provedení jobu. Nové jednotky práce obalí do třídy JobWrapper a JobsPool s joby nadále zachází pouze za použití IJobHandle rozhraní.

JobsPool má privátní thread-safe frontu – JobsQueue, kam přidává nové joby. Navíc vlastní také několik[2] „*worker*“ vláken, které berou z fronty připravené joby a provádí je. JobsPool je optimalizovaný na situace, kdy joby chodí ve várkách a mezi várkami je delší časová prodleva.

Dále JobsPool poskytuje veřejné metody, které slouží k synchronizaci stavů jobů. Metoda JobsPool.RegisterNewJobsPane() zaregistruje novou instanci JobsPanePresentera k odebírání aktualizací stavů jobů. Aby nově zaregistrovaný posluchač hned věděl o všech jobech, které JobsPool v sobě uchovává, bylo nutné JobsQueue na chvíli zmrazit. Jinými slovy bylo nutné použít zámek. Nový JobsPane se totiž musí v jednom okamžiku zaregistrovat pro odebíraní aktualizací a zároveň si okopírovat všechny joby z JobsQueue. JobsPool je optimalizovaný na situaci, kdy registrace probíhá jen málokdy a tak ostatní metody JobsPoolu fungují bez zámkově.

2 V aktuální verzi JobsPoolu je pouze jedno *„worker“* vlákno, avšak třída je připravena na použití více vláken.



Obrázek 2. Životní cyklus jobu.

3.2.4 FileSystemNodeSearch a SearchWrapper

Třída FileSystemNodeSearch se velmi podobá představě jobu. Reprezentuje konkrétní popis hledání a dá se přerušit i spustit. Avšak místo procentuálního průběhu, hlásí nalezené soubory a složky. Důvod proč FileSystemNodeSearch není job je, že uživatel nečeká, že by se jako job choval. Pokud uživatel chce něco vyhledat, nechce zažádat o hledání, nýbrž chce začít vyhledávat ihned. Proto je API knihovny pro hledání trochu jiné než pro zbytek operací.

Jediné co zůstává podobné je, že FileSystemNodeSearch se vytváří v třídě Operations a, stejně jako job i FileSystemNodeSearch, je zabalený do SearchWrapperu. Ten je předaný zpět jako ISearchHandle a tím je převedeno i jeho vlastnictví z knihovny MFSO na volajícího. V případě FileManageru, MainFormPresenter pak naalokuje nový SearchResultsPanePresenter, kterému předá ISearchHandle. SearchResulPanePresenter hledání odstartuje.

SearchWrapper spustí hledání v novém vlákně a sám si naalokuje konzumní vlákno. Konzumní vlákno reprezentuje podtřída SearchWrapper.Consumer. Zatímco vyhledávací vlákno přidává výsledky do thread-safe kolekce, konzumní vlákno si je vybírá a posílá vlastníkovi ISearchHandelu. Jako společná thread-safe kolekce byla vybrána BlockingCollection, jelikož umožňuje zarazit vyhledávací vlákno, pokud nachází příliš mnoho výsledků, které konzumní vlákno nestíhá odebírat. Jako notifikaci konzumnímu vláknu, že kolekce je dostatečně naplněná byla použita třída Monitor, jelikož se očekává, že výsledky budou přicházet postupně a ne ve várkách, jako tomu je u jobů v JobsPoolu. Třída Monitor tedy v této situaci představovala minimální zdržení vyhledávacího vlákna.

3.2.5 Těžké problémy při implementaci

Při implementaci knihovny jsem narazil na více problémů. Většina z nich ale spočívala v nerozumném používání vláken. Řídké používání Parallel.ForEach a PLINQu mělo za následek pomalé reakce hlavního vlákna aplikace, pravděpodobně kvůli velkému tlaku na mé dvě jádra. Taktéž moje představa, že každá složka se bude prohledávat, kopírovat a přesouvat v jiném vlákně s cílem urychlit operaci, se ukázala jako mylná (alespoň pro můj dvou jádrový procesor). Také prvotní implementace JobsPoolu počítala s větším vícevláknovým výkonem. Původně se totiž operace hned po zařazení do JobsPoolu začaly provádět jako nové Tasky. Výsledek byl takový, že hlavní vlákno nestíhalo aktualizovat záznamy operací a okno se stalo neresponzivním. Předpokládám, že bude problém i v mých déle trvajících handlerech na tyto události, ale po lehkém zamyšlení jsem správně usoudil, že hlavní chyba byla jinde.

4 Původní požadavky a dosažené cíle

Původní požadavky dle specifikace byly následující:

1. *Program by měl efektivně kopírovat, mazat, přesouvat a vyhledávat soubory.*
2. *Program by uživateli měl nabízet třídění dle různých parametrů souboru v aktuálním adresáři.*
3. *Uživateli by mělo být umožněno efektivně přejmenovávat větší množství souborů najednou.*
4. *Program by měl poskytnout k práci dvě okna, ve kterých uživatel bude moci navigovat ve dvou adresářích.*

1. Uznávám, že výsledný program neprovádí všechny tyto operace s maximální efektivitou. Myslím ale, že když se vezme do úvahy, že aplikace uživateli dává zpětnou vazbu o stavech operací a navíc dokáže zůstat responzivní, efektivita operací je velmi dobrá.

2. Třídění je umožněno na základě jmen sloupců FilesPane. Tedy požadavek splněn.

3. Mezi definovanými příkazy je i příkaz move, jehož varianta dokáže přejmenovat více souborů najednou. Pro více informací prosím nahlédněte do uživatelské dokumentace. Tedy požadavek splněn.

4. Hlavní okno File Manageru dává uživateli k dispozici právě dvě okna. Tedy uživatel si může pohodlně prohlížet dvě složky zároveň.

5 Možné budoucí vylepšení

5.1 FileManager

I přesto, že FileManager je fungující aplikace, profitovala by z menší přestavby. Nyní vím o WinForms více než, když jsem grafické rozhraní navrhoval a myslím, že bych odvedl lepší práci. Především bych se soustředil na minimalizování grafických prvků.

Dále systém, jakým FileManager registruje stisknuté klávesy je poněkud omezený. Umožňuje například zachytávat různé dvojstisky kláves jako *Shift + j*, což by FileManager zachytil jako velké *J*. Bohužel ale neumožňuje zachytávat další události spojené s tlačítky jako je KeyDown a KeyUp.

Jak jsem zmínil v uživatelské dokumentaci, k lepšímu pocitu z používání aplikace by bylo vhodné přidat pár oken. Například vestavěný manuál nebo seznam oblíbených složek. Vhodné by také bylo přidat způsob jak měnit přednastavené hodnoty jako je třeba font, barva ale i počet a definice zobrazovaných sloupců ve FilesPane.

5.2 MultithreadedFileSystemOperations

Největší limitace knihovny je absence heuristiky pro odhadování počtu vláken pro vykonání nějaké operace. Samozřejmě je to trochu velký skok, jelikož takové chování by vyžadovalo monitoring stavu celého počítače a navíc ještě algoritmus pro každou operací, který by mohl být parametrizovaný počtem vláken. Nicméně v této oblasti vidím velké potencionální zlepšení aplikace.

Velká limitace knihovny je menší počet předdefinovaných jobů. Je potřeba tuto množinu buď zvětšit, nebo se zamyslet nad způsobem, jak vykonávat joby definované uživatelem nebo například nějakým pluginem.

Dále mě napadá už jen optimalizování volání událostí zpět do hlavního vlákna aplikace.

6 Nahlašování nalezených chyb v aplikaci

Zatím je File Manager jen můj osobní menší projekt, a proto nemá oficiální githubovou stránku. Pokud se vám bude zdát chování programu nelogické či nesprávné, neváhejte mě kontaktovat a chybu nahlásit na david.burian(at)me.com.