# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Michal** Jméno: **Jan** Osobní číslo: **435559**

Fakulta/ústav: **Fakulta informačních technologií**

Zadávající katedra/ústav: **Katedra počítačových systémů**

Studijní program: **Informatika**

Studijní obor: **Informační technologie**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Integrace databáze do NAOqi frameworku**

Název bakalářské práce anglicky:

**Database Integration for the NAOqi Framework**

Pokyny pro vypracování:

NAOqi framework zajišťuje jednotné rozhraní pro vývoj aplikací pro robota NAO, které je nezávislé na volbě programovacího jazyka a umístění v síti. Podporované jazyky jsou C++, Java, Python a grafické prostředí Choregraphe. Navrhněte a implementujte modul pro NAOqi framework v jazyce C++, který bude poskytovat přístup k databázi pro obecné potřeby vývoje aplikací pro robota. Vyberte některou z běžně užívaných databází s ohledem na omezené výpočetní zdroje robota NAO, sestavte pro ni instalační balíčky pro Gentoo linux na virtuálním stroji a následně je instalujte do hlavy robota. Při implementaci modulu zohledněte základní bezpečnostní aspekty práce s databází. Vytvořte aplikační rozhraní modulu pro jazyky C++, Python a Java, a funkční blok pro Choregraphe a otestujte je. Užití databáze v prostředí NAOqi demonstrujte na jednoduché aplikaci.

Seznam doporučené literatury:

Dodá vedoucí práce.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Marek Danel, katedra číslicového návrhu FIT**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **20.12.2016** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: _____

| Ing. Marek Danel<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | doc. RNDr. Ing. Marcel Jiřina, Ph.D.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta



© ČVUT v Praze, Design: ČVUT v Praze, VIC

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

# Integrace databáze do NAOqi frameworku

***Jan Michal***

Vedoucí práce: Ing. Marek Danel

15. května 2018

# Poděkování

Chtěl bych velice poděkovat mému vedoucímu práce, panu Ing. Marku Danelovi, za jeho cenné rady a především jeho trpělivost, při vedení mé práce.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2018 Jan Michal. Všechna práva vyhrazena.
*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### Odkaz na tuto práci

Michal, Jan. *Integrace databáze do NAOqi frameworku*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

# Abstrakt

Bakalářská práce se zabývá návrhem a implementací modulu pro robota Nao v programovacím jazyku C++. Jeho hlavním účelem bude poskytování přístupu k databázi při vývoji dalších aplikací. Jelikož tento modul bude integrován přímo v NAOqi frameworku (rozhraní pro vývoj aplikací), který má omezené výpočetní zdroje je nutné vhodně vybrat nenáročný databázový systém pro robota Nao. Jako nejvhodnější se ukázal databázový systém SQLite, který je velice kompaktní a nenáročný. Modul byl zaveden do hlavy robota a úspěšně otestován základními operacemi nad databází.

**Klíčová slova** Databázový systém, robot NAO, modul, aplikace, NAOqi Framework, C++, SQLite

# Abstract

This bachelor thesis deals with the design and implementation of the module for the Nao robot in the programming language C++. Its main purpose is providing access to the database for developing other applications. This module will be integrated into the NAOqi framework (Application Development Interface), which has limited performance, it is necessary to select a low-resource database system for the Nao robot. The most appropriate proved to be the SQLite database system, which is very compact and lightweight. The module was loaded into the robot‘s head and successfully tested by basic operations in the database.

**Keywords** Database system, NAO robot, module, application, NAOqi Framework, C++, SQLite

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Při vývoji aplikací je občas nutné ukládat důležitá data do spolehlivého úložiště dat. K tomu nám dnes pomáhají databázové systémy. Ty slouží především k ukládání dat, ale také k jejich údržbě, konzistenci a zotavování po chybách. Jelikož při práci s robotem Nao je občas také zapotřebí nějaká databáze, ať už kvůli vývoji aplikací nebo uchovávání dat, zvolil jsem si jako téma své bakalářské práce Integrace databáze do NAOqi frameworku.

NAOqi framework zajišťuje jednotné rozhraní pro vývoj aplikací pro robota NAO, které je nezávislé na volbě programovacího jazyka a umístění v síti. Podporované jazyky jsou C++, Java, Python a grafické prostředí Choregraphe. Aplikace v NAOqi frameworku může běžet s jednotným rozhraním z libovolné stanice v síti.

My se nejdříve v kapitole 1 podíváme detailněji na cíl a smysl práce. Také si zde popíšeme přínos této práce a možnosti jejího dalšího užití.

V první půlce kapitoly 2 si ukážeme robota Nao, pro kterého budeme vyrábět modul. Popíšeme jeho vývoj a robota jako takového 2.1. Poté si popíšeme jak to vypadá uvnitř robota, jak vypadá jeho operační systém, NAOqi framework, jak se vytváří moduly a jak se volají 2.1.3.

Dále si krátce vysvětlíme základní pojmy o databázových systémech 2.2 a poté porovnáme databáze SQLite, MySQL a PostgreSQL. Také si vysvětlíme, proč jsem se rozhodl zvolit pro implementaci databázový systém SQLite2.3.

Následující kapitola představí jak jsem navrhl modul, který bude komunikovat s databází 3.2. Je zde také představeno aplikační rozhraní a jeho výroba v grafickém prostředí Choregraphe 2.1.5.

V poslední kapitole 4 se podíváme na implementaci. Nejdříve si řekneme, proč není databázi potřeba instalovat. V další sekci ukážeme implementaci modulu, jeho aplikačního rozhraní a boxu v Choregraphe. Nakonec se podíváme na bezpečnost modulu, jeho otestování a jednoduchou aplikační ukázku komunikace modulu s databází.

Kapitola 1

# Cíl práce

Cílem práce je navrhnout a implementovat modul pro robota NAO, který bude komunikovat s databází uvnitř robota. Proto je třeba vhodně vybrat databázi vzhledem k omezeným výpočetním schopnostem robota a následně ji nainstalovat. Nakonec je zapotřebí vytvořit aplikační rozhraní modulu pro jazyky C++, Java, Python a grafické prostředí Choregraphe. Modul se poté otestuje pomocí jednoduché aplikace.

Hlavní smysl této práce je podpora vývoje dalších pokročilých aplikací zavedením databázového modulu přímo do hlavy robota. Mezi hlavní výhody zabalení databáze do naoqi modulu běžícího v hlavně robota patří především snadné ukládání, načítání a údržba dat. Jelikož bude k modulu vytvořeno aplikační rozhraní pro všechny programovací jazyky, kterými robot disponuje, budou mít všichni budoucí tvůrci aplikací jednoduchý a spolehlivý přístup k datovému úložišti.

- **Jednoduchý**

  Navíc bude modul integrován do NAOqi Frameworku2.1.3, tím se zredukuje závislost aplikací na externím softwaru. Uživatel nebude muset řešit žádný výběr databáze ani její instalaci. Místo toho bude rovnou postaven před rozhraní modulu, který už bude integrován v NAOqi Frameworku.

- **Spolehlivý**

  Modul běžící v robotovi bude komunikovat s databází, která bude umístěna také v robotovi. Aplikace tedy bude imunní vůči výpadkům wifi nebo připojení k internetu. Tím pádem bude vše fungovat samostatně třeba i úplně bez připojení.

Kapitola 2

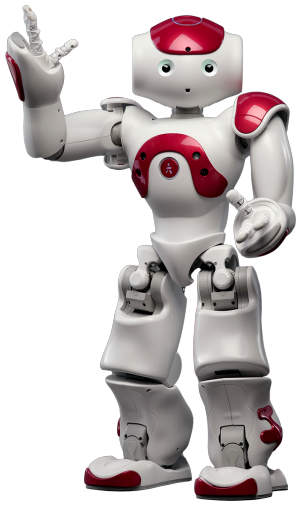
# Analýza

## 2.1 Robot NAO

Robot NAO je humanoidní robot, který je 58 centimetrů vysoký a váží 4.3 kilogramu. Pro lepší představivost je robot vidět na obrázku 2.1. Vyvinula ho francouzská společnost Aldebaran Robotics v roce 2006 a od té doby ho pořád vylepšuje. Nyní je k dispozici jeho 5. verze. Po celém světě se už prodalo více než 10 000 těchto robotů [2].

### 2.1.1 Specifikace robota

- Procesor ATOM Z530 1.6 GHz
- 1 GB RAM



Obrázek 2.1: Robot NAO [1]

- 2 GB Flash paměť
- 8 GB Micro SDHC paměť

### 2.1.2 Vývoj robota

Vývoj robota Nao začal Bruno Maisonnier v roce 2004 jako „Projekt Nao“, později v roce 2005 založil společnost Aldebaran Robotics [3]. Do této společnosti přesunul vývoj robota. Mezi lety 2005 a 2007 společnost vyrobila 6 prototypů. V březnu roku 2008 byla vypuštěna první produkční verze robota nesoucí název „Nao RoboCup Edition“, která se měla zúčastnit soutěže RoboCup téhož roku [4]. Později stejného roku byla vypuštěna další verze robota „Nao Academics Edition“ pro univerzity, vzdělávací instituty a výzkumné laboratoře.

V prosinci 2011 Aldebaran vypustila další verzi robota s názvem Nao Next Gen [5]. Ten měl hardwarové i softwarové vylepšení jako např. lepší kamery, zvýšenou robustnost, anti-kolizní systém a rychlejší chůzi. V roce 2014 vyšla poslední verze robota nesoucí název Nao Evolution s prodlouženou výdrží, vylepšeným hlasem a díky čtyřem mikrofonům lokalizaci zvuku v prostoru [3].

### 2.1.3 NAOqi Framework

NAOqi je operační systém, který využívá robot Nao. Je to GNU/Linuxová distribuce založená na Gentoo. Je vestavěnný a poskytuje robotovi vše, co potřebuje. Obsahuje spoustu programů a knihoven [6].

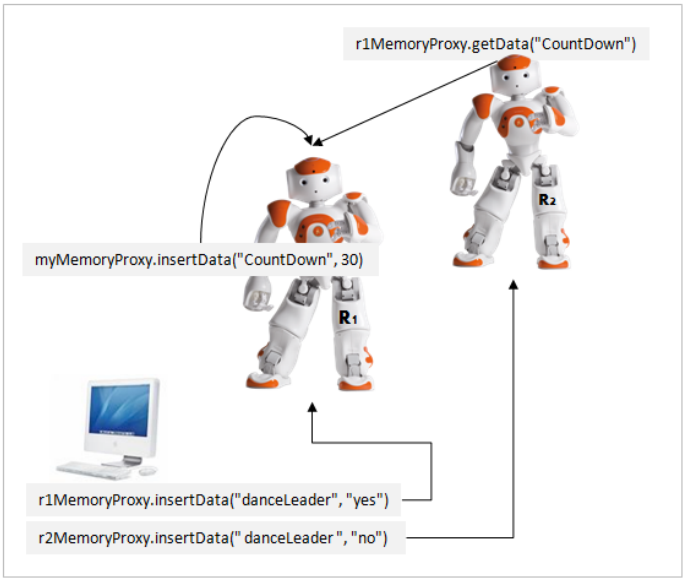
NAOqi Framework je framework využívaný k programování robota. Zodpovídá za jeho chování a to včetně: zdrojů, synchronizace a paralelního běhu programů [7].

Tento framework zajišťuje komunikaci mezi různými moduly uvnitř robota (pohyby, audio, video) a sdílení informací. Navíc je multi-platformní (Cross platform), multi-jazykový (Cross language) a dovoluje nám vytvářet distribuovanné aplikace (Distributed applications).

- **Multi-platformní**

  Může být použit na MS Windows, Linuxu nebo MacOS. Vyvíjí se následujícím způsobem:

  - Pomocí Pythonu můžeme jednodušše spustit kód buďto na našem PC nebo přímo na robotovi
  - Pomocí C++, ale kód musíme nejprve zkompilovat pro cílový operační systém. Pokud tedy chceme program spustit na robotovi, je nutné použít cross-kompilační nástroj, aby vygeneroval kód, který bude možné spustit na operačním systému robota.




Obrázek 2.2: Distribuovanné aplikace v NAOqi Frameworku [7]

- **Multi-jazykový**

  Software běžící na robotovi může být vyvíjen v Pythonu nebo C++. V obou případech jsou metody víceméně stejné. Na to, abychom přinutili robota říct „Hello World“ stačí jen:

  – V Pythonu:

  ```
  textToSpeechProxy.say("Hello World")
  ```

  – V C++:

  ```
  textToSpeechProxy->say("Hello World");
  ```

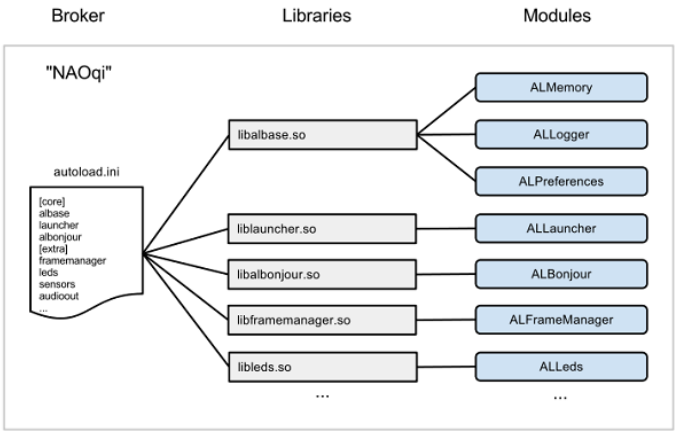
- **Distribuované aplikace**

  Aplikace běžící v reálném čase nemusí být jen spustitelné, ale mohou využívat několik procesů a/nebo modulů, které jsou rozmístěny na více robotů Nao.

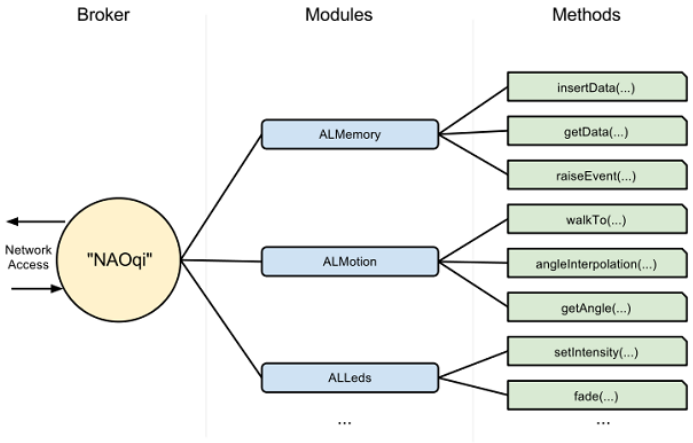
  Na obrázku 2.2 je vidět synchronizace dvou robotů Nao. Nejdříve specifikujeme robota R1 jako hlavního a R2 jako závislého na R1. Poté spustíme časovač na robotovi R1, ten se už postará, aby s robotem R2 začali ve stejný okamžik (roboti musí znát IP adresy a porty toho druhého).

### 2.1.4 NAOqi Proces

Spustitelný proces NAOqi, který běží na robotovi je broker 2.1.4.1. Při spuštění nahraje soubor autoload.ini 2.3, ten definuje, které knihovny se mají na-




Obrázek 2.3: Moduly načtené souborem autoload.ini [7]




Obrázek 2.4: Strom závislostí brokeru [7]

číst. Každá z těchto knihoven obsahuje jeden nebo i více modulů 2.1.4.2, které využívá broker k voláních jejich metod.

Broker vyhledává tak, aby každý modul ve stromu nebo přes síť mohl nalézt jakoukoli metodu, která byla načtená. Načítání modulů vytváří strom metod připojených k modulu a moduly připojené k brokeru 2.4.

#### 2.1.4.1 Broker

Je objekt, který nám poskytuje

- **Adresářové služby:** k nacházení modulů a metod
- **Síťový přístup:** dovolující spouštět metody načtených modulů zvenku procesu

#### 2.1.4.2 Modul

Typicky je každý modul třída v knihovně. Když je knihovna načtena souborem autoload.ini, je automaticky vytvořena instance třídy modulu. Konstruktor třídy, který dědí z ALModule[1] může zavázat metody. Díky tomu jsou nabízeny jejich jména a metody brokeru, takže jsou k dispozici ostatním. Modul může být lokální nebo vzdálenný.

- Pokud je vzdálený, je kompilován jako spustitelný soubor a může být spuštěn venku z robota. Vzdálené moduly jsou jednodušší na použití a mohou být debugovány jednoduše zvenčí, ale jsou méně efektivní z hlediska rychlosti a paměťové náročnosti.
- Pokud je lokální, je kompilován jako knihovna a může být spuštěn pouze na robotovi. Jsou ale více efektivní než vzdálené moduly.

Každý modul obsahuje různé metody. Mezi nimi jsou i některé vázané, neboli mohou být volány zvenku modulu, např. jiného modulu nebo spustitelného souboru. Způsob volání vázaných metod nezávisí na tom, jestli je modul lokální nebo vzdálenný, modul se automaticky přispůsobí.

- **Lokální moduly**

  Lokální moduly jsou dva (nebo i více) modulů spuštěných ve stejném procesu. Ke komunikaci mezi sebou používají pouze jeden Broker. Jelikož jsou ve stejném procesu, mohou sdílet proměnné a volat metody druhého modulu bez jakékoli serializace nebo síťování. To zaručuje nejrychlejší komunikaci mezi nimi.

- **Vzdálenné moduly**

  Vzdálenné moduly mezi sebou komunikují přes síť. Vzdálenný modul potřebuje Broker ke komunikaci s jinými moduly. Právě Broker je zde zodpovědný za všechnu síťovou činnost. Vzdálenné moduly pracují skrz SOAP[2] přes síť.

- **Připojení mezi vzdálennými moduly**

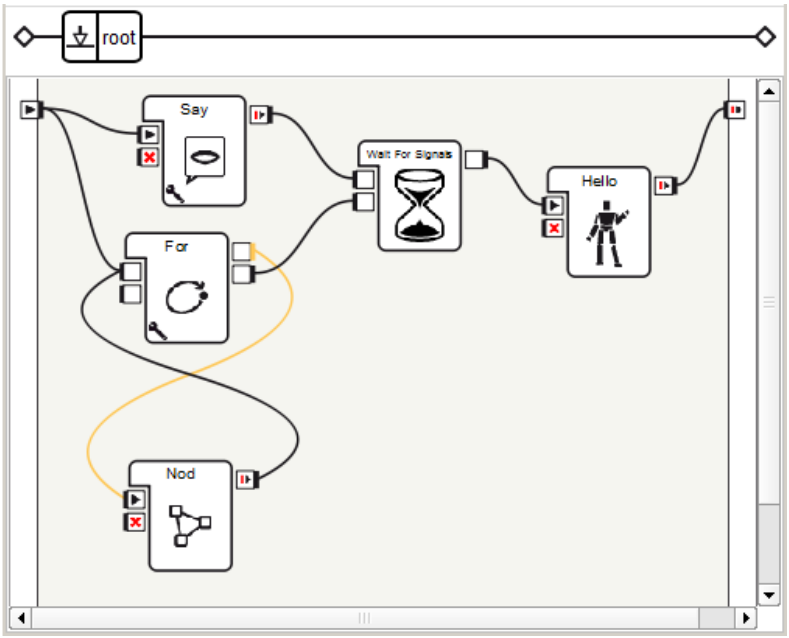
  Vzdálenné moduly mezi sebou mohou komunikovat připojením Brokeru k Brokeru jiného modulu. Připojení dvou Brokerů otevře obousměrnou komunikaci, oba moduly mohou mezi sebou komunikovat.

- **Připojení mezi Brokery**

  Je zde i možnost připojit dva moduly tím, že připojíme jejich Brokery mezi sebou. Pokud tedy máme dva moduly B a C, tak když připojíme

[1] Může být použit jako nadtřída pro potřeby uživatele, hlavně k pomoci obsluhovat a nabízet svoje metody[8]

[2] Simple Object Access Protocol




Obrázek 2.5: Pužití boxů v grafickém prostředí Choregraphe [9]

jejich Brokery, bude modul B přistupovat k metodám modulu C a obráceně, modul C bude moci přistupovat k metodám modulu B.

### 2.1.5 Choregraphe

Choregraphe [10] je multi-platformní desktopová aplikace, která nám umožňuje:

- Vytvářet animace, chování a dialogy,
- Testovat na simulovaném robotovi nebo přímo na reálném,
- Ovládat a monitorovat robota,
- Přidat vlastní chování robota pomocí kódu v Pythonu.

Choregraphe je grafické prostředí, takže je odsud možno vytvářet aplikace, jako např. pohyby robota nebo psaní e-mailů, bez toho abychom museli napsat jedinou řádku kódu. Na obrázku 2.5 je vidět ukázka programu v prostředí Choregraphe. Pokud ho spustíme, robot se představí a zároveň dvakrát pokývá hlavou, až budou obě instrukce dokončené, zamává nám na pozdrav.

V Choregraphe je přítomno velké množství těchto již předpřipravených boxů ve všemožných kategoriích např. audio, motion, flow control, apod. Ty nás ale zajímat nebudou. My potřebujeme vytvořit nový box, díky kterému bude umožněn uživateli přístup k databázi při vývoji aplikací.

#### 2.1.5.1 Box v Choregraphe

Box je základní prvek v tomto grafickém prostředí [11]. Může obsahovat velmi jednoduchou akci (např. box Say) nebo velmi komplexní aplikaci (např. průzkum celé místnosti). Každý box má následující vlastnosti: jméno, obrázek (pomáhá nám porozumět tomu, co box dělá) a popis, který se objeví pokud najedeme kurzorem na box. Boxy dále rozdělujeme podle toho co obsahuje na 4 typy:

- Python box - obsahuje skript, který je napsán v Pythonu,
- Flow diagram box - obsahuje skupinu boxů, které jsou na sebe navázané,
- Timeline box - pomáhá nám jednoduše synchronizovat boxy s pohyby, pohyby mezi sebou a/nebo boxy mezi sebou,
- Dialog box - obsahuje tzv. Dialog topic (Zde rozebírat nebudeme).

Každý box má vstup, výstup a parametry.

- Vstup - může mít jeden nebo více vstupů, vstup má vždy nějaký typ, např. String (box dostane na vstupu text) nebo Bang (boxu se prostě řekne, aby začal pracovat),
- Výstup - stejně jako u vstupu, může mít jeden nebo více výstupů, taktéž se musí vybrat jeden z několika typů výstupu,
- Parametry - boxu můžeme zadat i parametry, ty jsou nepovinné. Mohou např. upravovat chování boxu.

## 2.2 Výběr databáze

Ze všeho nejdřívě si něco popíšeme o databázích jako takových V dalších podkapitolách už budeme rozebírat konkrétní databáze. Těch bude několik málo, pouze ty, které jsem bral v potaz a hodily by se nejvíce pro implementaci, kvůli omezenému výkonu robota viz. 2.1.1.

### 2.2.1 Databáze

Databáze je soubor záznamů (zpráv), jako jsou znaky, čísla, diagramy, jejichž systematická struktura umožňuje, aby tyto zprávy mohly být vyhledávány pomocí počítače [12].

Využívá se k řízení velkého množství, perzistentních, spolehlivých a sdílených dat.

- Velkého množství - nestačí pro ně operační paměť
- Perzistentních - přetrvávají od zpracování ke zpracování

- Spolehlivých - lze je rekonstruovat po chybě
- Sdílených - jsou přístupná více uživatelům

Hlavní přínosy databázové tehchnologie:

- Nezávislost dat na aplikaci
- Efektivní přístup k datům
- Urychlení vývoje aplikací
- Integrita a ochrana dat
- Správa a zálohování dat
- Transakční zpracování
- Paralelní přístup k datům
- Zotavení po chybě

Databáze se dělí na relační, objektové a objektově-relační.

#### 2.2.1.1 Jazyk SQL

Správa databází je komplikovaný proces, tudíž se ke komunikaci s databází se využívá dotazovací jazyk SQL(Structured Query Language). Ten je zodpovědný za dotazování a úpravy informací uložených v databázi [13].

- **Historie SQL**

  SQL bylo vytvořeno v sedmdesátých letech v laboratořích IBM. V IBM vytvořili nový software jménem R. Ke správě dat uložených v systému R byl vyvinut právě jazyk SQL. Ten měl nejdříve název SEQUEL, ale později byl přejmenován na SQL, tak jak ho známe dnes.

  V roce 1979 uviděla společnost Oracle, tehdy zvaná Relational Software, komerční potenciál SQL a vydala jeho upravenou verzi zvanou Oracle V2.

- **SQL Standard**

  Standard SQL prošel lety řadou změn. Ty přidaly velké množství nových funkcí. Například podpora XML, regulární výrazy, rekurzivní dotazy a mnoho dalšího. Velké množství databázových řešení, např. MySQL nebo PostreSQL neimplementují celý standard kvůli jeho vysokému objemu. Takže i když všechny SQL implentace jsou poměrně podobné, nejsou mezi sebou kompatibilní.

- **Prvky jazyka SQL**

  - **Klauzule** - Jednotlivé komponenty dotazů a tvrzení
  - **Výrazy** - Vytváří hodnoty nebo tabulky, které se skládají z řádků a sloucpů dat
  - **Predikáty** - Specifikují podmínky, které slouží k omezení dotazů a tvrzení
  - **Dotazy** - Vrátí data na základě daných kritérií
  - **Tvrzení** - Kontrolují transakce, běh programu, připojení nebo diagnostiku. V databázových systémech se používají k odesílání dotazů od klientu do serveru, kde je databáze uložená a následné odpovědi klientovi

- **SQL Dotazy** (DQL - Data Query Language)

  Dotazy jsou nejčastější a nezbytné SQL operace. Díky dotazům, je možné vyhledávat informace v databázi. Začínají příkazem "SELECT". Pomocí těchto klauzulí mohou být více specifické:

  - **FROM**

    Tímto příkazem vybíráme tabulku, ve které dotaz provádíme
  - **WHERE**

    Říká které řádky nás zajímají. Veškeré řádky vyhodnocené jako "TRUE" budou vybrány
  - **ORDER BY**

    Určuje, jak seřadit výsledky dotazu. Jinak jsou vráceny v náhodném pořadí

- **SQL Manipulace s daty** (DML - Data Manipulation Language)

  Manipulovat s daty je nezbytná věc pro SQL tabulky. Můžeme přidat novou informaci, upravit stávající informaci nebo smazat již nepotřebnou informaci.

  - **INSERT**

    Tímto příkazem přidáme nový řádek dat do již existující tabulky
  - **UPDATE**

    Příkazem jednodušše upravíme stávající záznamy v tabulce novými hodnotami
  - **DELETE**

    Díky tomuto příkazu jsme schopni smazat nepotřebné řádky z tabulky na trvalo

- **SQL Definice dat** (DDL - Data Definition Language)

  Dovoluje uživateli definovat nové tabulky a prvky

  - **CREATE**

    Tímto příkazem přidáme novou tabulku do již existující databáze

  - **DROP**

    Díky tomuto příkazu jsme schopni smazat nepotřebné tabulky z databáze na trvalo

  - **TRUNCATE**

    Příkazem snadno smažeme veškeré stávající záznamy v tabulce, ale tabulku jako takovou necháme být(nesmažeme ji a bude prázdná)

  - **ALTER**

    Tento příkaz dovoluje uživateli upravovat již existující objekt, např. přidání řádku do tabulky

- **SQL Kontrola dat** (DCL - Data Control Language)

  Dovoluje uživateli definovat privilegia jiným uživatelům do různých databází

  - **GRANT**

    Tímto příkazem přidáme uživateli práva upravovat vybranou tabulku

  - **REVOKE**

    Díky tomuto příkazu jsme schopni uživateli odebrat veškerá, dříve přidaná práva

#### 2.2.1.2 ACID Vlastnosti

Jsou 4 vlastnosti, které garantují validitu databáze za všech okolností, včetně všech možných chyb, které mohou nastat.

- **Atomic** - Atomické

  Transakce musí proběhnou buď celá nebo vůbec

- **Consistent** - Konzistentní

  Databáze se mění z konzistentního stavu do jiného konzistentního stavu

- **Isolated** - Nezávislé

  Jednotlivé efekty transakce nejsou viditelné jiným transakcím

- **Durable** - Trvalé

  Úspěšně provedené transakce jsou trvale uloženy


Obrázek 2.6: SQLite Logo [14]

### 2.2.2 SQLite

SQLite je jednoduchá knihovna, která implementuje soběstačnou, bez serverovou, nepotřebující konfiguraci a transakční SQL databázi [14].

- **Soběstačnou**

  Knihovna je soběstačná, v tom smyslu, že jí k běhu stačí velmi málo závislostí. Běží na jakémkoli operačním systému, dokonce i na velmi ořezaných, vestavných systémech. Nejsou zapotřebí žádné externí knihovny nebo jiné rozhraní. Stačí pouze několik základních C-éčkových knihoven.

- **Bez serverovou**

  Nefunguje na architektuře klient-server

- **Nepotřebující konfiguraci**

  Nepotřebuje být "nainstalovaná"před použitím ani nijak nastavená

- **Transakční**

  Všechny změny a dotazy splňují ACID vlastnosti

SQLite je možno využít ke všem účelům, soukromým nebo komerčním. Zapisuje a čte přímo do běžných souborů. Jedna kompletní databáze, včetně všech tabulek, indexů a pohledů=jeden soubor. Soubor není závislý na platformě, lze ho tedy různě přesouvat a používat. Dokonce i mezi různě bitovými operačními systémy(32b a 64b) nebo architekturami little-endian a big-endian. Projekt SQLite byl spuštěn v roce 2000.

Pro více informací je možno navštívit oficiální stránky projektu: `https://www.sqlite.org` nebo přímo SQLite dokumentaci

### 2.2.3 MySQL

MySQL je nejpopulárnější Open Source databázový systém, který je vyvíjen, distribuován a podporován v Oracle Corporation [16].

- **Hlavní znaky**


Obrázek 2.7: MySQL Logo [15]

- Napsáno v C/C++
- Testováno dlouhou řadou různých kompilátorů
- Funguje na mnoha různých platformách
- Používá více-vrstvý server, který využívá nezávislé moduly
- Je navržen tak, aby byl více-vláknový pomocí užívání kernelových vláken, díky tomu jednodušše využívá více jader procesoru, pokud jsou k dispozici
- Užívá velmi rychlé B-stromové diskové tabulky (MyISAM) s kompresí indexů
- Je zde implementováno mezipaměťové ukládání hashovacích tabulek, které se využívají jako dočasné tabulky

- **Relační databáze**

  MySQL je relační databáze, to znamená, že ukládá data do různých tabulek, místo ukládání všech dat do jedné velké kolekce. Struktura databáze je rozdělena do několika fyzických souborů, kvůli vyšší rychlosti. Logický model obsahující různé objekty, např. jednotlivé databáze, tabulky, pohledy, řádky a slouce, nabízí flexibilní programovatelné prostředí. Uživatel zde nastavuje vztahy vedoucí mezi různými datovými objekty, např. one-to-one(jedna k jedné), one-to-many(jeden k mnoha), unique(jedninečný), required(nutný), optional(volitelný) a pointers(ukazatele) mezi jednotlivými tabulkami.

- **Open Source**

  MySQL je Open Source, což znamená, že je možné volně používat MySQL a dokonce upravovat. Kdokoliv si může tento software stáhnout a používat zcela zdarma. Pokročilejší uživatelé mohou dokonce změnit zdrojový kód, dle svého uvážení, tak aby jim více vyhovoval. Používá se zde GPL (GNU General Public License), která určuje, co se smí a nesmí dělat

s tímto softwarem v různých situacích. Pro ty, komu nevyhovuje GPL licence, je možno zakoupit komerční licenci přímo od Oracle Corporation.

- **Klient/Server Architektura**

  MySQL funguje na architektuře klient/server. Neboli systém se skládá z více-vláknového SQL serveru, který může obsluhovat zároveň několik různých klientů, knihoven a celou řadu aplikačních programovacích rozhraní(API). Avšak je zde i možnost mít MySQL Server jako vestavěnnou více-vláknovou knihovnu, kterou pouze nalinkujeme do naší aplikace. Touto možností získáme malý, jednoduchý a snadno konfigurovatelný produkt.

- **Velmi rychlý, spolehlivý, škálovatelný a snadno použitelný**

  MySQL je možno rozeběhnout na jakémkoli počítači nebo netobooku, vedle další aplikací, třeba webových serverů. MySQL je možno škálovat do více počítačů(clusteru), připojených mezi sebou. Cílem vývoje bylo původně ovládání velkých databází mnohem rychleji, než všechny dosud existující řešení.

- **Datové typy**

  Je zde mnoho datových typů: znaménkové/neznaménkové celočíselné hodnoty (integery) 1,2,3,4 a 8 bytů dlouhé, dále např. float, double, char, varchar, binary, varbinary, text, blob, date, time, datetime, timestamp, year, set nebo enum. Pro více informací o datových typech navštivte MySQL dokumentaci o datových typech.

- **Bezpečnost**

  Systém hesel a privilegií je velmi flexibilní a bezpečný a tak povoluje ověření přímo u hostitele. Hesla jsou zabezpečena šifrováním a včetně veškeré heslové komunikace mezi serverem a klientem.

- **Konektivita**

  Klienti se mohou k MySQL Serveru přihlásit skrze několik protokolů:

  – Skrze TCP/IP sokety na jakékoli platformě
  – Na Unixových systémech pomocí Unixových doménových soketů

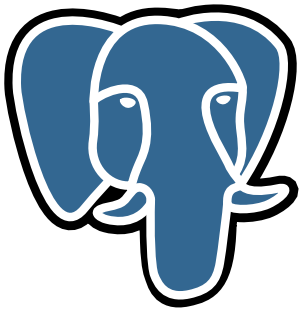
  Programy MySQL klientů mohou být napsány v jednom z mnoha jazyků. Jsou k dispozici API pro C, C++, Eiffel, Javu, Perl, PHP, Python, Ruby nebo Tcl.

- **Vestavná MySQL Serverová Knihovna**

  Dle [17] existuje vestavěnná knihovna MySQL. Dokáže běžet přímo v aplikaci u klienta. Hlavní výhodou je především větší rychlost a zjednodušenné nastavení pro vestavnné aplikace. Je založena stejné architektuře

| Funkce | Popis |
|---|---|
| mysql_ library_ init() | Inicializuje databázi, je jí tedy vhodné volat co nejdříve po spuštění programu, musí být volaná jako první ze všech MySQL funkcí |
| mysql_ library_ end() | olá se před koncem programu, slouží k ukončení běhu databáze |
| mysql_ thread_ init() | Je potřeba zavolat v každém vlákně, které vytvoříme a chceme v něm přistupovat k MySQL |
| mysql_ thread_ end() | Volá se před ukončením vlákna, které využívalo MySQL |

Tabulka 2.1: Hlavní funkce vestavěnné MySQL knihovny



Obrázek 2.8: PostgreSQL Logo [18]

jako MySQL opět napsána v C/C++. API je identické pro obě verze, jak pro vestavěnnou MySQL knihovnu a MySQL s architekturou klient-server. V tabulce 2.1 jsou vypsány její hlavní funkce.

- **Zajímavosti**

  MySQL je pojmenováno po dceři jednoho ze spoluzakladatelů Montyho Wideniuse, která se jmenuje My. „Sakila“ je jméno delfína, který je vidět na logu MySQL, obrázek 2.7. Jméno bylo vybráno jako jedno z mnoha v soutěži „Name the Dolphin“, do které zasílali uživatelé své návrhy.

### 2.2.4 PostreSQL

PostgreSQL je výkonný, Open Sourcový, objektově-relační, databázový systém, který využívá a rozšiřuje standard jazyka SQL. Přidává do SQL mnoho nových funkcí, díky kterým bezpečně ukládá a škáluje i ty nejkomplikovanější data. Počátky projektu sahají až do roku 1986 jako součást projektu POSTGRES [19].

- **Hlavní znaky**

  PostgreSQL si získalo svou silnou reputaci především za svojí prověřenou architekturu, spolehlivost, datovou integritu a rozšiřitelnost. Lze jej nainstalovat na všechny majoritní operační systémy a od roku 2001 splňuje vlastnosti ACID.

- **Rozšiřitelnost**

  PostgreSQL přichází s mnoha nový funkcemi, ty mají pomoci vývojářům vyvíjet aplikace, administrátorům zabezpečit data a vytvořit prostředí tolerující chyby. Navíc je PostgreSQL velmi rozšiřitelné. Zkušenější uživatel může definovat nové vlastní datové typy, vytvořit nové funkce nebo dokonce psát kód z různých programovacích jazyků, bez nutnosti překompilování databáze.

- **Podpora SQL**

  PostgreSQL se snaží co nejvíce přiblížit Standardu SQL, ale v některých případech by to vedlo ke špatným návrhům v architektuře nebo k rozporu s tradičními funkcemi. Mnoho funkcí z SQL Stadardu je tedy podporováno, bohužel občas s lehce pozměněnou syntaxí nebo funkčností.

- **Datové typy**

  - Základní typy - Integer(celočíselná hodnota), Numeric(číslo), String(řetězec), Boolean(pravda/nepravda), Date/Time(datum nebo čas), Array(pole), Range(rozsah)
  - Dokumentové typy - JSON/JSONB, XML, Key-value
  - Geometrické typy - Point(bod), Line(přímka), Circle(kružnice), Polygon
  - Vlastní datové typy - Uživatel si sám může definovat nové datové typy, pokud žádný nevyhovuje jeho účelům

- **Historie PostgreSQL** [20]

  - **Projekt POSTGRES**

    Tento projekt vzniknul v roce 1986 na Berkeley University v Kalifornii. Byl veden profesorem Michealem Stonebrakerem a sponzorován společnostmi DARPA[3], ARO[4], NSF[5] a ESL Inc[6]. První demoverze byla spustitelná až rok poté a představena v roce 1988.

[3]Defense Advanced Research Projects Agency
[4]Army Research Office
[5]National Science Foundation
[6]Electromagnetic Systems Laboratory

| Databáze | SQLite | MySQL | PostgreSQL |
|---|---|---|---|
| Vyžaduje instalaci | Ne | Ano | Ano |
| Architektura klient-server | Ne | Ano i Ne | Ano |
| Stačí nalinkovat do programu | Ano | Ne | Ne |

Tabulka 2.2: Porovnání databází

Projekt byl implementován k různému vyhledávání a produkci aplikací. Především pro analýzu finančních dat, monitorování výkonu tryskových motorů, stopování asteroidů, lékařský informační systém a geografické systémy. Byl také využit jako výukový nástroj na několika univerzitách.

– **Postgres95**

V roce 1994 byl přidán SQL jazyk do POSTGRESu a tím vznikl Postgres95, který byl na webu publikován jako Open Source. Ten byl napsán v ANSI C, čímž se zmenšila jeho velikost. Mnoho interních změn mělo za následek vylepšení výkonu a udržitelnosti. Nakonec byl dotazovací jazyk PostQUEL kompletně nahrazen jazykem SQL.

– **PostgreSQL**

V roce 1996 nakonec bylo jasné, že opět bude nutná změna jména, vybralo se tedy jméno PostgreSQL. Tím se zachoval původní název POSTGRES a jeho kompatibilita s jazykem SQL.

- **Logo projektu**

Logo, které je vidět na 2.8, obsahuje slona, který se jmenuje „Slonik“. Dle [21] je jeho původ tohoto jména veřejnosti neznámý.

## 2.3 Shrnutí

Nejdříve jsme si v sekci 2.1 podívali blíže na robota Nao a jeho operační systém NAOqi, poté v sekci 2.2, jsme si podrobně rozebrali všechny databáze, které jsem bral v potaz. V tabulce 2.2 je vidět porovnání všech databází uvedených v této práci.

SQLite je poměrně malá databáze, které běží na víceméně všech operačních systémech. K jejímu běhu není zapotřebí velkých zdrojů a tím pádem se velmi hodí pro vestavné systémy.

MySQL je nejvíce populární open sourcová databáze. Díky tomu se na internetu nachází velké množství uživatelů, které s ní řešili nejrůznější problémy. Tudíž se skoro každý problém nechá rychle najít a opravit. Bohužel je už poměrně velká a komplikovaná, tudíž se pro naše účely moc nehodí.

PostgreSQL je obdobně jako MySQL poměrně dost velká databáze. Funguje na architektuře klient-server, což se pro naše účely nehodí a je to spíše na škodu.

Nakonec jsem se rozhodl zvolit SQLite. Především kvůli jejímu velmi snadnému užití. Není potřeba ji nějak instalovat, stačí pouze z oficiálních stránek projektu stáhnout zdrojové kódy a ty přidat do programu jako knihovnu.

Kapitola 3

# Návrh

## 3.1 Databáze

Jak již bylo zmíněno kapitole 2, v sekci 2.3, rozhodl jsem se vybrat jako nejvhodnější databázi právě SQLite. SQLite se nemusí instalovat, stačí pouze její zdrojové kódy nalinkovat do programu. To nám velmi usnadní práci a testování.

## 3.2 Návrh modulu

Modul pro NAOqi Framework bude napsán v programovacím jazyku C++. Aplikačním rozhraním modulu se zde rozumí hlavičkový soubor[7] modulu.

Modul je vlastně třída v C++. Naše třída(modul) se jmenuje „sqlstatement“ a je rozdělena do dvou souborů:

- První soubor je hlavičkový, s názvem „sqlstatement.h“. Ten obsahuje deklarace proměnných a metod. Také jsou v něm rozepsány komentáře o jednotlivých metodách, jejich argumentech a návratových hodnotách.
- Druhý soubor je zdrojový. Jmenuje se „sqlstatement.cpp“. Bude obsahovat implementaci metod, které jsou deklarované v hlavičkovém souboru a několik dalších funkcí.

Nyní si blíže představíme metody, kterými modul disponuje:

- setDatabaseName(dbName)

  Funkce nastaví modulu jméno databáze, kterou má používat. Očekává jeden parametr a to právě jméno databáze.

[7] Hlavičkový soubor má příponou .h. Jsou v něm deklarovány proměnné, funkce a metody. Ty jsou rozumně okomentovány tak, aby programátorovi při používání modulu rozumně napovídal a tedy nemusel neustále nahlížet do dokumentace, aby zjistil, která funkce co dělá a jaké má parametry.

- setSQLStatement(statement)

  Funkce nastaví SQL příkaz. V tomto případě není nijak kontrolován a uživatel si může s databází dělat, co uzná za vhodné. Má pouze jeden parametr, ve kterém se zadává SQL příkaz.

- executeSQLStatement()

  Funkce vykoná nastavený SQL příkaz. Nemá žádné parametry. Má návratovou hodnotu a to typ bool, ten vrací, zdali se příkaz vykonal či nikoli.

- setTable(table)

  Funkce nastaví tabulku, ve které bude posléze možné provádět dotazy. Očekává jméno tabuklu zadané prvním parametrem.

- selectAll()

  Funkce zobrazí všechna data z předem nastevené tabulky. Vrací zdali se dotaz povedl.

- selectTop(top, order)

  Funkce zobrazí prvních x řádků dat, zadaných prvním parametrem, defaultně 5. Pomocí druhého parametru je možné setřídit dle sloupce. Vrací zdali se dotaz povedl.

- selectColumns(columns)

  Funkce zobrazí pouze sloupce tabulky zadané parametrem, defaultně všechny. Vrací zdali se dotaz povedl.

- selectTopColumns(columns, top, order)

  Funkce zobrazí zadané sloupce tabulky a znich pouze zadaných x řádků. Defaultně zobrazí všechny sloupce a 5 řádků. Třetím parametrem je možné uspořádat zobrazenné řádky. Vrací zdali se dotaz povedl.

## 3.3 Aplikační rozhraní

Dále je zapotřebí vytvořit aplikační rozhraní modulu. Pro každý z jazyků C++, Java a Python bude stačit vytvořit příslušný hlavičkový soubor, ve kterých budou vhodně okomentované všechny metody, návratové hodnoty a parametry. Pro grafické prostředí Choregraphe2.1.5 je zapotřebí navrhnout a implementovat tzv. box2.1.5.1.

### 3.3.1 Návrh boxu

Boxů budeme vytvářet několik, tak abychom co nejvíce uspokojili uživatele a nabídli mu co největší pohodlí při vytváření aplikací. Tedy ty nejvíce používané dotazy budou předem připravenné. Dále bude vytvořen jeden obecný box, díky tomu bude možné provést v databázi jakýkoli přikaz.

- Základní box se nazývá executeSQLStatement. Má dva vstupy. První vstup očekává String, díky tomu jsme schopni z výstupu jiného boxu zadat rovnou dotaz do databáze. Druhý vstup bude typu Bang. To kvůli tomu, abychom mohli zadat SQL přikaz parametrem. Jak již vyplývá z předchozí věty je zde jeden parametr, který očekává SQL příkaz. Ani u jednoho SQL příkazu nebude žádná kontrola, tudíž co uživatel zadá, to se s databází také stane. Box má tři výstupy. První dva jsou typu Bang. Jeden je použit, pokud se příkaz povedl, druhý pokud příkaz selhal. Třetí výstup vrací data typem String.
- Další box se jmenuje selectFromDB. Opět, jako v předchozím případě má několik vstupů. První typu String, kde očekává jméno databáze, ze které má zobrazovat data. Druhý vstup je typu Bang, který se musí využít současně s parametrem, ve kterém musíme boxu zadat jméno databáze. Dále obsahuje několik nepovinných parametrů, ve kterých můžeme nastavit, jaké sloupce zobrazit, kolik řádků zobrazit a možnost seřazení zobrazených řádků. Narozdíl od předchozího boxu, zde jsou dva výstupy. Oba typu String. První z nich vrací v případě úspěšného dotazu požadovaná data. Pokud dotaz selže, využije se druhý výstup, který vrací proč dotaz selhal.

Kapitola 4

# Realizace

## 4.1 Databáze

Jelikož databázi není třeba instalovat, stačí nám pouze z `https://www.sqlite.org/download.html` stáhnout archiv zdrojovových kódů SQLite. Ty jsou pod položkou Source Code. Poté je rozbalíme v adresáři, kde budeme program psát. Do programu stačí pouze přidat hlavní hlavičkový soubor. Pokud jsme tedy stažený archiv rozbalili do složky „sqlite“, stačí nám v C++ zadat příkaz:

```
#include "sqlite/sqlite3.h";
```

Tím se načte celá knihovna do programu a můžeme využívat její proměnné, funkce a metody.

### 4.1.1 Ukázka instalace

Výše již bylo zmíněno, že databázový systém SQLite není třeba instalovat. Jelikož byl jedním z cílů práce nainstalovat databázi, zkusil jsem nainstalovat alespoň MySQL klienta do robota Nao.

K tomu je zapotřebí NAOqi OS ve virtuálním stroji. Do kterého se přihlásíme a zadáme následující příkazy:

```
su
emerge mysql
```

- su - Přepne nás na uživatele root (vyžaduje heslo)
- emerge mysql - stáhne potřebné balíčky a uloží je do složky /home/nao/opennao-distro/packages

Poté spustím následující příkaz z našeho PC, tím se stažený obsah přesune do aktuálního adresáře. Parametr „-r“ kopíruje rekurzivně všechny podadresáře a parametr „-P 2222“ určuje port k virtuálnímu PC.

```
scp -r -P 2222 nao@localhost:/home/nao/opennao-distro/packages .
```

Když se podíváme na obsah uvidíme následující strukturu:

```
packages
├── dev-db
│   └── mysql-5.1.56.tbz2
├── Packages
└── virtual
    └── mysql-5.1.tbz2
```

Nyní nám už jen stačí obsah zkopírovat do robota a následně rozbalit do kořenového adresáře.

```
cd /
scp -r user@IPofLocalPC:/path/to/packages/dev-db/mysql-5.1.56.tbz2 .
tar -jxvf mysql-5.1.56.tbz2
scp -r user@IPofLocalPC:/path/to/packages/virtual/mysql-5.1.tbz2 .
tar -jxvf mysql-5.1.tbz2
```

Tímto postupen jsme úspěšně nainstalovali MySQL klienta.

## 4.2 Modul

### 4.2.1 Vytvoření modulu

Na to abychom vytvořili C++ modul, nám dle [22] stačí zadat příkaz:

```
qisrc create mymodule
```

Tím se nám vytvoří v aktuálním adresáři nový projekt, s názvem „mymodule“. Když se podíváme do nově vytvořené složky, bude mít následující strukturu:

```
mymodule
├── CMakeLists.txt
├── main.cpp
├── qibuild.cmake
└── qibuild.manifest
```

- mymodule: Název modulu, projektu i složky kterou jsme vytvořili
- CMakeLists.txt: Skript, který čte CMake, k vygenerování makefilů
- main.cpp: Zatím jen standartní „Hello World“
- qibuild.cmake: Musí zde být, CMakeLists.txt ho využívá k hledání qiBuild CMake frameworků
- qibuild.manifest: Také zde musí být, je nutný k vytvoření projektu pro qiBuild

### 4.2.2 Upravení modulu

Nyní budeme pokračovat podle návrhu z kapitoly 3 ze sekce 3.2. Nejdříve vytvoříme třídu, kterou nazveme sqlstatement. Naše nově vytvořená třída bude dědit z nadtřídy ALModule. Do hlavičkového souboru sqlstatement.h umístíme deklaraci našich proměnných, metod a příslušně je okomentujeme. Do zdrojového souboru sqlstatement.cpp implementujeme všechny metody. Dále je potřeba upravit soubory main.cpp a CMakeLists.txt. Náš modul bude lokální, jelikož chceme, aby byl co nejvíce spolehlivý. Pokud nám například vypadne wifi nebo bude nějaký jiný problém na síti nebude to na náš modul mít vůbec žádný vliv. Tím pádem musíme soubor CMakeLists.txt upravit tak, aby nám vygeneroval knihovnu:

```
# Příkaz qi_create_bin nahradíme příkazem
qi_create_lib
# Tím se nám místo binárního spustitelného souboru vygeneruje knihovna.
```

Dále musíme upravit soubor main.cpp. V tomto souboru už nepotřebujeme hlavní funkci main, ale pouze vstupní a výstupní body, které bude využívat naše knihovna. Přidáme tedy tyto dvě funkce:

```
int _createModule(boost::shared_ptr<AL::ALBroker> broker);
int _closeModule();
```

Takto vytvořený lokální modul musíme nahrát na robota a přidat cestu do souboru autoload.ini. Při každém spuštění se poté náš modul načte a budeme ho moci používat.

### 4.2.3 Vytvoření boxu v Choregraphe

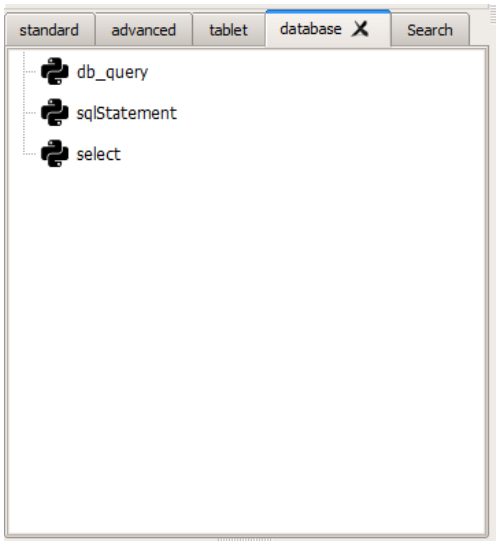
Abychom vytvořili box nám stačí kliknou na „Create a new box“ a vybrat si druh. Poté zadat jméno, popis a zvolit si obrázek boxu. Dále si zvolíme jaké bude mít vstupy, výstupy a parametry. Tím je náš box vytvořen.

My budeme vytvářet dva Python boxy. Jeden z nich bude sloužit k zobrazování dat (select) a druhý na ostatní SQL příkazy, které nevracejí žádná data, ale pouze mění stávající. Jelikož jsou boxy programovány v jazyku Python, nebudeme muset ani zde řešit instalaci SQLite. Jelikož od verze 2.5 byl do Pythonu přidán SQLite modul [23]. Takže nám stačí modul pouze importovat příkazem:

```
import sqlite3
```

- Box **sqlStatement**

  Má dva vstupy, první je typu Bang a využívá se současně s parametrem sqlParam. Druhý vstup očekává String. SQL příkaz tedy můžeme zadat na vstupu (pokud např. záleží na předchozí operaci) nebo ho zadat




Obrázek 4.1: Knihovna v Choregraphe

parametrem (pokud ho víme při překládání programu). Dále je k dispozici parametr „databaseName“. Ten má defaultní hodnotu „test.db“, pokud ale chceme použít jinou databázi stačí ho jednoduše přepsat. Má výstup typu Bang, který se zavolá, když se provede SQL příkaz a ukončí komunikace s databází.

- Box **select**

  Opět jako v předchozím případě i tento box má dva vstupy. První typu Bang, který se využívá současně s parametrem sqlStatement a druhý typu String. Ten se využívá pokud box vrací data v závislosti na předchozích rozhodnutí (SQL příkaz mu dá výstup předchozího boxu). Dále má parametry databaseName (pro výběr databáze), tableName (pro výběr tabulky) a columns (pokud chceme zobrazit jen určité sloupce). Výstup má jen jeden a to typu String, tím vrací zobrazená data.

Nyní vytvoříme novou knihovnu, tu nazveme „database“ a do ní umístíme tyto dva boxy. Knihovna bude připravena na import do grafického prostředí Choregraphe. Bude umístěna na přiloženém CD. Jak takto vytvořená knihovna vypadá je vidět na obrázku 4.1.

### 4.2.4 Bezpečnost modulu

Jelikož jeden z cílů práce je zohlednit bezpečnost databáze při implementaci, jsou v modulu implementovány bezpečností aspekty. Všechny metody modulu kontrolují hodnoty vložené parametrem, tak aby nebyly v hodnotách nechtěné příkazy. Každý z parametrů je jednotlivě otestován a nesmí obsahovat nic ji-

ného než alfanumerické znaky anglické abecedy. Parametr „columns“, kterým určuji jaké sloupce se mají zobrazit může navíc obsahovat čárku a mezeru.

### 4.2.5 Testování

Pro otestování modulu jsem napsal kód v C++, ten se nachází v souboru testSQLStatement.cpp (je k nalezení na přiloženém CD). Tento program se zkompiluje pomocí CMakeLists.txt. Tím se vytvoří spustitelný binární soubor, kterému parametrem zadáme IP adresu robota Nao a on nám modul otestuje. Pokud vše dopadne dobře vypíše na posledním řádku hlášku „All tests done well“. Program zkouší všechny možné neplatné vstupy a očeká vyhození vyjímek. Dále zkouší všechny platné vstupy a očekává úspěšné návratové hodnoty.

### 4.2.6 Aplikace

Poslední cíl práce byl vyzkoušet užití databáze na jednoduché aplikaci. K tomuto úkolu jsem naprogramoval zjednodušenou verzi hry 20 otázek. Počítač se nás zeptá na dvě otázky a na základě našich odpovědí zadá dotaz do databáze zvířat. Program je k vidění na přiloženém CD. Hra je založena na matici příznaků, kde každé zvíře v databázi obsahuje zdali je pro něj hodnota true (1) nebo false (0). Zde je výstup tohoto programu:

```
Think of an animal
Is it a mammal?
Enter [y/n]: n
Does it live in the water?
Enter [y/n]: y
You think of: Shark
```

# Závěr

V práci je nejdříve představen robot Nao. Pro něj jsem navrhnul a implementoval modul, který poskytuje uživatelům přístup k databázi. Jelikož robot žádnou databází nedisponuje, bylo zapotřebí nejdříve nějakou vhodnout vybrat, nakonec jsem se rozhodl použít databázový systém SQLite. Navrhnul jsem modul, jeho funkce a jeho aplikační rozhraní. Podle návrhu jsem jej implementoval, otestoval a předvedl na jednoduché aplikaci.

Cílem práce bylo vybrat databázový systém vzhledem k omezeným výpočetním schopnostem robota Nao. Jako nejvhodnější databázový systém se ukázal SQLite, díky své jednoduchosti, nenáročnosti a nenutnosti instalace. Díky jeho kompaktním rozměrům je jeho kompletní zdrojový kód (který je napsán v programovacím jazyku C++, stejně jako náš vyvíjený modul) přímo nalinkován v implementovaném modulu. Protože byl jeden z cílů práce sestavit instalační balíčky a nainstalovat je do hlavy robota, ukázali jsme si jak nainstalovat alespoň MySQL databázového klienta do robota.

Při implementaci modulu a všech jeho funkcí bylo přihlíženo i na bezpečnost. Takže není možné, aby nám někdo podstrčil např. SQL injekci. Dále bylo zapotřebí vytvořit aplikační rozhraní modulu a funkční blok pro Choregraphe, o tom pojednává kapitola 4. Jsou k dispozici na přiloženém CD. Nakonec jsme ukázali použití modulu na jednoduché verzi hry 20 otázek.

Pro navazující práci by bylo možné do modulu implementovat více specializovaných funkcí např. pro vložení tabulky do databáze, vložení dat do tabulky apod.

# Literatura


[1] NAO Evolution Educator Pack. [online], © 2016, [cit. 2.5.2018]. Dostupné z: https://www.robotlab.com/store/nao-evolution-educator-pack

[2] Discover Nao, the little humanoid robot from SoftBank Robotics. [online], [cit. 26.4.2018]. Dostupné z: https://www.ald.softbankrobotics.com/en/robots/nao

[3] NAO Evolution. [online], [cit. 26.4.2018]. Dostupné z: https://www.ald.softbankrobotics.com/en/press/press-releases/unveiling-of-nao-evolution-a-stronger-robot-and-a-more-comprehensive-operating

[4] RoboCup Standard Platform League. [online], [cit. 26.4.2018]. Dostupné z: http://spl.robocup.org/

[5] Melanson, D.: Aldebaran Robotics announces Nao Next Gen humanoid robot. [online], 2011, [cit. 26.4.2018]. Dostupné z: https://www.engadget.com/2011/12/10/aldebaran-robotics-announces-nao-next-gen-humanoid-robot-video/

[6] NAOqi OS - Getting started. [online], [cit. 26.4.2018]. Dostupné z: http://doc.aldebaran.com/2-1/dev/tools/opennao.html

[7] key concepts - Aldebaran 2.1.4.13 documentation. [online], [cit. 26.4.2018]. Dostupné z: http://doc.aldebaran.com/2-1/dev/naoqi/index.html

[8] AL:ALModule Class Reference. [online], [cit. 2.5.2018]. Dostupné z: http://doc.aldebaran.com/2-1/ref/libalcommon/a00008.html

[9] Using Flow Control Boxes. [online], [cit. 8.5.2018]. Dostupné z: http://doc.aldebaran.com/2-1/software/choregraphe/tutos/flow_control.html

[10] What is Choregraphe. [online], [cit. 8.5.2018]. Dostupné z: http://doc.aldebaran.com/2-1/software/choregraphe/choregraphe_overview.html

[11] Using Flow Control Boxes. [online], [cit. 8.5.2018]. Dostupné z: http://doc.aldebaran.com/2-1/software/choregraphe/objects/box.html

[12] Valenta, M.: Přednáška č. 1. [online], 2018, [cit. 22.4.2018]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-DBS/_media/lectures/b172/hand-dbs_lec_1_cz_uvod.pdf

[13] What is SQL, how does it work and how is it being used. [online], © 2002-2018, [cit. 22.4.2018]. Dostupné z: https://www.ntchosting.com/encyclopedia/databases/structured-query-language/

[14] About SQLite. [online], [cit. 22.4.2018]. Dostupné z: https://www.sqlite.org/about.html

[15] MySQL. [online], © 2018, [cit. 2.5.2018]. Dostupné z: https://www.mysql.com/

[16] MySQL 5.7 Reference Manual. [online], © 2018, [cit. 23.4.2018]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

[17] libmysqld, the Embedded MySQL Server Library. [online], © 2018, [cit. 5.5.2018]. Dostupné z: https://dev.mysql.com/doc/refman/5.5/en/libmysqld.html

[18] Logo - PostgreSQL wiki. [online], Květen 2014, [cit. 2.5.2018]. Dostupné z: https://wiki.postgresql.org/wiki/Logo

[19] PostgreSQL: About. [online], © 1996-2018, [cit. 24.4.2018]. Dostupné z: https://www.postgresql.org/about/

[20] A Brief History of PostgreSQL. [online], ©1996-2018, [cit. 26.4.2018]. Dostupné z: https://www.postgresql.org/docs/10/static/history.html

[21] Dybka, P.: The History of Slonik, the PostgreSQL Logo. [online], 2016, [cit. 24.4.2018]. Dostupné z: http://www.vertabelo.com/blog/notes-from-the-lab/the-history-of-slonik-the-postgresql-elephant-logo

[22] Extending NAO API - Creating a new module. [online], [cit. 6.5.2018]. Dostupné z: http://doc.aldebaran.com/2-1/dev/cpp/tutos/create_module.html

[23] Python Documentation. [online], © 1990-2018, [cit. 10.5.2018]. Dostupné z: https://docs.python.org/2/library/sqlite3.html

Příloha **A**

# Obsah přiloženého CD

- `text` ..... text práce
  - `BP_Michal_Jan.pdf` ..... text práce ve formátu PDF
  - `BP_Michal_Jan.tex` ..... kód textu ve formátu LaTeX
- `src` ..... adresář
  - `module` ..... zdrojové kódy implementace modulu
  - `packages` ..... instalační balíky do robota
  - `database.cbl` ..... knihovna do Choregraphe