ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

Katedra biomedicínské informatiky

Návrh a implementace knihovny pro ovládání robotické paže

Studijní program: Biomedicínská a klinická technika

Studijní obor: Biomedicínská informatika

Autor diplomové práce: Milan Poláček

Vedoucí diplomové práce: Mgr. Radim Krupička, Ph.D.

České vysoké učení technické v Praze, Fakulta biomedicínského inženýrství

Katedra biomedicínské informatiky | Akademický rok: 2013/2014

Zadání bakalářské práce

Student:	**Milan Poláček**
Obor:	Biomedicínská informatika
Téma:	**Návrh a implementace knihovny pro ovládání robotické paže**
Téma anglicky:	Design and implementation of library for robot arm control

Zásady pro vypracování:

Cílem bakalářské práce je navrhnout a implementovat v jazyce C# knihovnu pro ovládání robotické paže. Knihovna by měla být dostatečně abstraktní, aby umožňovala připojit libovolnou paži s libovolnými motory a dostatečně robustní, aby implementovala všechny potřebné funkce pro ovládání robotické paže. Vytvořte a důkladně zdokumentujte interface, který definuje třídy a metody pro začlenění ovládací knihovny robotické paže (návrhový vzor Adaptér). Knihovnu ověřte na libovolné robotické paži přístupnou na KBI.

Seznam odborné literatury:

[1] Judith Bishopová, C# návrhové vzory, ed. 1, Zoner Press, 2010, ISBN 978-80-7413-076-2
[2] Christian Nagel, Bill Evjen, Jay Glynn, C# 2008 programuje profesiálně, ed. 1, 2009, Computer Press, 978-80-251-2401-7

zadání platné do:	11.09.2015
Vedoucí:	Mgr. Radim Krupička
Konzultant:	Ing. Jan Kauler, Ph.D.

vedoucí katedry / pracoviště

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

děkan

V Kladně dne 17.02.2014

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci na téma **Návrh a implementace knihovny pro ovládání robotické paže** vypracoval samostatně. Veškerou použitou literaturu a podkladové materiály uvádím v přiloženém seznamu literatury.

V Kladně dne _____ _____

podpis

PODĚKOVÁNÍ

Chtěl bych poděkovat zejména svému vedoucímu práce Mgr. Radimovi Krupičkovi, Ph.D. za cenné připomínky a rady při realizaci projektu. Také bych rád poděkoval Ing. Janu Kaulerovi, Ph.D. za poskytnuté konzultace a informace k této problematice.

Velké poděkování náleží mé rodině, která mne nejen při psaní této bakalářské práce, ale i po dobu celého studia podporovala.

Poděkování také patří mým kolegům a kamarádům z IT skupiny, kteří mi během let, co s nimi pracuji a trávím čas, poskytli cenné rady a zkušenosti.

A v neposlední řadě bych chtěl ještě poděkovat i kolegům z Albertova, kteří mě také mnoho naučili.

Název bakalářská práce:

Návrh a implementace knihovny pro ovládání robotické paže

Abstrakt:

Předmětem bakalářské práce je návrh a implementace knihovny pro ovládání robotické paže. Obecný návrh knihovny umožňuje její využití pro více druhů paží. Práce seznamuje s využitím návrhových vzorů v problematice robotických paží. Rovněž jsou zde probrány matematické analýzy sloužící pro návrh knihovny. Nedílnou součástí této práce je i rozbor vlastností pohybů horní končetiny. Výsledkem bakalářské práce je návrh tříd podle návrhových vzorů Adapter a Composite. Bakalářská práce je zakončena tvorbou a implementací tříd pro ovládání robotické paže a otestováním na jedné z robotických paží, kterou vlastní Fakulta biomedicínského inženýrství v Kladně.

Klíčová slova:

robotická paže, .NET knihovna robotické paže, C#, návrhové vzory

Bachelor´s Thesis title:

Design and implementation of library for robot arm control

Abstract:

The aim of this thesis is design and implementation of library for control of robotic arm. General design of the library enables its utilization for more kinds of robotic arms. Part of the thesis is mathematical analysis which is used for design of the library. This thesis acquaints with design patterns in the problems of robotic arms. Integral part of the thesis is analysis of properties of human upper limb movements. The result of bachelor's thesis is design of classes according to design patterns, Adapter and Composite. The bachelor thesis is terminated with the creation and implementation of a class to control a robotic arm and this software is tested on one of the robotic arms owned by the Faculty of Biomedical Engineering in Kladno.

Key words:

robotic arm, .NET library robotic arm, C#, design patterns

Obsah

1. Úvod 1

1.1. Přínosy práce 1

2. Současný stav problematiky 2

2.1. Požadavky na knihovnu RoboticControl 2

2.2. Základní předpoklady řídícího softwaru 2

2.3. Robotické paže na Fakultě biomedicínského inženýrství 3

2.3.1. Robotická paže založená na stavebnici Robix 3

2.3.2. Robotická paže AL5A 4

2.3.3. Robotická paže založená na návrhu Ing. Jana Froňka 5

2.4. Způsoby návrhu softwaru 6

2.4.1. Návrhové vzory 6

3. Analýza a návrh softwarového řešení 12

3.1. Analýza anatomických vlastností paže pro návrh knihovny 12

3.2. Analýza matematické modelu robotické paže 12

3.2.1. Analýza kinematiky hmotných bodů 12

3.2.2. Analýza dynamiky 14

3.3. Požadavky na knihovnu RoboticControl 14

3.4. Funkční požadavky na knihovnu RoboticControl 15

3.5. Alternativy řešení návrhových vzorů 17

3.6. Metodika vývoje softwaru 17

4. Vlastní řešení softwaru 19

4.1. Návrh jednotlivých komponent 21

4.1.1. Vlastnosti a možnosti rozhraní IMotor 21

4.1.2. Vlastnosti a možnosti rozhraní IRoboticControl 24

4.1.3. Vlastnosti a možnosti třídy Motor 29

4.1.4. Vlastnosti a možnosti třídy Joint 29

4.1.5. Vlastnosti a možnosti třídy RoboticArm 31
5. Testování návrhu 35
6. Diskuze 39
7. Závěr 40
8. Seznam použité a citované literatury 41

1. Úvod

V současné době se na Fakultě biomedicínského inženýrství ČVUT vyskytují 3 robotické paže. Dvě z nich jsou vytvořeny komerční sférou a obě pro ovládání využívají vlastní knihovnu. Při vývoji nové robotické paže na FBMI byla zjištěna potřeba kvalitní robustní knihovny, která by umožňovala společný interface pro všechny robotické paže, které fakulta vlastní a bude do budoucna vlastnit.

Cílem této práce je vytvořit jednu knihovnu (interface) robotické paže (dále jen RoboticControl) metod a funkcí, která zastřeší všechny ovládací prvky nižších úrovní funkcí nebo knihoven. Uživatel pak bude moci používat jednu knihovnu bez ohledu na použitou robotickou paži. Tato knihovna by měla poskytovat flexibilnější výuku robotiky na Katedře biomedicínské informatiky, kde by si mohli studenti vyzkoušet použitelnost kódu skrze různé platformy pro ovládání. Také si osvojí znalosti používání objektového programování při navrhování vlastního uživatelského rozhraní (GUI).

Práce se bude zabývat dispozicemi lidské paže viz kapitola 3.1. Dále prozkoumá problematiku z matematického hlediska viz kapitola 3.2. A na základě těchto analýz a konzultací s Ing. Jan Kaulerem, Ph.D. a Mgr. Radimem Krupičkou, Ph.D. bude sestavena univerzální funkcionalita viz 3.3.

1.1. Přínosy práce

Bakalářská práce by měla být základem pro další práce nejen v informatickém oboru, ale i v oboru technickém. Měla by sjednocovat volání funkcí již existujících robotických paží, ale i ty, které budou následně vytvořeny ať už výrobci nebo studenty v rámci nějakého projektu. Tato bakalářská práce tedy bude zajišťovat základ pro rychlé nasazení robotické paže buď to do výuky, nebo jiné pro účely. Snahou této práce je také to, aby ve větším, než fakultním měřítku, nezáviselo vytváření dalších knihoven pro jiné robotické paže jen na Fakultě biomedicínského inženýrství.

2. Současný stav problematiky

V této kapitole se budu zabývat obecným návrhem a schématem knihovny RoboticControl a na ni navazujících tříd a knihoven. Knihovna RoboticControl by měla být hlavním Interface, kterým se tato práce bude zabývat.

Dále krátce specifikuji robotické paže, které jsou na Fakultě biomedicínského inženýrství k dispozici. A proberu zde i tématiku návrhových vzorů a jejich výhody pro objektové programování.

2.1. Požadavky na knihovnu RoboticControl

Hlavním požadavkem a účelem této bakalářské práce je sjednotit volaní funkcí k jednotlivým robotickým pažím. K ovládání robotických paží je potřeba, aby uživatel prostudoval manuál popřípadě ovládací knihovny ke každé z nich. To velmi komplikuje výuku předmětů používající tyto výukové pomůcky, kdy studenti nejprve musí prostudovat implementační jazyk jednotlivých příkazů a dále experimentovat, zda porozuměli správně funkcím.

Knihovna RoboticControl bude používat knihovnu robotická paže (dále jen RoboticArm), která bude využívat knihovnu motorů (dále jen Motor). Motor volá samostatné příkazy v knihovnách jednotlivých robotických paží.

Při změně robotické paže se změní volání knihovny RoboticArm, která bude též volat jiné knihovny Motor. Uživatel však nepozná rozdíl, protože bude stále pracovat s RoboticControl.

Uživatel (student) může například volat funkci *otočit paží doprava* ve svém kódu. A ať připojí jakoukoli robotickou paži, která bude mít doimplementovány knihovny RoboticArm a Motor, bude kód fungovat.

2.2. Základní předpoklady řídícího softwaru

Předpokladem základních funkcionalit softwaru je navrhnout základní metody tak, aby bylo možné následně vytvářet třídy pro sekvence příkazů, které budou věrně napodobovat pohyby a úkony živé paže řízené centrální nervovou soustavou.

Je tedy zřejmé, že řídící software bude nutno odstupňovat podle jednotlivých komponentů a ovládacích knihoven k jednotlivým kontrolérům.

Aspekt odstupňování tříd nás přivádí k tomu, že je zde vhodné vytvořit řídící knihovny pro jednotlivé motory, robotické paže a knihovnu RoboticControl, která zastřeší všechny knihovny jednotlivých robotických paží. Účelem této práce by tedy mělo být vytvoření knihovny RoboticControl a doplňkově vytvořit knihovnu se základní funkčností pro jednu z robotických paží, na které by se odzkoušela funkčnost. Předpokladem knihovny RoboticControl je, že bude schopna ovládat, bez rozdílu, jakoukoliv robotickou paži, kterou uživatel připojí k řídicímu systému, na základě již připravených knihoven k jednotlivé robotické paži. To klade důraz na nutnost velké abstrakce knihovny RoboticControl, vzhledem k tomu, že může nastat situace, kdy ne všechny robotické paže budou disponovat stejnými funkcionalitami. Proto je nutné vytvořit vhodné nastavení ošetření chybových stavů a správně je klasifikovat jako chybu či varování.

Nevhodná klasifikace těchto stavů, by nemusela při připojení určité konfigurace robotické paže plně fungovat nebo by s touto knihovnou byla zcela nefunkční.

2.3. Robotické paže na Fakultě biomedicínského inženýrství

V současné době má Katedra biomedicínské informatiky na Fakultě biomedicínského inženýrství k dispozici dvě robotické paže a robotické stavebnice, ze kterých je možné postavit taktéž robotickou paži.

Hlavním nedostatkem v této problematice je to, že robotické paže jsou zhotoveny od rozdílných výrobců a zatím se na tyto výrobky nevztahují žádné návrhové standardy ani ISO normy či jiné normy. Každá robotická paže tedy pracuje ve většině případů s rozdílným typem motorů, což znemožňuje jednoduché zobecnění ovládacích módů a ovládacích funkcí. A tento problém má řešit tato bakalářská práce.

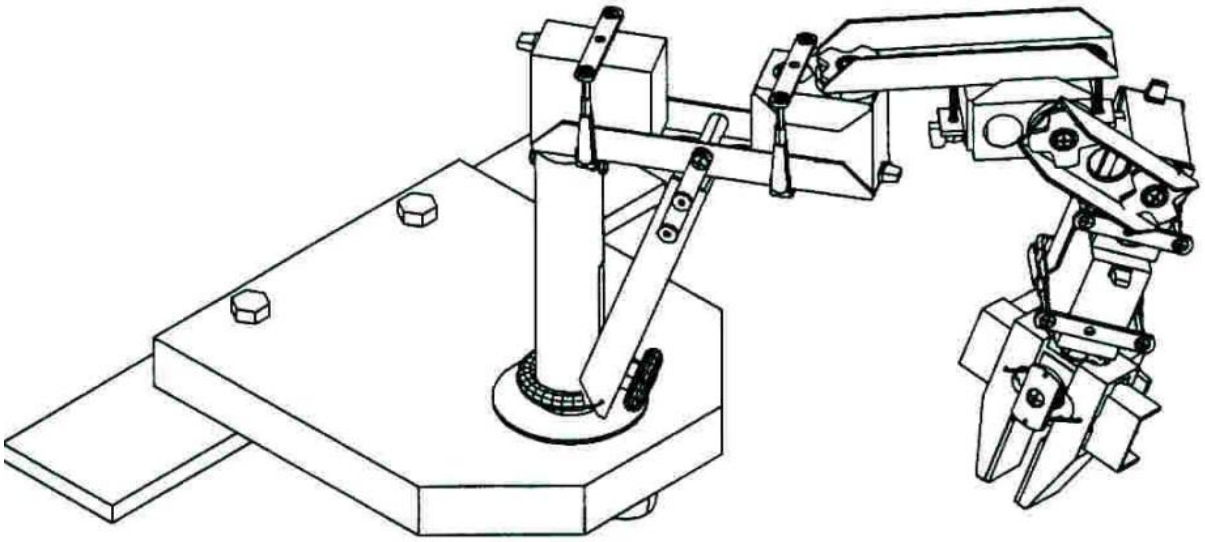
2.3.1. Robotická paže založená na stavebnici Robix

Tato robotická paže je založena na stavebnici Robix od firmy Rascal a poskytuje set určený pro studenty složený z kontroléru, servo motorů a spojovacích prvků konstrukce.

Kontrolér je schopen pracovat až s 32 servo motory. Ty je možno kontrolovat pomocí 32 analogových vstupů a řídit pomocí 32 proudových výstupů.

Robotická paže je postavena podle návodu, který poskytuje výrobce. Podle tohoto návodu je několik možností, jaký typ robotické paže je možno sestavit. Na Katedře biomedicínské informatiky zvolili model Chemist projekt (viz obr.1).

Model této robotické paže se skládá z 6 servo motorů HS-422. Čtyři servo motory na paži umožnují pohyb ve 4 stupních volnosti. Tedy v ose x (respektive pohyb doleva a doprava), kde jsou umístěny 2 servo motory, které dovolují robotické paži pohyb s větší flexibilitou. Další 2 servo motory umožňují pohyb v ose y, ale i v ose z. Jeden servo motor zajišťuje možnost rotace v zápěstí a druhý servo motor je určen pro takzvaný grip.



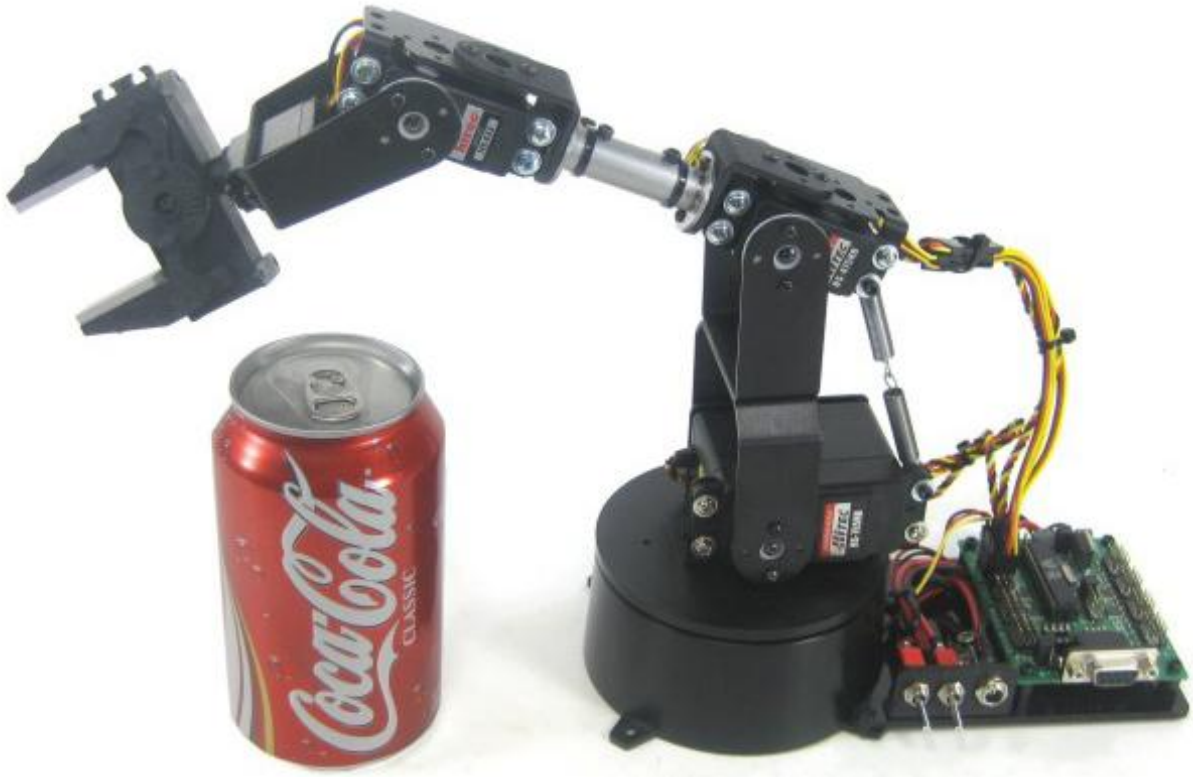
obr.1 Robotické paže Chemist projekt

2.3.2. Robotická paže AL5A

Robotická paže AL5A je stavebnice od firmy Lynxmotion se zabývá výrobou robotických produktů.

Tato robotická paže je složena z 5 servo motorů a má rovněž grip jako má robotická paže postavená ze setu Robix. V základně je umístěn 1 servo motor (HS-422), který umožňuje pohyb v ose x. Další servo motor je v rameni (HS-755HB). Tento motor umožňuje pohyb v ose y. Dále se nachází motor v lokti (HS-645MG) umožňující pohyb v osách y a z. Třetí servo motor (HS-422) pracuje v rozmezí os y a z. Poslední servo motor (HS-422) je určen pro grip. Robotická paže může tedy pracovat ve čtyřech stupních volnosti.

Tyto servo motory řídí kontrolér (SSC-32) s 32 kanály a přesností 1μS, který umožňuje pohyb v synchronizovaném módu nebo v takzvaném skupinovém. V tomto kontroléru je umístěn mikroprocesor ATMEGA168.

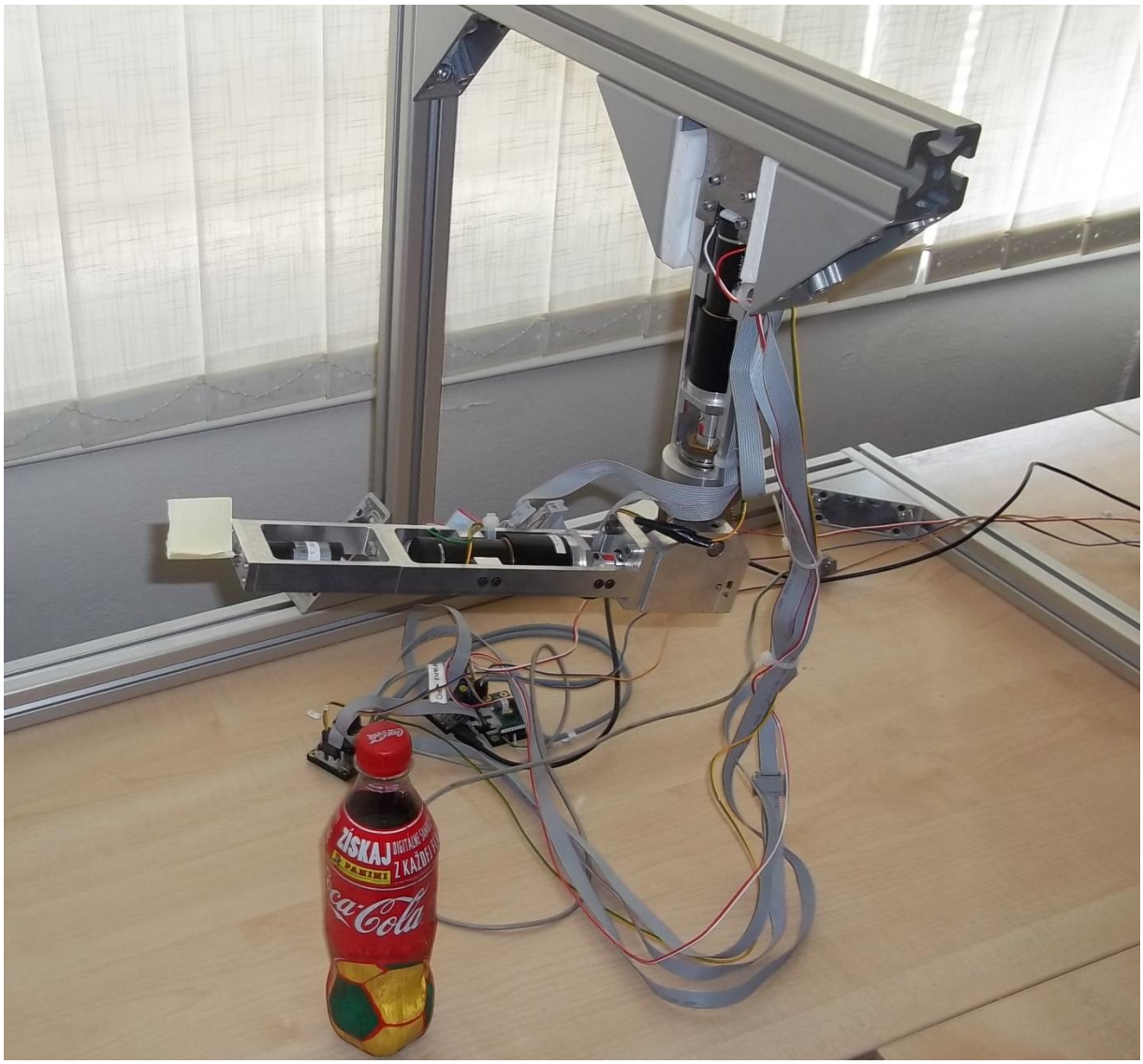


obr.2 Fotografie modelu AL5A s měřítkem k plechovce nápoje CocaCola

2.3.3. Robotická paže založená na návrhu Ing. Jana Froňka

Tato robotická paže byla vytvořena na základě diplomové práce Ing. Jana Froňka (obr.3). Disponuje 3 stupni volnosti a na rozdíl od dvou předchozích není opatřená gripem. Robotická paže je opatřena třemi DC motory od firmy Maxon včetně převodovek. Tyto sestavy zajišťují pohyb robotické paže a to rotaci zápěstí, elevaci v lokti a poslední zajišťuje vnitřní a zevní rotaci v ramenním kloubu.

Všechny tři DC motory jsou řízeny třemi kontroléry EPOS 2. Tyto jednotky jsou propojeny průmyslovou CAN sběrnicí a vše je řízeno master kontrolérem.



obr.3 Fotografie robotické paže podle návrhu Ing. Froňka

2.4. Způsoby návrhu softwaru

V této kapitole se budeme zabývat, známými postupy, jak by bylo vhodné řešit tuto problematiku, tak aby výsledek této práce vyhovoval známým konvencím a standardům.

2.4.1. Návrhové vzory

S prvními náznaky návrhových vzorů se mohli začínající programátoři možnost seznámit již v roce 1987 na konferenci OOPSLA v Orlandu. Poznatky prezentované Ward Cunningham a Kent Beck měli pomoci začínajícím programátorům naučit se efektivněji psát svůj objektový kód v jazyce Smalltalk. Problematika návrhových vzorů je úzce spjatá s objektovým programováním, jejich potřebu si uvědomovala většina programátorů.

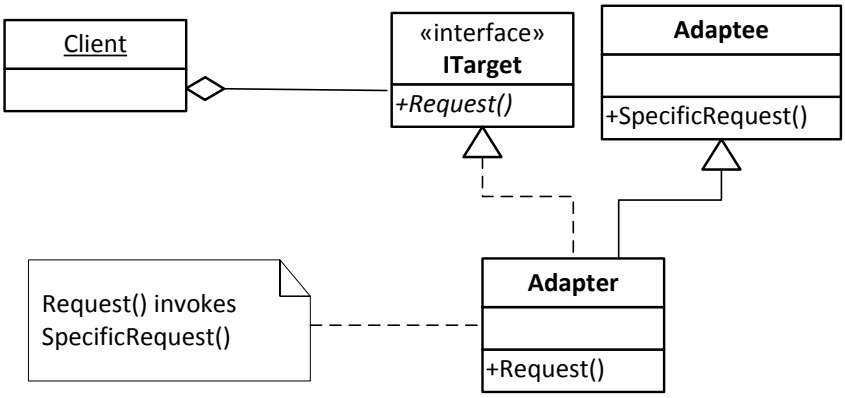
Za stěžejní práci zabývající se touto problematikou se považuje kniha Design Patterns: Elements of Reusable Object-Oriented Software, jejímiž autory jsou Erich Gamma, Richard Helm, Ralph Johnson a John Vlissides.

Návrhové vzory neřeší problémy jako algoritmy, ale slouží jako obecné šablony nezávislé na kódu a funkcích. V současné době je u každého většího projektu nutností se zamyslet nad tím, zda nepoužít nějaký návrhový vzor.

V následujících kapitolách se budu zabývat některými z návrhových vzorů, které by bylo vhodné použít v této práci. Jedná se o strukturální návrhové vzory, které se snaží zvyšovat flexibilitu vytvořeného kódu.

2.4.1.1. Adapter

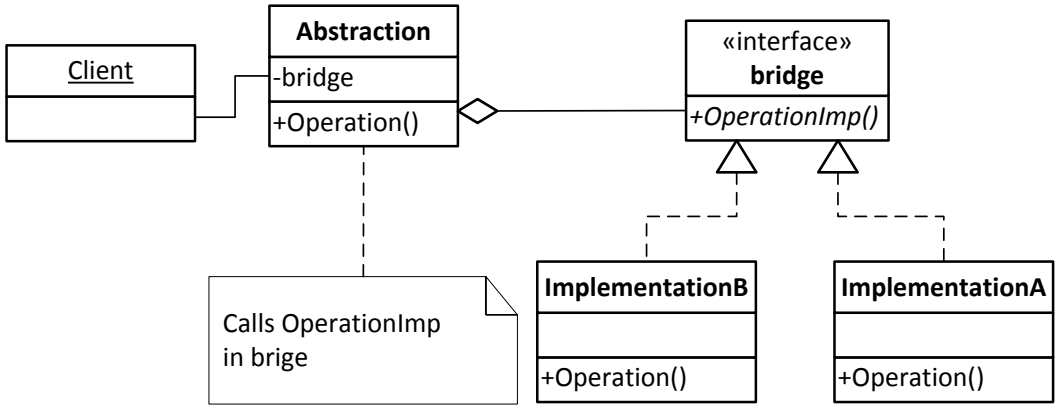
Návrhový vzor Adapter (obr.4) byl použit při přechodu na jinou procesorovou architekturu ve firmě Apple, jak je uvedeno v knize C# návrhové vzory [**1**]. Protože firma IBM ukončila výrobu procesorů řady PowerPC, byla firma Apple nucena začít používat ve svých výrobcích procesory od firmy Intel. Vývojáři byli najednou postaveni před dilema, zda oznámit konec podpory softwaru jak starého, který by na nových osobních počítačích vybavenými procesory od firmy Intel nefungoval, tak nového softwaru, který by nebyl zpětně kompatibilní se starým hardwarem (PowerPC) z důvodu rozdílné instrukční sady. Vývojáři operačního systému MacOS, avšak přišli s řešením v podobě frameworku, který vytvořili (Accelerate) a který je schopen převádět vše na správnou instrukční sadu, jak pro architekturu Intel, tak architekturu PowerPC (IBM). Tím se podařilo zajistit funkčnost starého softwaru a vytvořila se i zpětná kompatibilita pro nově vytvořený software.



obr.4 UML diagram návrhového vzoru Adapter

2.4.1.2. Bridge

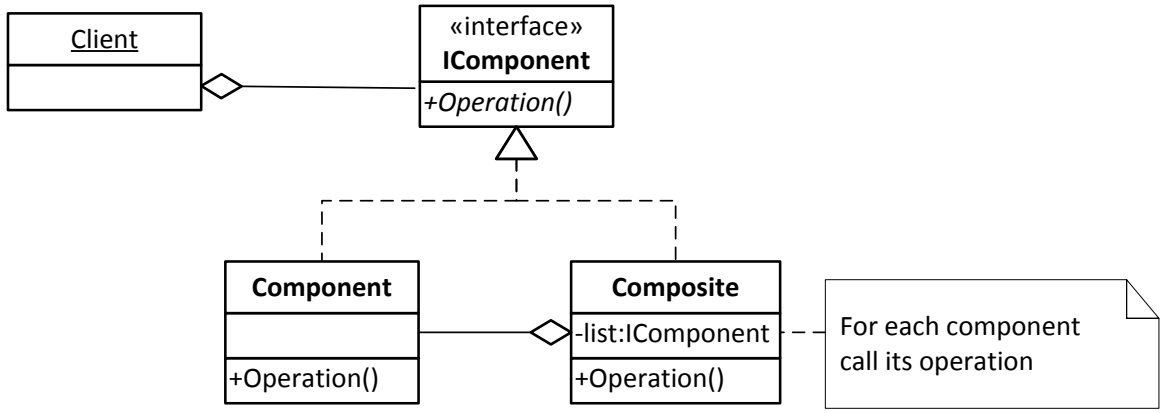
Návrhový vzor Bridge (obr.5) odděluje návrh, implementaci i dané řešení do samostatných celků. Tuto výhodu srozumitelně popisuje J. Bishop [**1**] ve svém příkladu. Operační systém Windows nastavuje cestu k .NET Frameworku podle verze, ve které je napsána aplikace. Bridge tedy využívá operační systém, takže interface rozhoduje o použití funkcí .NET Frameworku verze 3.0 nebo 3.5. Nutnost implementovat Bridge nastává ovšem jenom tehdy, pokud mají být použity zaměnitelné funkce mezi dvěma nebo více verzemi určitého softwaru.



obr.5 UML diagram návrhového vzoru Bridge

2.4.1.3. Composite

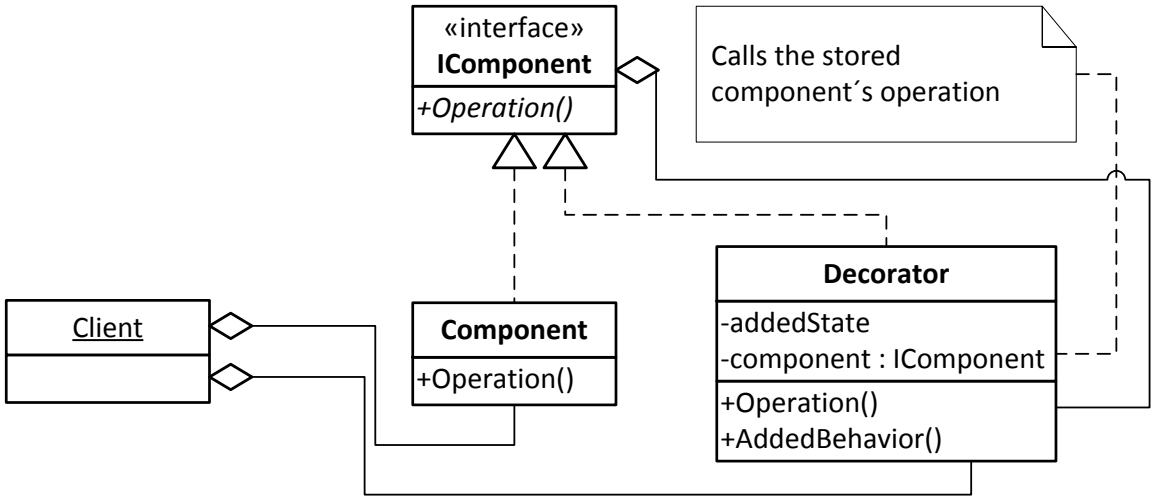
Návrhový vzor Composite (obr.6), jak uvádí J. Bishop [**1**] se dá uplatnit například v internetovém bankovnictví. Uživatel používá IComponent svůj uživatelský účet v internetové bance. U dané banky má například dva bankovní účty (Component). Tyto účty jsou tedy v seznamu (respektive Listu) IComponent, který je součástí Composite. Účty mají své funkce jako například měnění svého stavu, posílání transakcí, svůj stav, typ atd. A tyto funkce a atributy dědí Composite.



obr.6 UML diagram návrhového vzoru Composit

2.4.1.4. Decorator

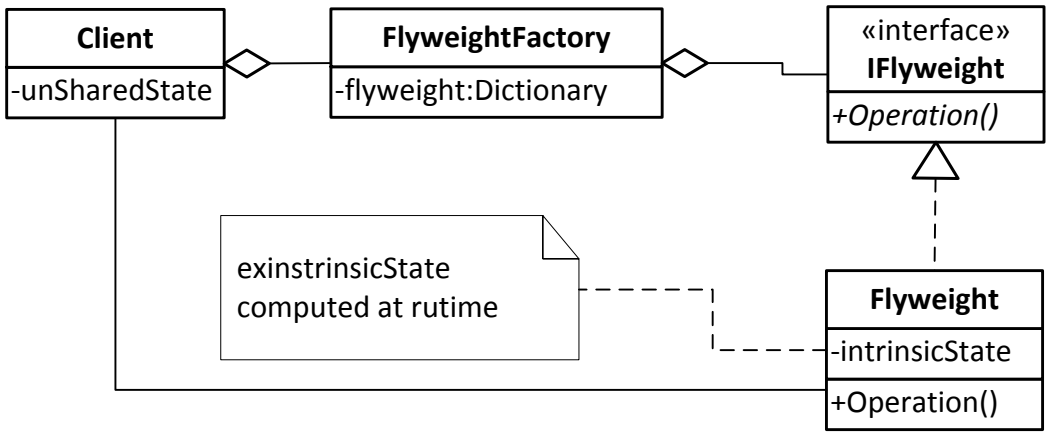
Snahou tohoto návrhového vzoru není obsáhnout všechnu funkcionalitu a předvídat všechny budoucí potřeby dané třídy, jak popisuje ve své knize J. Bishop [**1**]. Setkat se s tímto návrhovým vzorem můžeme v jazyce C# při použití knihoven pro aplikační rozhraní System.IO, kdy Component (viz diagram na obr.7) je knihovna System.IO.Stream, které jsou Dekorátorem například knihovny System.IO.FileStream a System.IO.MemoryStream. Podtřídy, které jsou Dekorátorem dědí instanci System.IO.Stream a jejichž vlastnosti a metody se tedy v důsledku toho k této třídě vztahují, ale zároveň rozšiřují o novou funkcionalitu bez jakéhokoliv ovlivnění původní třídy.



obr.7 UML diagram návrhového vzoru Decorator

2.4.1.5. Flyweight

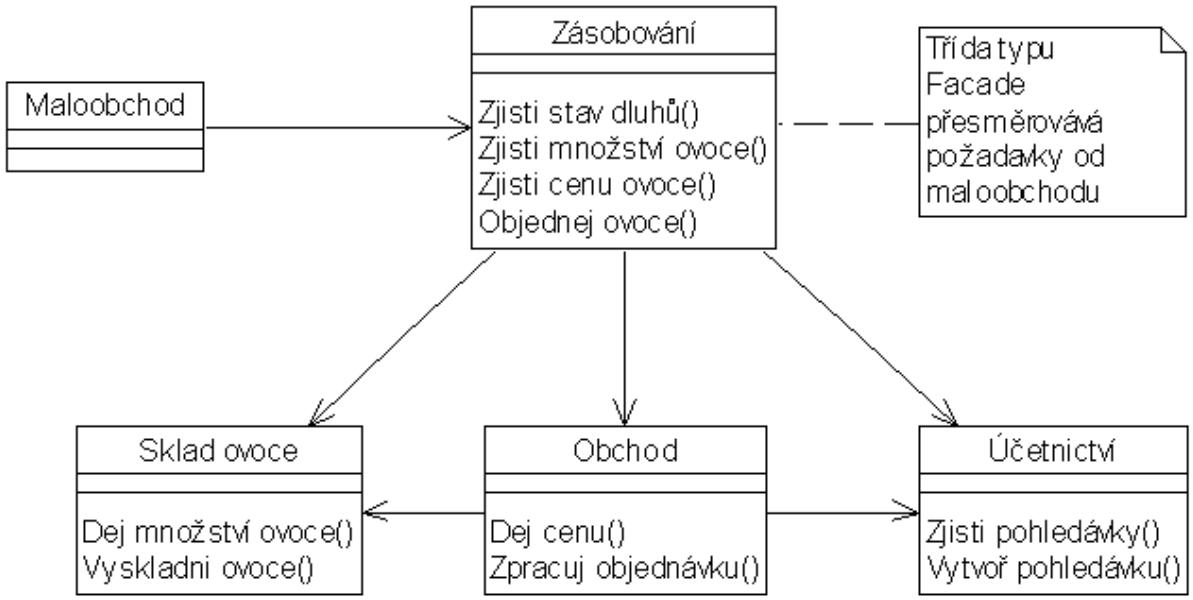
Flyweight (obr.8) neboli muší váha je návrhový vzor, který má snížit zejména paměťové nároky. Ing. Dvořák [**2**] uvádí na svém webu příklad, který jednoduše a stručně popisuje princip tohoto návrhového vzoru. Vezměme si, že máme objekt písmeno, který popisuje jedno písmeno. Objekt písmeno má atributy jako je font, velikost a o jaké písmeno se jedná. Pokud tedy napíšeme nějaké slovo například o 5 znacích, objekt písmeno se nám alokuje v paměti 5 krát, což v tomto případě není tak kritické, ale při psaní delšího textu by takto fungující textový editor byl nevyhovující z důvodu paměťové náročnosti. Návrhový vzor Flyweight funguje tak, že atributy, které jsou pro všechny společné, uloží na jedno místo do FlyweightFactory a tím snižuje paměťové nároky. V našem případě by ve FlyweightFactory byly atributy font a velikost. A tento objekt by se vytvořil vždy při využití nového nastavení objektu písmeno.



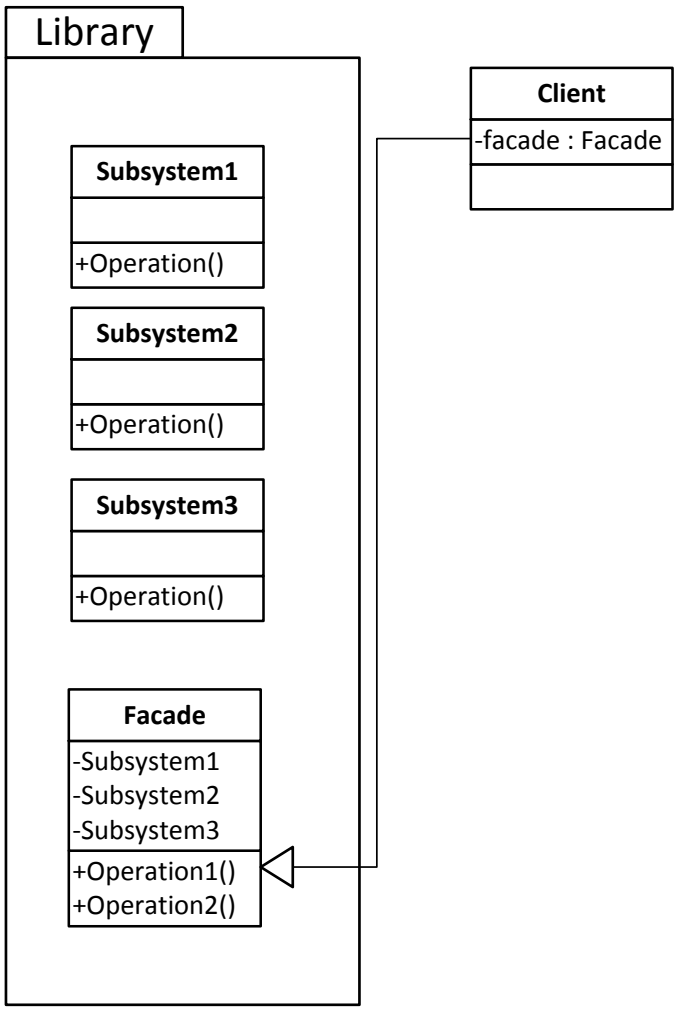
obr.8 UML diagram návrhového vzoru Flyweight

2.4.1.6. Facade

Návrhový vzor Facade (fasáda - obr.10) je jedním z dnes nejčastěji používaných návrhových vzorů. U tvoření programu se nám velice často stává, že se přestáváme orientovat, které funkce (systémy) mají předcházet funkcím, a které na nich závisí (subsystémy). Využiji zde příklad maloobchodu Ing. Dvořáka z webu [**3**], který je uveden na obr.9. Uživatel zná (vidí) jen operace třídy Zásobování. Subsystémy Obchod, Sklad ovoce a Účetnictví vidí a může používat jen Fasáda Zásobování. Uživatel tedy nemusí znát, že funkce *Dej množství ovoce* je součástí subsystému *Sklad ovoce* a je tedy nutno nejprve inicializovat tento objekt. Uživatel zavolá funkci *Zjisti množství ovoce* z fasády Zásobování a o zbylé operace se postará fasáda.



obr.9 Diagram převzatý z [**3**]



obr.10 UML diagram návrhového vzoru Facade

3. Analýza a návrh softwarového řešení

3.1. Analýza anatomických vlastností paže pro návrh knihovny

Všechny modely robotických paží se snaží kopírovat (napodobovat) základní pohyby každé zdravé lidské paže v jednotlivých kloubech. Bohužel, ale ne každý model robotické paže disponuje stejnou pohyblivostí respektive množstvím motorů.

Nyní nastíním možnosti pohybu zdravé paže.

Popis začnu od zápěstního kloubu, který se z technického hlediska může pohybovat nahoru a dolů (podle lékařské terminologie palmární flexi a extenzi). Dále je možno zápěstím pohybovat doleva a doprava (dle lékařské terminologie radiální a ulnární dukce). Posledním pohybem je rotace kolem osy doleva či doprava (v lékařské terminologii pronace a supinace). To nám dává tři stupně volnosti. Odvozené pohyby, ke kterým je zapotřebí všech těchto pohybů (respektive motorů), což je krouživý pohyb (cirkumdukce) není zapotřebí implementovat. Jak jsem se již zmínil, jedná se o odvozený pohyb a žádá si tedy jen správnou implementaci sekvence pohybů (motorů) v zápěstním kloubu.

Dalším kloubem směrem nahoru (cranialis) je loketní kloub. Tento kloub má rovněž pohyb nahoru a dolů (flexe a extenze) a dále rotaci kolem své osy (dle lékařské terminologie pronace a supinace). To znamená, že má dva stupně volnosti.

Jedním z posledních kloubů je kloub ramenní. Tento kloub se pohybuje ve směru dopředu respektive předpažení (v lékařské terminologii anteverze) a dozadu neboli zapažení (podle lékařské terminologie retroverze). Další pohyb je připažení (addukce a abdukce). Za poslední pohyb se dá považovat vzpažení (terminologie elevace), který někteří považují za odvozený, protože se jedná o kombinovaný pohyb.

3.2. Analýza matematické modelu robotické paže

3.2.1. Analýza kinematiky hmotných bodů

V návrzích robotických paží se setkáváme s kinematickými řetězci. Tyto řetězce se dále pak dělí na kinematické dvojice. Popisem těchto dvojic, transformačními maticemi (v Denavit-Hartenbergově notaci) jsme schopni určit jejich vzájemnou pozici středů a zda se daná soustava pohybuje posuvem či rotací vůči sobě.

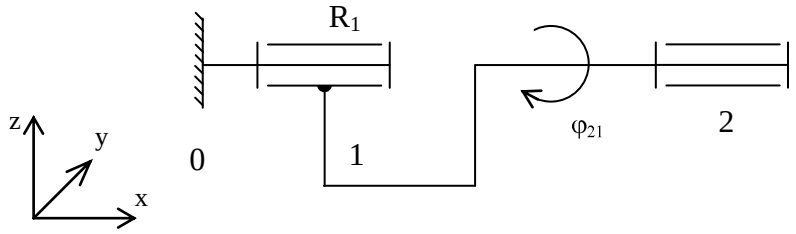
Na základě součinu transformačních matic jsme tedy schopni určit pozici konce kinematického řetězce, od počátku kinematického řetězce (tzv. rámu). Tyto matice, ale neslouží jen k zápisu pozic a pohybů jednotlivých kloubů. Na jejich základě se stanovuje, o kolik se jednotlivé klouby mají pootočit či posunout, aby se konec kinematického řetězce (v našem případě ruka) dostal z původní pozice na žádanou pozici.

Pro ukázku použijeme kinematickou dvojici (viz schéma na obr.11) s počátkem a zároveň generálním referenčním bodem (tzv. rámem) 0, na kterém je upevněn motor s označením 1 a motor s označením 2. Jak je vidět ze schématu na obr.11, tato kinematická dvojice rotuje podle osy x a tomu se podle tabulkových hodnot podřídí zápis ve směrové submatici (viz matice S_{21} ve vzorci (1)). Vzdálenost středů motorů si stanovíme na 1m. Jelikož se naše domnělá konstrukce nachází ve vodorovné poloze, je námi zvolená hodnota pro tuto ukázku na pozici osy x ve vektoru vzdálenosti středů r_{21} (viz vzorec (1)). Abychom mohli matici T_{21} (viz vzorec (1)) prohlásit za transformační, musí být doplněna do homogenních souřadnic. Tyto hodnoty jsou nulové kromě hodnoty ve 4. řádku, 4. sloupci. Tato hodnota je využívána zejména ve strojnictví a určuje zvětšení (popř. zmenšení), ale v našem případě nic takového využívat nepotřebujeme a proto tuto hodnotu ponecháme 1, tedy v původní velikosti (1:1).

$$T_{21} = \begin{matrix} 1 & 0 & 0 & 1 \\ 0 & \cos\varphi_{21} & -\sin\varphi_{21} & 0 \\ 0 & \sin\varphi_{21} & \cos\varphi_{21} & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \rightarrow S_{21} = \begin{matrix} 1 & 0 & 0 \\ 0 & \cos\varphi_{21} & -\sin\varphi_{21} \\ 0 & \sin\varphi_{21} & \cos\varphi_{21} \end{matrix}, \quad r_{21} = \begin{matrix} 1 \\ 0 \\ 0 \end{matrix} \qquad (1)$$

Zápis zobecněné notace Denavitovi-Hartenbergovi transformační matice můžete vidět níže (viz vzorec (2). Kde S značí směrovou matici a koeficienty *x*, *y* a *z* uvedené v pravém sloupci, značí posunutí středu systému vůči předchozímu systému.

$$T_{ba} = \begin{matrix} & S & & \begin{matrix} x \\ y \\ z \end{matrix} \\ 0 & 0 & 0 & 1 \end{matrix} \qquad (2)$$



obr.11 Schéma kinematické dvojice

3.2.2. Analýza dynamiky

Pro rovnoměrný a regulovaný pohyb však nestačí znát možnosti pohybu kinematického řetězce (robotické paže). Pro pozdější získání popisu Dynamiky systému potřebujeme znát momenty setrvačnosti jednotlivých segmentů robotické paže. Dynamika soustavy se vyjádří pohybovou rovnicí v maticovém tvaru (viz rovnice (4)), kde matici dynamiky D vyjádříme z kinetické energie soustavy a prvky matice G z potenciální energie soustavy. Hodnoty se získávají kvůli náročnosti na výpočet za pomoci softwaru jako je například Matlab, ve kterém je pro tyto výpočty určen toolbox SimMechanics.

Obecný tvar matice setrvačnosti je popsán níže (viz vzorec (3)). Kde na hlavní diagonále koeficienty I_{xx}, I_{yy} a I_{zz} označují momenty setrvačnosti v hlavních osách. A ostatní koeficienty D_{xy}, D_{xz}, D_{yx}, D_{yz}, D_{zx}, D_{zy} označují deviační momenty, jejichž hodnoty jsou nenulové tehdy, pokud těžiště tělesa neprochází jednou z hlavních os a není kolineární jednou z hlavních os.

$$I = \begin{matrix} I_{xx} & -D_{yx} & -D_{xz} \\ -D_{yx} & I_{yy} & -D_{yz} \\ -D_{zx} & -D_{zy} & I_{zz} \end{matrix} \quad (3)$$

$$\tau = D\ q\ *q + C\ q,q\ + G(q) \quad (4)$$

3.3. Požadavky na knihovnu RoboticControl

Tyto požadavky byly určeny na základě analýzy z kapitol 3.1 a 3.2, jejímž účelem je vysvětlit proč byly zvoleny právě tyto funkce a atributy. Dalším krokem byla konzultace s Ing. Janem Kaulerem, Ph.D., který se problematikou robotických paží zabývá.

Seznam požadovaných jednotlivých funkcí:

- Inicializace
- Kalibrace
- Pohyb v rychlostním módu na pozici
- Pohyb v pozičním módu na pozici
- Pohyb v momentovém módu na pozici
- Nastavení akcelerace
- Nastavení decelerace
- Nastavení hodnot krajních a inicializační pozic
- Zjištění aktuální pozice

Seznam požadovaných jednotlivých atributů:

- List jednotlivých kloubů
    - Matice setrvačnosti
    - Transformační matice
    - Vzdálenost od předchozího kloubu
    - Vlastnosti pohybu kloubu
- Jednotky systému

3.4. Funkční požadavky na knihovnu RoboticControl

Inicializace je funkce, která slouží k nastavení motorů do úvodních poloh. Robotická paže se tedy dostane do úvodní pozice. Funkci Inicializace bude vždy předcházet funkce kalibrace nebo funkce nastavení hodnot krajních pozic proto, aby motory nepřekročily konstrukční maxima a nedošlo k poškození robotické paže.

Kalibrace je funkce pro robotické paže, které po vypnutí neudrží v paměti kontroléru svou aktuální pozici. Motory po spuštění této funkce se dopraví do obou krajních mezí., kde například sepnou mikrospínač. Po prvním sepnutí se odečte hodnota ze senzoru pozice motoru a změní se směr otáčení motoru na opačnou stranu. Po sepnutí druhého spínače se opět odečte hodnota ze senzoru pozice a motor se zastaví. Na základě získaných hodnot se

dozvíme hodnotu krajních mezí, ale jsme také schopni vypočítat střed (polohu pro inicializaci). Po získání těchto hodnot můžeme zavolat funkci Nastavení hodnot krajních pozic.

Funkce pohybu v rychlostním módu na pozici bude zajišťovat pohyb v rychlostním módu. Pro tuto funkci bude nutné, aby byly naplněny atributy objektu kloub transformační matice a matice setrvačnosti. V opačném případě tato funkce bude nefunkční, protože bez těchto atributů není možné se přesně pohybovat, jelikož pro tuto funkcionalitu je nutné, aby byl vypočítán průběh rychlostní křivky. Na jejím základě se v dané okamžiky budou nastavovat konkrétní hodnoty rychlosti, než se dosáhne cílové polohy.

Další funkcí je pohyb v pozičním módu na pozici. Tato funkce bude, na základě přijatých hodnot o nové pozice, vypočítávat diferenci mezi aktuální a novou pozicí. A o tuto hodnotu se motory posunou.

Pohyb v momentovém módu na pozici je další z funkcí, kterou bez nenulových atributů objektu kloub (transformačních matic a matic setrvačnosti), nebude možno používat. Funkce na základě transformačních matic a matic setrvačnosti vypočítá průběh momentové křivky. Tyto vypočtené hodnoty budou v dané okamžiky nastavovány.

Funkce nastavení akcelerace bude nastavovat hodnotu zrychlení pro akceleraci jednotlivých motorů či celé soustavy.

Nastavení decelerace je funkce, která by měla měnit hodnotu zrychlení pro decelerace buď u jednotlivých motorů, nebo u celé soustavy.

Funkce nastavení maximální rychlosti, bude měnit rychlost které bude možno dosáhnout při vykonávaném pohybu.

Funkce nastavení hodnot krajních a inicializačních pozic bude zajišťovat nastavení krajních hodnot, které motor (respektive robotická paže) nebude smět překročit, aby nedošlo k poškození. Dále bude zajišťovat nastavení inicializačních hodnot pozic.

Zjištění aktuální pozice je funkce, kterou využije každá z robotických paží. Tato funkce bude vracet hodnotu pozice každého z motorů. Hodnotou je myšlena hodnota iterátoru jednotlivých kroků motoru.

Obě třídy (robotická paže a kloub) budou obsahovat atributy jako je matice setrvačnosti a transformační matice. Dále budou atributy vzdálenost od předchozího kloubu, vlastnosti pohybu a list jednotlivých kloubů. Ačkoliv by někteří mohli namítat, že atributy jako

vzdálenost od předchozího kloubu a vlastnosti pohybu jsou zde duplicitně a lze je odvodit z transformačních matic. Jde zde jen o pomocné atributy, které mají urychlit orientaci.

Atribut jednotky systému je stanoven pro určení, v jakých jednotkách jednotlivé atributy jsou.

3.5. Alternativy řešení návrhových vzorů

Na základě funkcí lze již vybrat vhodné řešení návrhovými vzory. Řešení se dá rozdělit na dvě části. Na část kdy řešení interfacu robotických paží (knihovna RoboticControl) můžeme řešit jako samostatnou část. A na samotné řešení motorů (knihovna Motor), kdy robotickou paži tvoří taktéž samostatnou část.

Pro řešení knihovny Motor je vhodný návrhový vzor Adapter, o kterém jsem se zmínil v kapitole 2.4.1.1, kde Adaptee je knihovna pro ovládání motoru dodávána výrobcem. Adapter je knihovna, kterou vytvoří uživatel na základě již předefinovaných metod v ITarget, které jsou přepisovány na základě vztahu Realizace.

Pro knihovnu RoboticArm je vhodným řešením návrhový vzor Composite, který uvádím v kapitole 2.4.1.3 . Composite tedy bude RoboticArm, která bude závislá na IComponent (IRoboticControl). RoboticArm bude implementovat funkce IRoboticControl . A třída RoboticArm bude mít List kloubů respektive třídy Joint.

Třída Joint je závislá na Interfacu Motor a je tedy Client knihoven Motor.

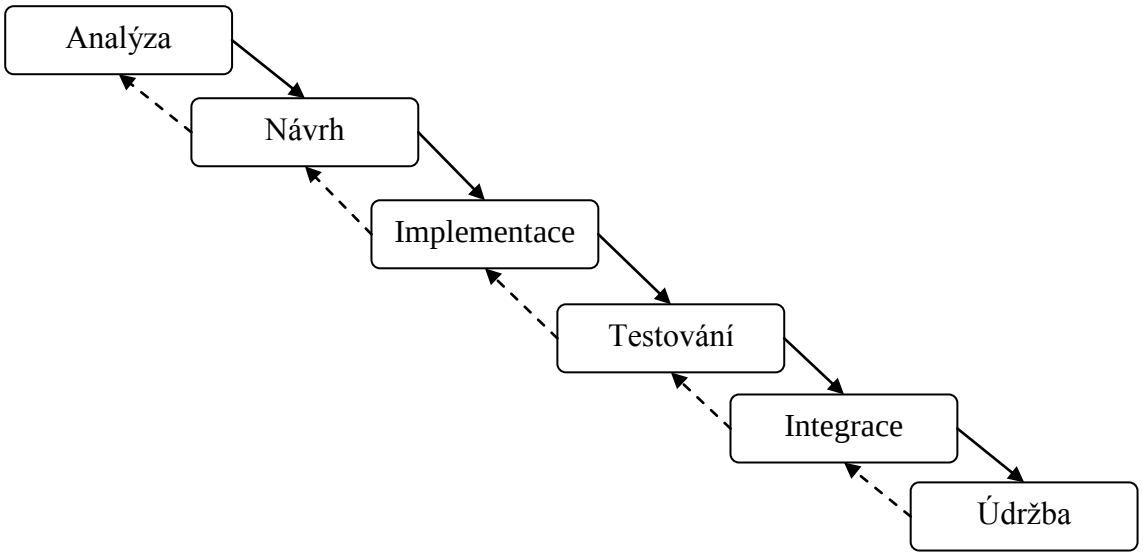
3.6. Metodika vývoje softwaru

Pro vývoj tohoto softwaru jsem využil metodiku Vodopádového modelu s úpravou zpětných vazeb tzv. Sašimi model. Zpětné vazby nejsou jedinou vlastností Sašimi modelu, další vlastností modelu je, že se některé fáze vývoje mohou překrývat.

Vodopádový model se skládá z

1) Analýzy požadavků
2) Návrhu
3) Implementace
4) Testování (validace)
5) Integrace (instalace)
6) Údržby

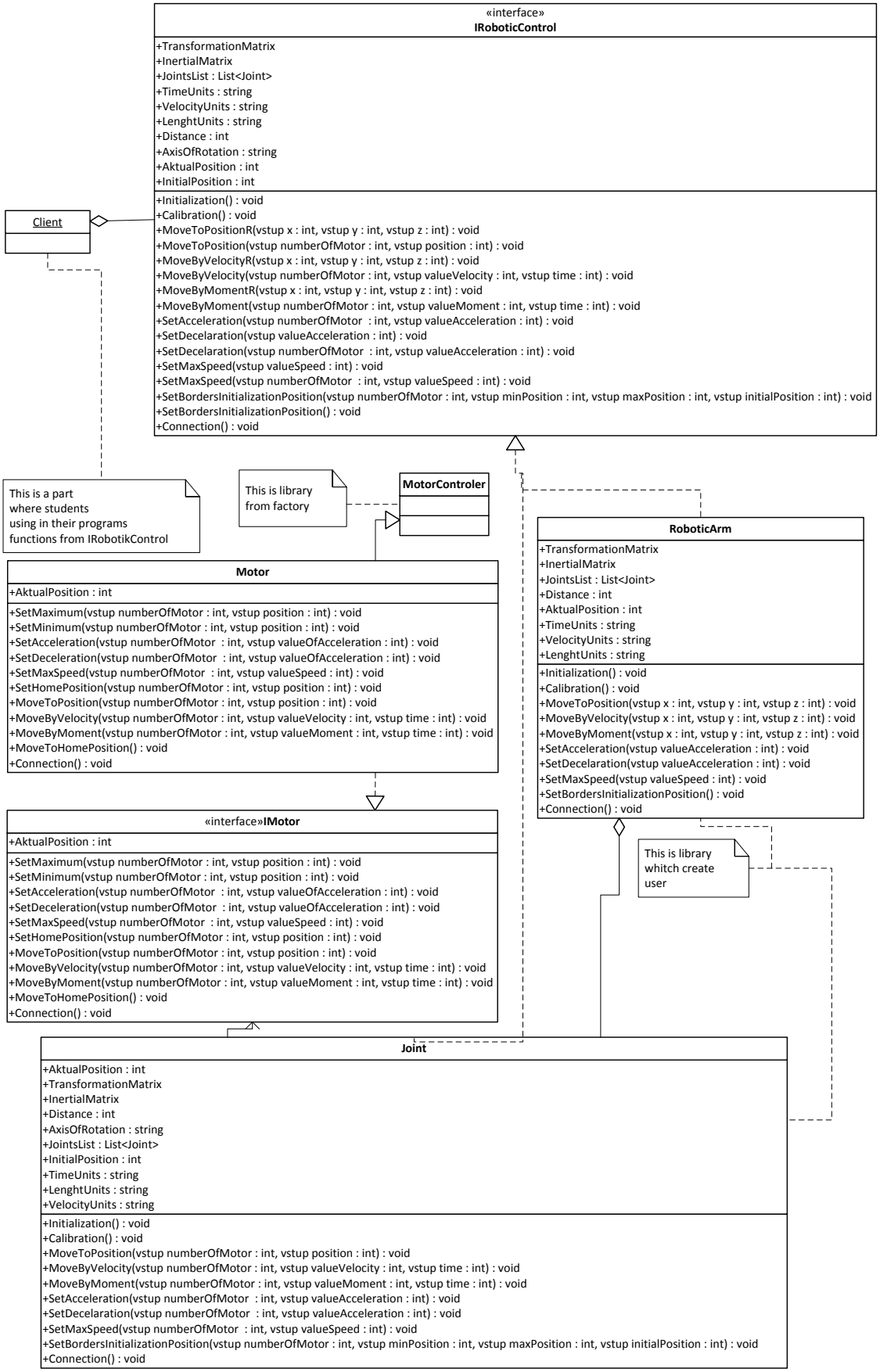
Vodopádový model je sekvenční vývojový proces a je velmi často kritizován za nedostatky. Jako například, když zadavatel v průběhu vývoje mění požadavky na vyvíjený produkt a programátor tyto změny zahrne až v novém běhu tohoto modelu. Proto se tento model používá více s modifikacemi jako je například Sašimi model (se zpětnými vazbami a překrývajícími se částmi).



obr.12 Diagram Vodopádového modelu s čárkovaně jsou označené zpětné vazby z Sašimi modelu

4. Vlastní řešení softwaru

Samotné řešení softwaru se zdá být triviálním. Avšak při vývoji nastává plno úskalí, která se nedají zcela jednoduše řešit. Přímá implementace na základě diagramu na obr.13 je patrně správnou volbou. Ale nesmíme zapomínat na základní pravidla vývojového procesu, která by se měla dodržovat. Jako je například zpětné testování a ověřování, že funkčnost splňuje námi předem zadané požadavky. Dále je nutné ověřovat, zda návrh není chybný a není potřeba vytvořit obecnější návrh, který by obsahoval kompletní sadu tříd, funkcí a atributů.



obr.13 Úplný UML diagram vlastního řešení problematiky robotické paže

4.1. Návrh jednotlivých komponent

Tato kapitola se bude zabývat obecným popisem všech tříd, které jsou vidět v úplném návrhu na diagramu na obr.13

4.1.1. Vlastnosti a možnosti rozhraní IMotor

Jak uvádí web Microsoft developer network „*rozhraní obsahuje pouze popisy metod*“ [**4**]. Podle diagramu na obr.14 byly předefinovány všechny předpokládané funkce a atributy.

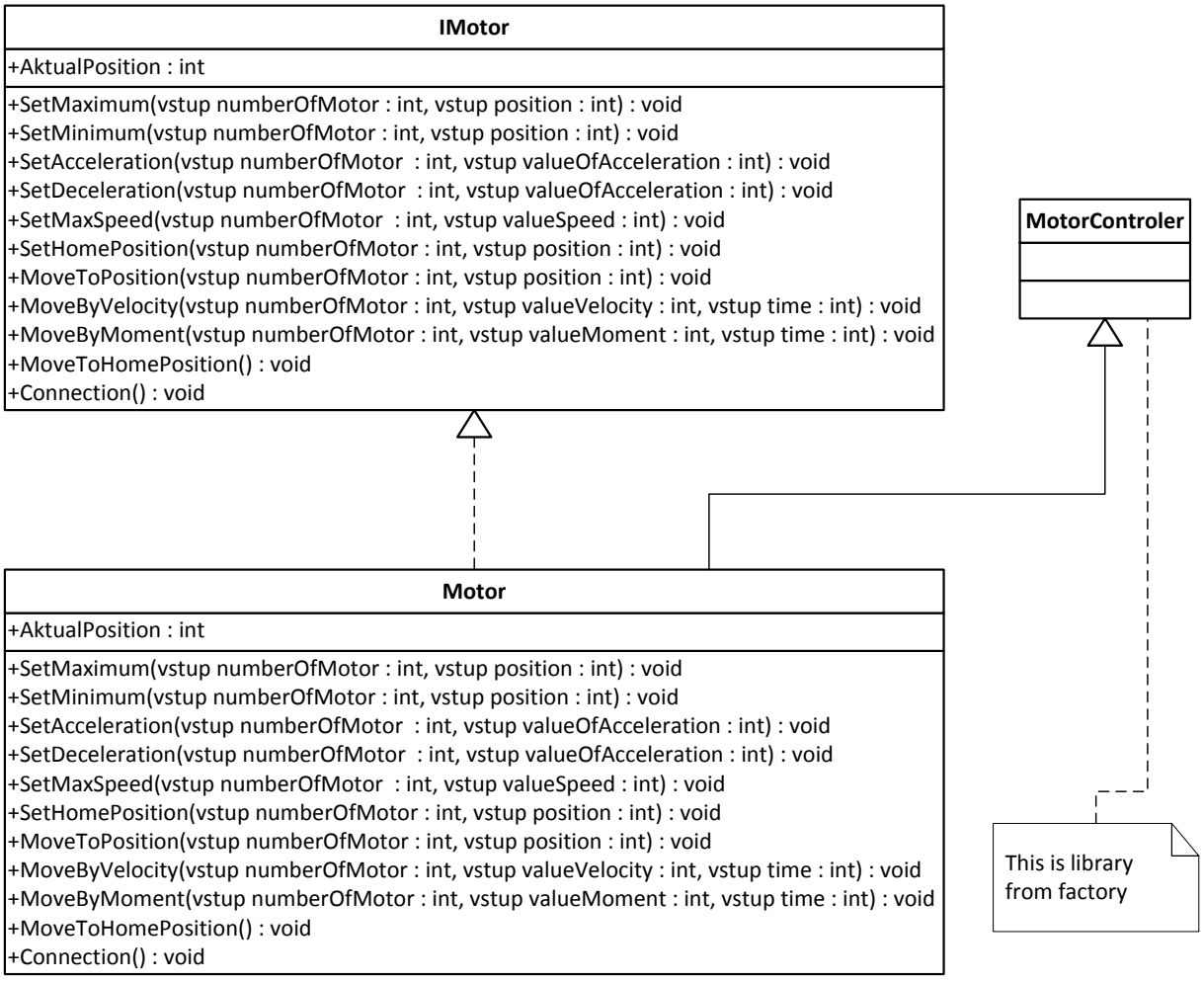
Atributy

- *AktualPosition* vrací číslo ze senzoru pozice motoru.

Funkce

- *SetMaximum* a *SetMinimum* jsou funkce, které nastavují krajní hranice pohybu motorů. Tyto funkce jsou podporovány většinou kontrolérů motorů. Některé kontroléry mají i vlastní ochranu proti přechodu přes tuto hranici.
  Obě tyto funkce mají tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *position* je číselný parametr určující jednu z krajních pozic.
- *SetAcceleration* a *SetDeceleration* jsou funkce nastavující zrychlení, kterým motor bude akcelerovat nebo decelerovat. U některých motorů je tato hodnota spojena v jednu.
  Funkce mají tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueAcceleration* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.
- *SetMaximumSpeed* je funkce nastavující maximální rychlost, které lze dosáhnout při pohybu.
  Funkce má tyto parametry
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueSpeed* je číselný parametr určující hodnotu maximální rychlosti

- *SetHomePosition* je funkce nastavující hodnotu pozice, na kterou se motor otočí po inicializaci.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *position* je číselný parametr určující pozici pro inicializaci.
- *MoveToPosition* je funkcí pohybu, které po zadání cílové pozice, otočí motor na danou pozici.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *position* je číselný parametr určující cílovou pozici.
- *MoveByVelocity* je funkce, která pohybuje motorem danou rychlostí po určitý čas.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueVelocity* je číselný parametr určující cílovou rychlost motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveByMoment* funkce pohybuje motor momentem po danou dobu.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueMoment* je číselný parametr určující moment motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveToHomePosition* je funkce, která pohne motorem na inicializační pozici. Tato funkce nemá žádné vstupní parametry.
- *ReadInputs* je funkce načítající vstupy, například spínače. Funkce nemá žádné vstupní parametry, ale bude vracet pole vstupů.
- *Connection* je funkce, která zajišťuje připojení ke kontroléru. Funkce nemá žádné vstupní parametry.



obr.14 UML diagram s detailem části Motors

4.1.2. Vlastnosti a možnosti rozhraní IRoboticControl

Jak jsem již citoval v kapitole o IMotor 4.1.1, v rozhraní (interface) jsou jen popisky metod. Z diagramu obr.15 UML diagram s detailem části RoboticArm je vidět, že v rozhraní IRoboticControl jsou definovány všechny funkce a atributy.

Atributy

- *TransformationMatrix* je atribut s transformační maticí. Tato matice obsahuje hodnoty strukturované, jak bylo zmíněno v kapitole 3.2.1.
- *InertialMatrix* je atributem, který obsahuje matici hodnot, jak bylo zmíněno v kapitole 3.2.2.
- *JointsList* obsahuje seznam tříd Joint.
- *TimeUnits* je textový atribut, který udává v jakých jednotkách času jednotlivé motory (popřípadě celá soustava) pracuje.
- *VelocityUnits* je textový atribut, který udává v jakých jednotkách rychlosti jednotlivé motory (popřípadě celá soustava) pracuje.
- *LenghtUnits* je textový atribut, který udává používané jednotky délky.
- *Distance* je atribut, který obsahuje číselnou hodnotu vzdálenosti od předchozího kloubu.
- *AxisOfRotation* tento textový atribut nese informaci o tom, podle jaké osy se motor otáčí.
- *AktualPosition* vrací číslo ze senzoru pozice motoru.
- *InitialPosition* vrací číslo inicializační pozice.

Funkce

- *Initialization* je funkce, která robotickou paži (každý z motorů) otáčí na inicializační pozici.

  Tato funkce nemá žádné parametry.
- *Calibration* je funkce, která hledá krajní polohy motorů a využívá při tom funkci *ReadInputs*. Funkce bude následovně volat funkci *SetBordersInitializationPosition*, aby nastavila krajní hranice a inicializační polohu.

  Tato funkce nemá žádné parametry.

- *MoveToPosition* je funkcí pohybu, která po zadání cílové pozice, otočí motor na danou pozici.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *position* je číselný parametr určující cílovou pozici.
- *MoveToPositionR* je funkce pohybu, která pohybuje celou robotickou paží na cílovou pozici podle zadání koncového bodu robotické paže v kartézských souřadnicích.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce
- *MoveByVelocity* je funkce, která pohybuje motorem danou rychlostí po určitý čas.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueVelocity* je číselný parametr určující cílovou rychlost motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveByVelocityR* je funkce pohybu, která pohybuje celou robotickou paží v rychlostním módu na cílovou pozici podle zadání koncového bodu robotické paže v kartézských souřadnicích.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce

- *MoveByMoment* je funkce pohybující motorem momentem po danou dobu.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueMoment* je číselný parametr určující moment motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveByMomentR* je funkce pohybu, která pohybuje celou robotickou paží v momentovém módu na cílovou pozici podle zadání kartézských souřadnic koncového bodu.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce
- *SetAcceleration* a *SetDeceleration* jsou funkce nastavující zrychlení, kterým motor bude akcelerovat nebo decelerovat. U některých motorů je tato hodnota spojena v jednu.
  Funkce mají tyto parametry v přetížení pro třídu Joint.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueAcceleration* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.

  Funkce má v přetížení pro třídu RoboticArm tento parametr.
  - *valueAcceleration* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.
- *SetMaxSpeed* jen funkce nastavující maximální rychlost, které motor může při pohybu dosáhnout.
  Funkce má tyto parametry v přetížení pro třídu Joint.
  - *numberOfMotor* je číselný parametr označující index motoru.

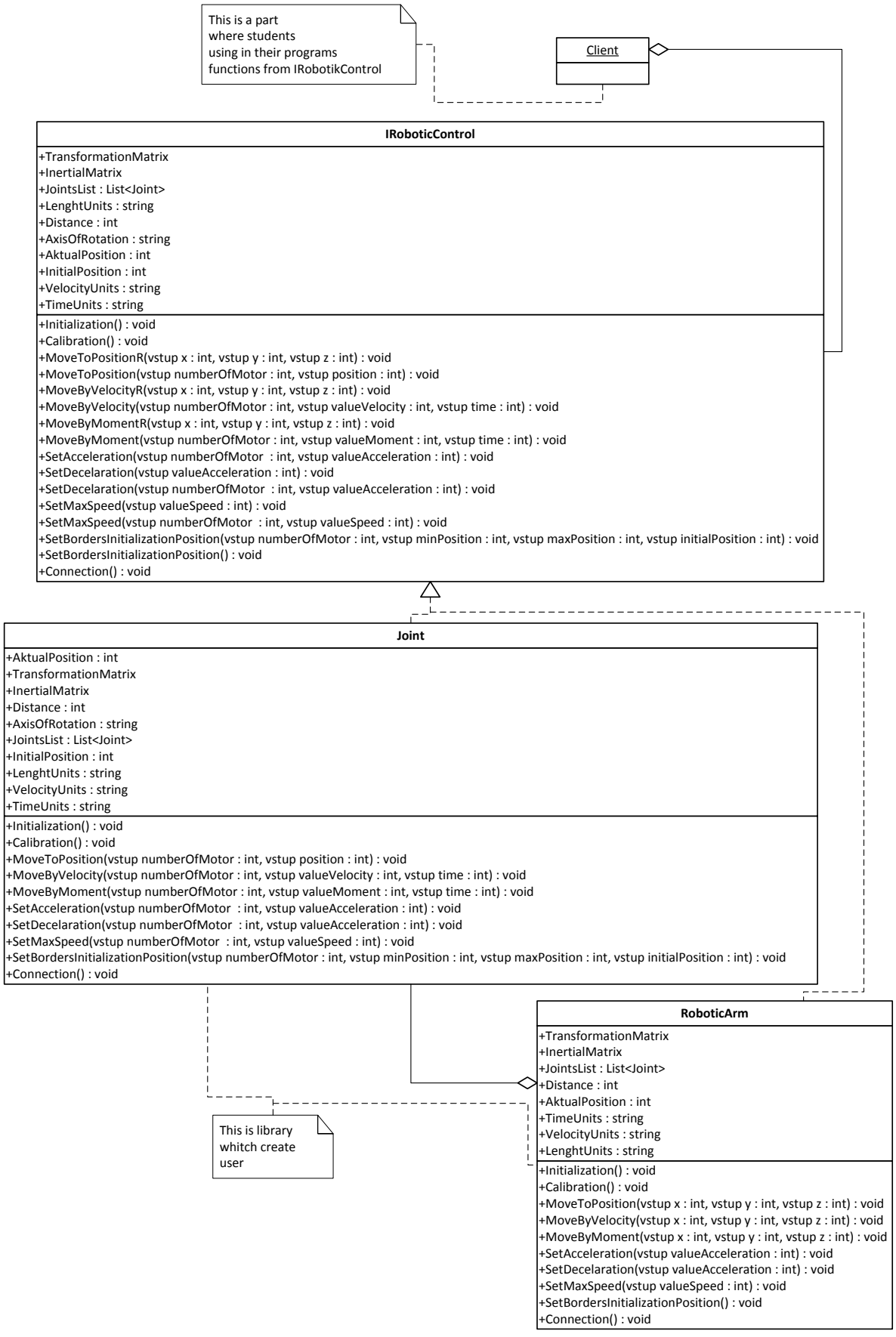
- *valueSpeed* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.

Funkce má v přetížení pro třídu RoboticArm tento parametr.

- *valueSpeed* je číselný parametr určující hodnotu maximální rychlosti.

- *SetBordersInitializationPosition* je funkce nastavující krajní pozice a inicializační pozici. Funkce je využívána buďto funkcí anebo ji uživatel zavolá samostatně. Funkce má tyto parametry v přetížení pro třídu Joint.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *minPosition* je číselný parametr určující pozici minimální krajní pozice.
  - *maxPosition* je číselný parametr určující pozici maximální krajní pozice.
  - *initialPosition* je číselný parametr určující pozici pro inicializaci.

  Funkce nemá v přetížení pro třídu RoboticArm žádné parametry.
- *ReadInputs* je funkce, která nemá žádné vstupní parametry a vrací pole číselných hodnot vstupů.
- *Connection* je funkce, která zajišťuje připojení ke kontroléru. Funkce nemá žádné vstupní parametry.



obr.15 UML diagram s detailem části RoboticArm

4.1.3. Vlastnosti a možnosti třídy Motor

Třída Motor bude implementovat rozhraní (interface) IMotor a bude specializací knihovny pro ovládání vytvořenou od výrobce.

Funkce a atributy, které nebude možno implementovat, protože danou funkcionalitu nemá ovládací knihovna vytvořená výrobcem, bude dle konvencí obsahovat příkaz pro výjimku (NotImplementedException).

4.1.4. Vlastnosti a možnosti třídy Joint

Třída Joint je jednou z implementací rozhraní (interface) IRoboticControl. Dále je závislá na rozhraní (interface) IMotor, jak je vidět z UML diagramu obr.13. Vytváří jednotlivé položky motorů (respektive kloubů) v robotické paži, takže je součástí listu v třídě RoboticArm, jak je vidět z diagramu obr.13.

Atributy

- *TransformationMatrix* je atribut s transformační maticí pro konkrétní kloub.
- *InertialMatrix* je atributem maticí setrvačnosti pro konkrétní kloub
- *JointsList* obsahuje seznam tříd Joint se sousedními klouby.
- *TimeUnits* je textový atribut, který udává, v jakých jednotkách času jednotlivé motory pracují.
- *VelocityUnits* je textový atribut, který udává, v jakých jednotkách rychlosti jednotlivé motory pracují.
- *LenghtUnits* je textový atribut, který udává používané jednotky délky.
- *Distance* je atribut, který obsahuje číselnou hodnotu vzdálenosti od předchozího kloubu.
- *AxisOfRotation* tento textový atribut nese informaci o tom, podle jaké osy se motor otáčí.
- *AktualPosition* vrací číslo ze senzoru pozice motoru.
- *InitialPosition* vrací číslo inicializační pozice.

Funkce

- *Initialization* je funkce, která každý z motorů otáčí na inicializační pozici.
  Tato funkce nemá žádné parametry.
- *Calibration* je funkce hledá krajní polohy motorů a využívá při tom funkci *ReadInputs*. Funkce bude následovně volat funkci *SetBordersInitializationPosition*, aby nastavila krajní hranice a inicializační polohu.

  Tato funkce nemá žádné parametry.
- *MoveToPosition* je funkcí pohybu, které po zadání cílové pozice, otočí motor na danou pozici.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *position* je číselný parametr určující cílovou pozici.
- *MoveToPositionR* tato funkce není v této třídě implementována.
- *MoveByVelocity* je funkce, která pohybuje motorem danou rychlostí po určitý čas.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueVelocity* je číselný parametr určující cílovou rychlost motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveByVelocityR* tato funkce není v této třídě implementována.
- *MoveByMoment* je funkce pohybující motorem momentem po danou dobu.
  Funkce má tyto parametry.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueMoment* je číselný parametr určující moment motoru.
  - *time* je číselný parametr určující dobu, po kterou se bude motor pohybovat.
- *MoveByMomentR* tato funkce není v této třídě implementována.
- *SetAcceleration* a *SetDeceleration* jsou funkce nastavující zrychlení, kterým motor bude akcelerovat nebo decelerovat. U některých motorů je tato hodnota spojena

v jednu.

Funkce mají tyto parametry.

- *numberOfMotor* je číselný parametr označující index motoru.
- *valueAcceleration* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.

- *SetMaxSpeed* jen funkce nastavující maximální rychlost, které motor může při pohybu dosáhnout.

  Funkce má tyto parametry.

  - *numberOfMotor* je číselný parametr označující index motoru.
  - *valueSpeed* je číselný parametr určující hodnotu maximální rychlosti.

- *SetBordersInitializationPosition* je funkce nastavující krajní pozice a inicializační pozici. Funkce je využívána buďto funkcí anebo ji uživatel zavolá samostatně.

  Funkce má tyto parametry.

  - *numberOfMotor* je číselný parametr označující index motoru.
  - *minPosition* je číselný parametr určující pozici minimální krajní pozice.
  - *maxPosition* je číselný parametr určující pozici maximální krajní pozice.
  - *initialPosition* je číselný parametr určující pozici pro inicializaci.

- *ReadInputs* je funkce, která nemá žádné vstupní parametry a vrací pole číselných hodnot vstupů.
- *Connection* je funkce, která zajišťuje připojení ke kontroléru. Funkce nemá žádné vstupní parametry.

4.1.5. Vlastnosti a možnosti třídy RoboticArm

RoboticArm je implementací rozhraní (interface) IRoboticControl a Agreguje třídu Joint. Třída RoboticArm vlastní seznam (List) kloubů (Joint) a jako jediná tedy implementuje atribut *JointsList*.

Atributy

- *TransformationMatrix* je atribut s celkovou transformační maticí, která obsahuje součin všech matic kloubů ze seznamu JointList.

- *InertialMatrix* je prázdným atributem.
- *JointsList* obsahuje seznam tříd Joint (kloubů) celé robotické paže.
- *TimeUnits* je textový atribut, který udává v jakých jednotkách času robotická paže pracuje.
- *VelocityUnits* je textový atribut, který udává v jakých jednotkách rychlosti robotická paže pracuje.
- *LenghtUnits* je textový atribut, který udává používané jednotky délky.
- *Distance* je atribut, který obsahuje číselnou hodnotu vzdálenosti od počátku.
- *AxisOfRotation* je prázdným atributem.
- *AktualPosition* vrací číslo ze senzoru pozice motoru
- *InitialPosition* vrací číslo inicializační pozice.

Funkce

- *Initialization* je funkce, která robotickou paži nastaví na inicializační pozici.
  Tato funkce nemá žádné parametry.
- *Calibration* je funkce hledající krajní polohy motorů a využívá při tom funkci *ReadInputs*. Funkce bude následovně volat funkci *SetBordersInitializationPosition,* aby nastavila krajní hranice a inicializační polohu.

  Tato funkce nemá žádné parametry.
- *MoveToPosition* tato funkce není v této třídě implementována.
- *MoveToPositionR* je funkce pohybu, která pohybuje celou robotickou paží na cílovou pozici podle zadání koncového bodu robotické paže v kartézských souřadnicích.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce

- *MoveByVelocity* tato funkce není v této třídě implementována.
- *MoveByVelocityR* je funkce pohybu, která pohybuje celou robotickou paží v rychlostním módu na cílovou pozici podle zadání koncového bodu robotické paže v kartézských souřadnicích.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce
- *MoveByMoment* tato funkce není v této třídě implementována.
- *MoveByMomentR* je funkce pohybu, která pohybuje celou robotickou paží v momentovém módu na cílovou pozici podle zadání kartézských souřadnic koncového bodu.
  Funkce má tyto parametry.
  - *x* je číselný parametr označující pozici na x ose s počátečním bodem v počátku kinematického řetězce
  - *y* je číselný parametr označující pozici na y ose s počátečním bodem v počátku kinematického řetězce
  - *z* je číselný parametr označující pozici na z ose s počátečním bodem v počátku kinematického řetězce
- *SetAcceleration* a *SetDeceleration* jsou funkce nastavující zrychlení, kterým budou všechny motory akcelerovat nebo decelerovat. Funkce projde celý JointsList a u každé položky nastaví stejnou hodnotu zrychlení.

  Funkce má tento parametr.
  - *valueAcceleration* je číselný parametr určující hodnotu zrychlení pro akceleraci nebo deceleraci.
- *SetMaxSpeed* je funkce nastavující maximální rychlost, kterou každý z motorů může při pohybu dosáhnout. Funkce projde celý JointsList a u každé položky nastaví stejnou

hodnotu maximální rychlosti.

Funkce má v přetížení pro třídu RoboticArm tento parametr.

- *valueSpeed* je číselný parametr určující hodnotu maximální rychlosti.

- *SetBordersInitializationPosition* je funkce nastavující krajní pozice a inicializační pozici. Funkce je využívána buďto funkcí anebo ji uživatel zavolá samostatně. Funkce má tyto parametry v přetížení pro třídu Joint.
  - *numberOfMotor* je číselný parametr označující index motoru.
  - *minPosition* je číselný parametr určující pozici minimální krajní pozice.
  - *maxPosition* je číselný parametr určující pozici maximální krajní pozice.
  - *initialPosition* je číselný parametr určující pozici pro inicializaci.

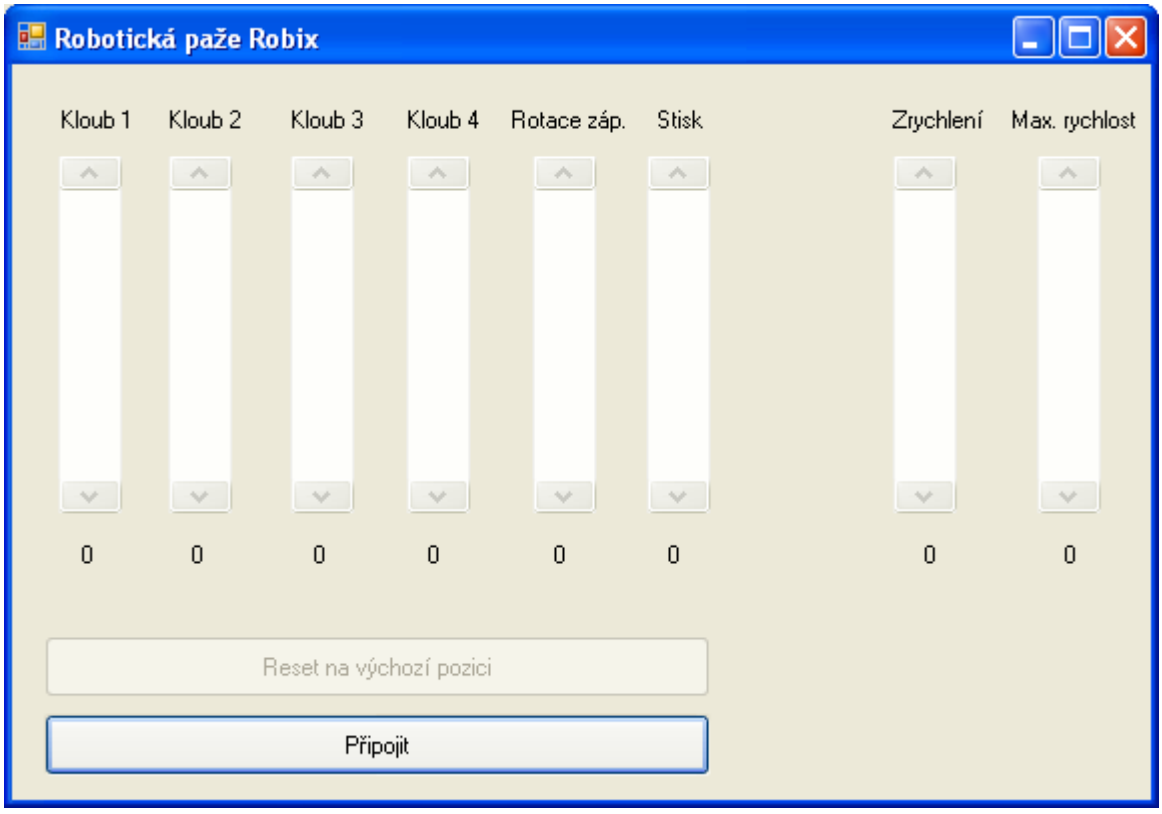
  Funkce nemá v přetížení pro třídu RoboticArm žádné parametry.
- *ReadInputs* je funkce, která nemá žádné vstupní parametry a vrací pole číselných hodnot vstupů.
- *Connection* je funkce, která zajišťuje připojení ke kontroléru. Funkce projde celý JointsList a u každé provede připojení. Funkce nemá žádné vstupní parametry.

5. Testování návrhu

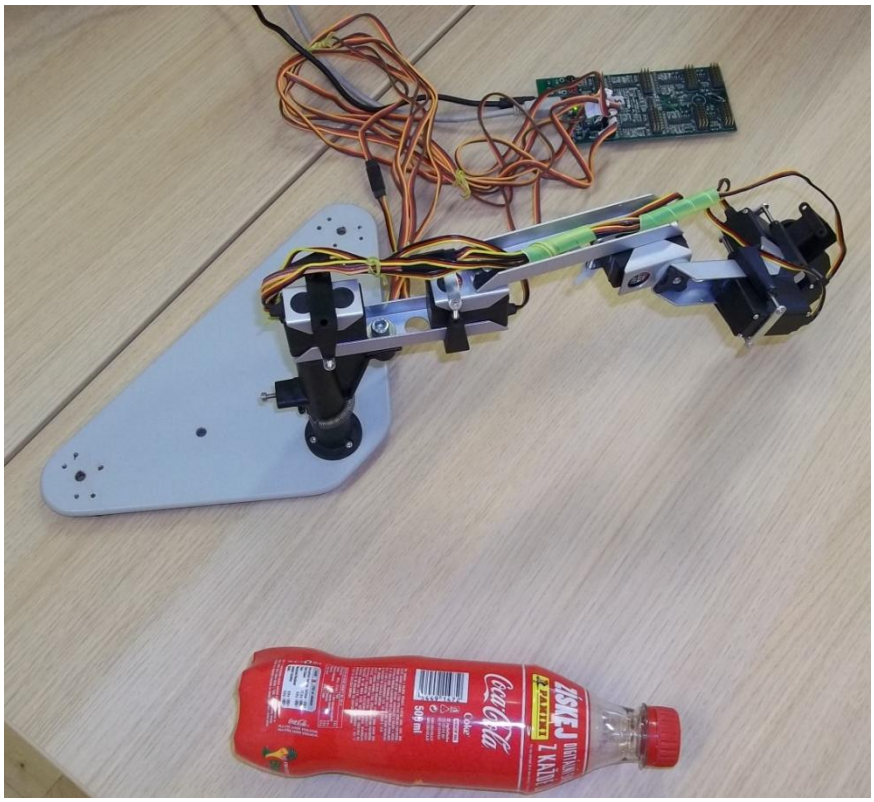
Návrh jsem otestoval na robotické paži Robix (viz kapitola 2.3.1).

Implementoval jsem všechny funkce, které tato robotická paže podporuje. Funkce, které nebyly podporovány, jsem nechal prázdné (viz příloha složka RoboticArm). Robotická paže nemá funkci pro získání aktuální pozice motoru. Proto byla tato hodnota nastavena virtuálně a po případném provedení inicializace, se tato hodnota nastaví na počátek (na pozici 0).

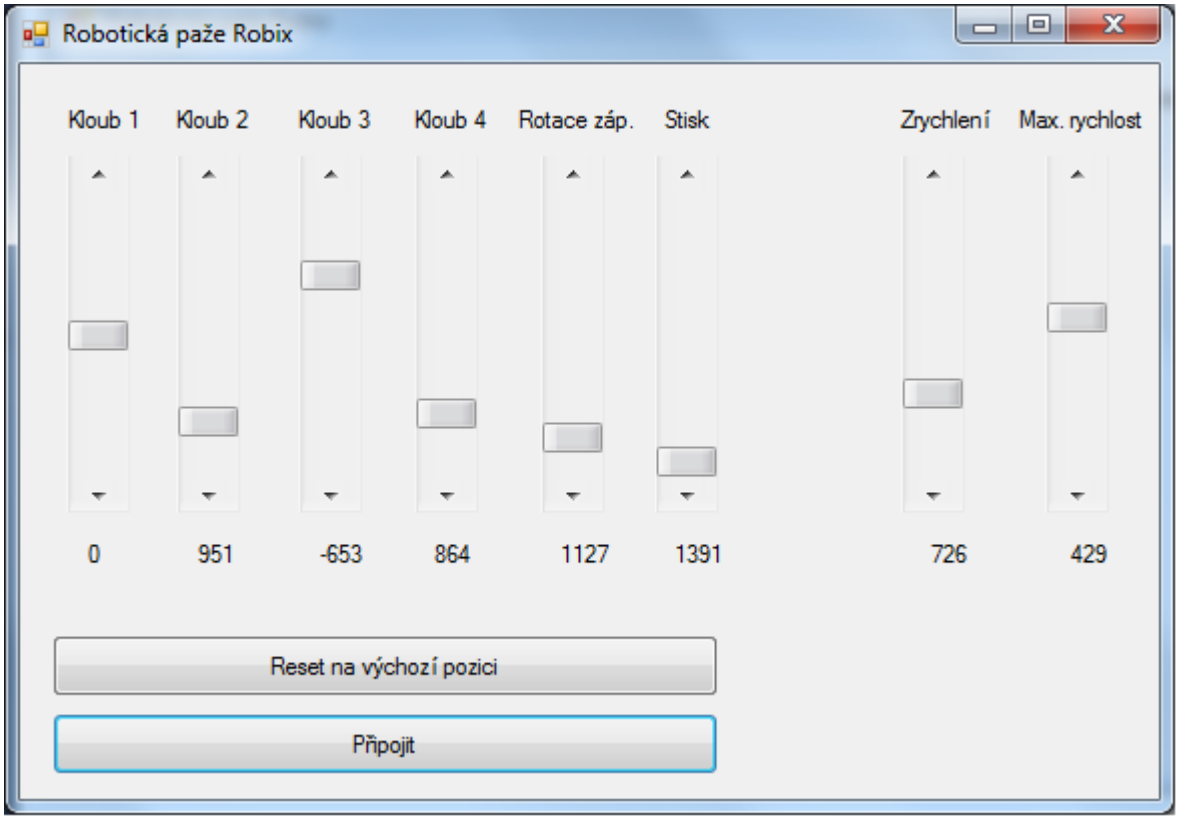
Vytvořil jsem základní grafické uživatelské rozhraní (viz obr. 16), na kterém jsem otestoval připojení se ke kontroléru, nastavení maximální rychlosti a zrychlení. Nastavení těchto parametrů jsem vyzkoušel nastavením motorů na určité pozice a následným spuštěním funkce inicializace na výchozí pozici. Pokus dopadl úspěšně viz obrázková dokumentace níže.



obr. 16 Grafické uživatelské rozhraní (GUI) před připojením ke kontroléru

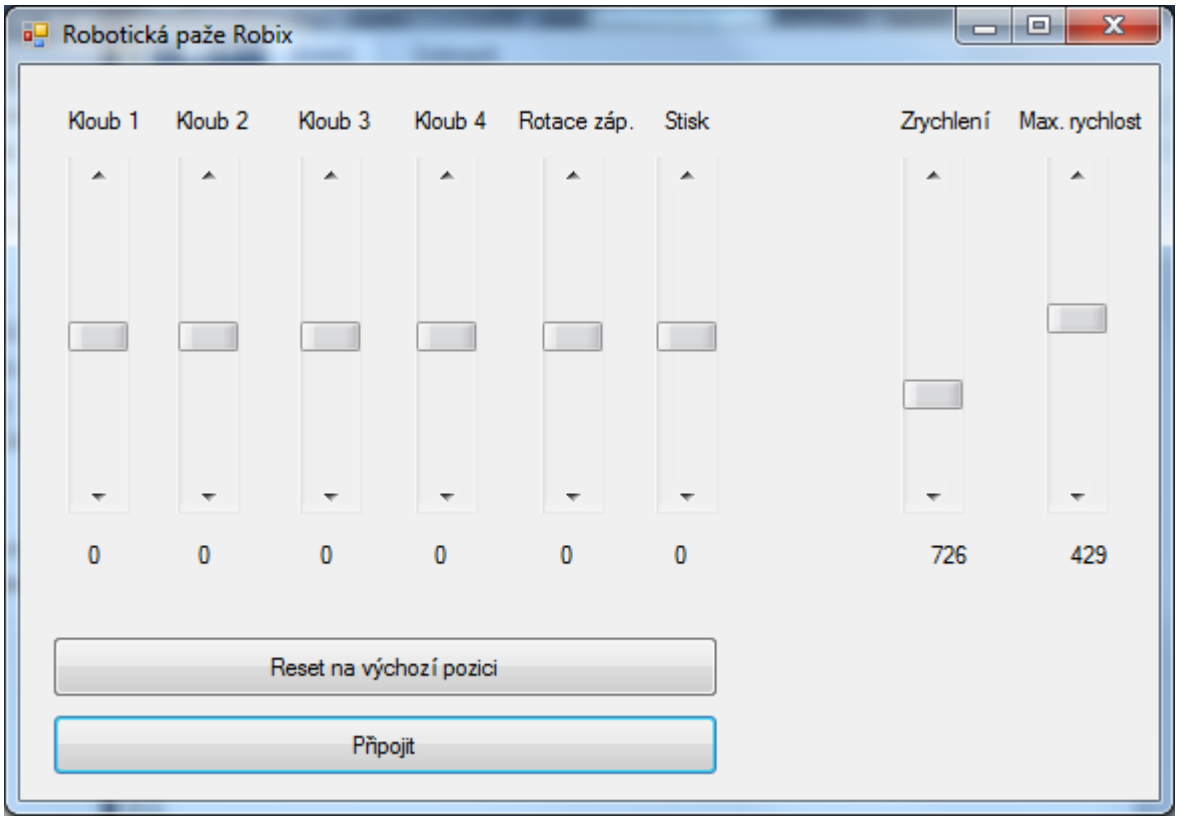


obr. 17 Robotická paže Robix po připojení kontroléru a zavolání inicializační funkce

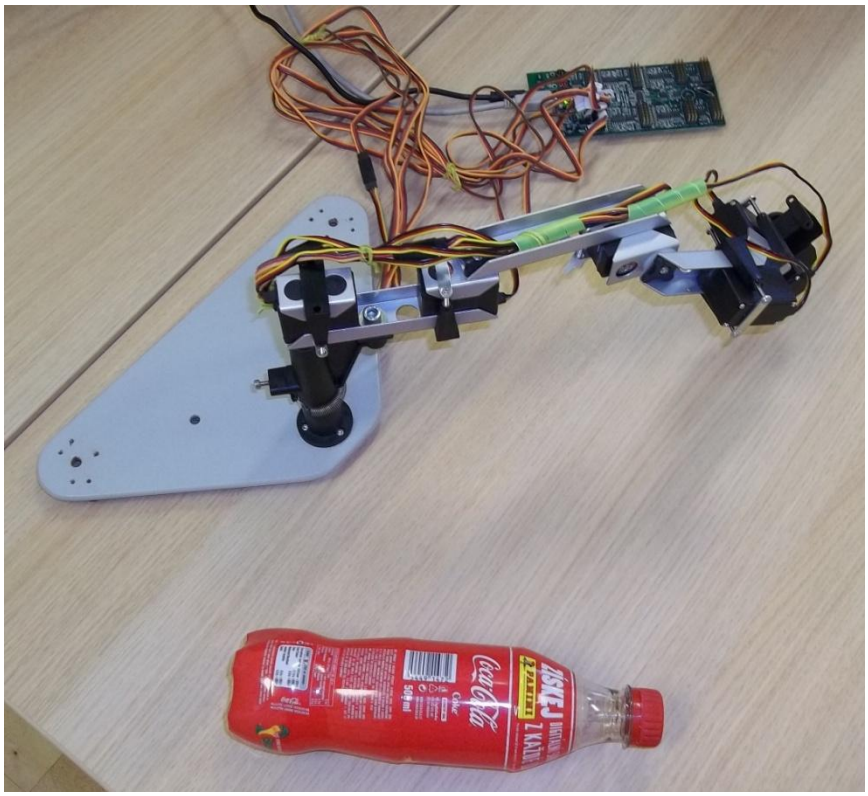


obr. 18 Nastavení pozic, zrychlení a maximální rychlosti v GUI

obr. 19 Robotická paže v cílové pozici



obr. 20 GUI po stisknutí tlačítka „Reset na výchozí pozici“



obr. 21 Robotická paže Robix v inicializační pozici po stisknutí tlačítka „Reset na výchozí pozici“ v GUI

6. Diskuze

Výsledek této bakalářské práce je jen jednou z částí velkého celku. Další práce (bakalářské, diplomové nebo jiné) by měly implementovat knihovny pro robotické paže podle této bakalářské práce. Avšak tato práce by neměla být podkladem jen pro studenty, ale i pro programátory, kteří se s touto problematikou setkají.

Fakulta biomedicínského inženýrství vlastní tři robotické paže (viz kapitola 2.3 a kapitoly 2.3.1 až 2.3.3). Ke dvěma z nich je nyní potřeba implementovat třídy, ze kterých se následně vytvoří knihovny pro ovládání.

Při konzultacích s Ing. Janem Kaulerem, Ph.D. se zjistila potřeba práce (bakalářské, diplomové nebo jiné) na téma .NET knihovny pro řešení inverzní kinematiky.

7. Závěr

Bakalářská práce měla za cíl vytvoření standartu, který by měl zjednodušit a sjednotit volání funkcí pro různé knihovny, které ovládaly rozdílné kontroléry pro robotické paže.

Implementace byla testována na robotické paži vytvořené z robotické stavebnice Robix, kterou jsem zmínil v kapitole 2.3.1. Jde pouze o obecnou implementaci určenou jen pro testování návrhu této bakalářské práce. Test ověřil, že návrh je funkční. Pro potvrzení komplexnosti a robustnosti návrhu je potřeba otestovat návrh tříd na dalších robotických pažích.

8. Seznam použité a citované literatury

1. BISHOP, J. *C# - návrhové vzory*. Překlad Jiří KOUTNÝ. Brno: Zoner Press, 2010. ISBN: 978-80-7413-076-2.

2. ING. DVOŘÁK, M. Flyweight Pattern. In: *Vysoká škola ekonomická-Katedra informačních technologií-Návrhové vzory (design paterns)* [online]. 2005 [cit. 2014-05-02]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory-Flyweight

3. ING. DVOŘÁK, M. Facade Pattern. In: *Vysoká škola ekonomická-Katedra informačních technologií-Návrhové vzory (design paterns)* [online]. 2005 [cit. 2014-02-05]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory-Facade

4. MICROSOFT. interface (Referenční dokumentace jazyka C#). In: *Microsoft developer network* [online]. [cit. 2014-05-13]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/87d83y5b.aspx

5. Patobiomechanika a Patokinesiologie. *Loketní kloub* [online]. [cit. 2014-03-05]. Dostupné z: http://biomech.ftvs.cuni.cz/pbpk/kompendium/anatomie/hk_paze_loket.php

6. CHRISTIAN NAGEL, BILL EVJEN, JAY GLYNN. *C# 2008 programuje profesiálně*. Praha: Computer Press, 2009. ISBN: 978-80-251-2401-7.

7. O počítačích, IT a internetu - Živě.cz. *Poznáváme C# a Microsoft.NET – 6. díl* [online]. 31. 12. 2004 [cit. 2014-05-08]. Dostupné z: http://www.zive.cz/clanky/poznavame-c-a-microsoftnet--6-dil/sc-3-a-121684/default.aspx

8. ČÁPKA, D. Adapter (wrapper). In: *devbook.cz programátorská sociální síť* [online]. 2014 [cit. 2014-05-08]. Dostupné z: http://www.devbook.cz/adapter-wrapper-navrhovy-vzor

9. ČÁPKA, D. Facade (fasáda). In: *devbook.cz programátorská sociální síť* [online]. 2014 [cit. 2014-05-08]. Dostupné z: http://www.devbook.cz/facade-navrhovy-vzor

10. SCHAUER, P. Kinematika hmotného bodu. In: *VUT Brno - Doplňkové materiály* [online]. 2007 [cit. 2014-04-27]. Dostupné z: http://fyzika.fce.vutbr.cz/doc/vyuka_schauer/kinematika_hmotneho_bodu.pdf

11. JUNGJOHANN VERLAGSGESELLSCHAFT MBH. Základní pohyby těla. In: *SZŠ Kroměříž* [online]. [cit. 2014-05-06]. Dostupné z: http://www.szskm.cz/soma/1B_pohyby.jpg

12. ING. NĚMEC, M. UML: Diagramy tříd. In: *Miloš Němec* [online]. 15. 05. 2010 [cit. 2014-05-05]. Dostupné z: http://www.milosnemec.cz/clanek.php?id=199

13. KATEDRA ANATOMIE A BIOMECHANIKY. Pažní kost. In: *Patobiomechanika a Patokinesiologie* [online]. [cit. 2014-03-04]. Dostupné z: http://biomech.ftvs.cuni.cz/pbpk/kompendium/anatomie/hk_paze_kost.php

14. KATEDRA ANATOMIE A BIOMECHANIKY. Paže a předloktí. In: *Patobiomechanika a Patokinesiologie* [online]. [cit. 2014-03-04]. Dostupné z: http://biomech.ftvs.cuni.cz/pbpk/kompendium/anatomie/hk_paze.php

15. MICROSOFT. Dědičnost (Průvodce programováním v C#). In: *Microsoft developer network* [online]. [cit. 2014-05-13]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/ms173149.aspx

16. MICROSOFT. Polymorfismus (Průvodce programováním v C#). In: *Microsoft developer network* [online]. [cit. 2014-05-13]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/ms173152.aspx

17. MICROSOFT. Abstraktní a uzavřené třídy a jejich členové (Průvodce programováním v C#). In: *Microsoft developer network* [online]. [cit. 2014-05-13]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/ms173150.aspx

18. MICROSOFT. enum (Referenční dokumentace jazyka C#). In: *Microsoft developer network* [online]. [cit. 2014-05-13]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/sbbt4032.aspx

19. DOC. ING. MIROSLAV BENEŠ, PH.D. 1.2.3. Agregace a kompozice. In: *Katedra informatiky FEI VŠB-TU Ostrava - Principy objektově orientovaného programování* [online]. [cit. 2014-05-13]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s02s03.html

20. DOC. ING. MIROSLAV BENEŠ, PH.D. 1.1.1. Objekt. In: *Katedra informatiky FEI VŠB-TU Ostrava - Principy objektově orientovaného programování* [online]. [cit. 2014-05-13]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s01s01.html

21. DOC. ING. MIROSLAV BENEŠ, PH.D. 1.1.2. Třída. In: *Katedra informatiky FEI VŠB-TU Ostrava - Principy objektově orientovaného programování* [online]. [cit. 2014-05-13]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s01s02.html

22. DOC. ING. MIROSLAV BENEŠ, PH.D. 1.3.1. Dědičnost tříd. In: *Katedra informatiky FEI VŠB-TU Ostrava - Principy objektově orientovaného programování* [online]. [cit. 2014-13-05]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s03s01.html

23. BC. FRONĚK, J. *Analýza a konstrukce protézy paže s využitím inteligentních pohonů*. Kladno: Froněk, Bc. Jan, ČVUT, Fakulta biomedicínského inženýrství, Katedra přírodovědných oborů, 2012. Bakalářská práce.

24. ČÁPKA, D. 4. díl - UML - Doménový model. *devbook.cz programátorská sociální síť* [online]. 2012 [cit. 2014-05-21]. Dostupné z: http://www.devbook.cz/uml-domenovy-model-diagram

25. ČÁPKA, D. 7. díl - Dědičnost a polymorfismus. *devbook.cz programátorská sociální síť* [online]. 2013 [cit. 2014-05-08]. Dostupné z: http://www.devbook.cz/c-sharp-tutorial-dedicnost-a-polymorfismus

26. 14. díl - Rozhraní (interface). *devbook.cz programátorská sociální síť* [online]. 2013 [cit. 2014-05-08]. Dostupné z: http://www.devbook.cz/c-sharp-tutorial-interface-rozhrani

27. DOC. ING. MIROSLAV BENEŠ, PH.D. 1.3.2. Abstraktní třídy a rozhraní. In: *Katedra informatiky FEI VŠB-TU Ostrava - Principy objektově orientovaného programování* [online]. [cit. 2014-05-13]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s03s02.html

28. KATEDRA ANATOMIE A BIOMECHANIKY. Ramenní kloub. In: *Patobiomechanika a Patokinesiologie* [online]. [cit. 2014-03-05]. Dostupné z: http://biomech.ftvs.cuni.cz/pbpk/kompendium/anatomie/hk_paze_rameno.php

Příloha

Struktura souborů na DVD

- Bakalarska_prace_Polacek.pdf
- Abstrakt_cz.pdf
- Abstract_en.pdf
- Klicova_slova.pdf
- Zadani_bakalarske_prace.pdf
- Složka RoboticArm obsahující testovací projekt RoboticArm