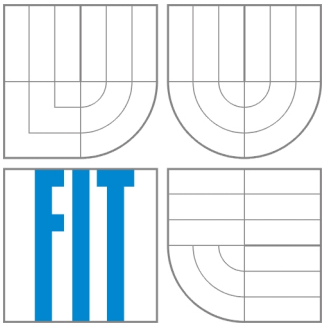


Simluátor Trilobota

(projekt do předmětu ROB)

Kamil Dudka `xdudka00`
Jakub Filák `xfilak01`

BRNO 2008

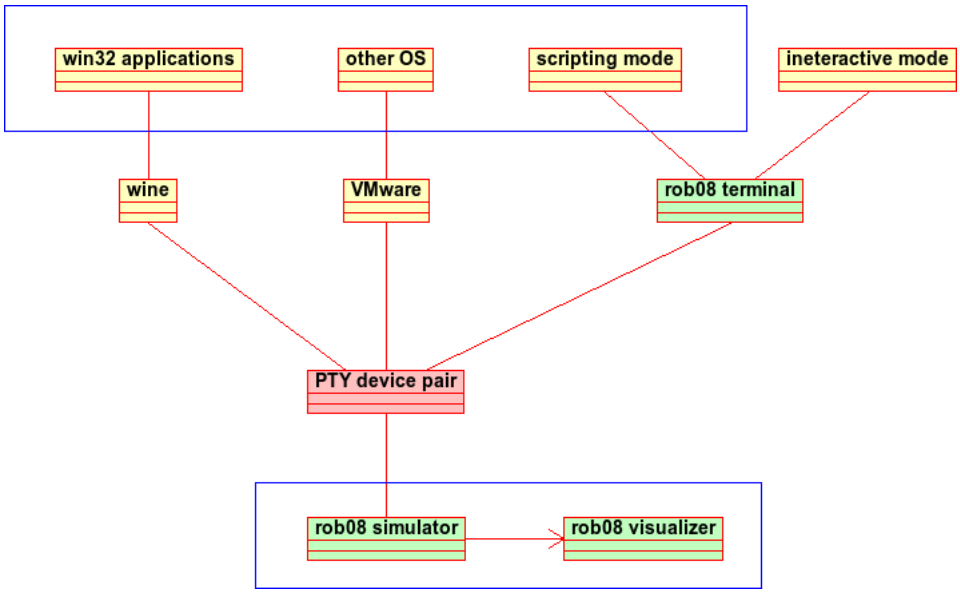
1 Úvod

Jako školní týmový projekt jsme si zvolili simulátor trilobota[1] a jeho prostředí. Simulátor komunikuje s aplikacemi pomocí virtuálního sériového portu. Komunikující aplikace na druhé straně portu tedy neví, že komunikují se simulátorem místo skutečného robota. Tento přístup má několik výhod:

- Práce se simulátorem je stejně „jednoduchá“ jako práce s robotem.
- Není potřeba nijak zasahovat do kódu stávajících aplikací.
- Aplikace používající simulátor mohou běžet na jiném operačním systému.

Simulaci robota a jeho prostředí jsme se snažili udělat co nejvíce věrohodnou a do simulace jsme záměrně zahrnuli některé chyby, které se vyskytují při práci se skutečným robotem (prokluz kol apod.).

2 Návrh



Obrázek 1: Konceptuální schéma simulátoru

[1] Trilobot je typ robota.

Simulátor se skládá z několika částích, které jsou na sobě téměř nezávislé. Klíčovým prvkem celého systému je virtuální sériový port zmíněný výše. Ten je součásti linuxového (případně FreeBSD) jádra a v terminologii jádra se mu říká pseudoterminál (PTY). Jeho úloha je znázorněna na obr. 1.

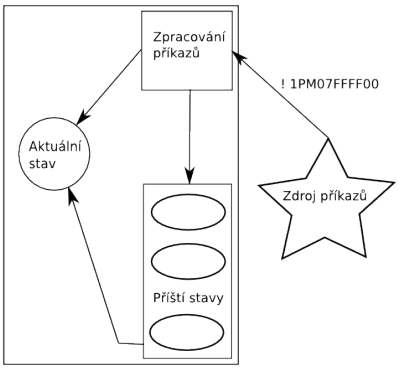
Pseudoterminál vidí uživatelský proces jako pár zařízení. Na jedno zařízení (master) je napojen *proces simulátoru*, který řídí komunikaci. Na druhé stranu (slave) se mohou připojovat aplikace pro řízení robota. Ty mohou se zařízením pracovat přímo, pokud se jedná o linuxové aplikace. Aplikace, které jsou původně napsané pro jiné platformy mohou se simulátorem komunikovat pomocí emulace (wine) nebo virtualizace (VMware).

Speciálním typem klientské aplikace je *terminál*, což je program patřící k našemu projektu. Tento terminál umožňuje se simulátorem komunikovat textem a to ve dvou módech. Prvním módem je skriptovací režim, který je vhodný pro ovládání robota jednoduchými skripty – např. skript `demo.sh`, který je součástí projektu, také používá terminál a jeho skriptovací režim. Naopak interaktivní režim je vhodný pro interakci s uživatelem. V tom případě je k dispozici historie příkazů dříve zadaných, ve které je možné zpětně vyhledávat.

Poslední částí simulátoru je *vizualizátor*, který je oddělený od simulátoru naslouchajícího na virtuálním portu. Procesy simulátoru a vizualizátoru tedy mohou např. běžet na různých strojích, přičemž je zajišťována komunikace relativně úsporným protokolem – posílají se pouze inkrementální změny stavu robota (rychlost, poloměr zatáčky, stav odometru, ...) a plánované události (srážka s překážkou, ...).

Princip simulace chování robota je založen na hierarchickém DEVS formalismu. Rootkoordinátorem celého simulátoru je třída `Trilobot`. Tak jak ve skutečnosti se simulátor trilobota skládá z několika komponent, které ovlivňují trilobotovo chování. Na rozdíl od hierarchického DEVS v našem simulátoru neexistují žádné viditelné propojení mezi komponentami. Komponenty však mohou s ostatními komponentami komunikovat díky tomu, že simulátor má veřejnou množinu komponent. Dalším rozdílem mezi naším simulátorem je to, že zpracování událostí se provádí pouze při příchodu externí události. Náš simulátor tedy nikdy nevykonává interní události, ale veškeré jeho chování, které je způsobeno externí událostí, se musí vytvořit do budoucnosti. To znamená, že se musí vykonat všechny interní události, naplánované po externí události, dokud jsou nějaké plánované. Model simulátoru je znázorněn na obrázku 2.

Chování robota v prostředí simulátoru odpovídá chování robota v reálném prostředí. Jednotlivé komponenty trilobota napodobují chyby, které se vyskytují i ve skutečném trilobotovi. Odpovídající odchylky mezi tím, co bylo trilobotovi zadáno, a tím, co skutečně vykonal, jsme změřili v labora-



Obrázek 2: Stručný náčrt modelu simulátoru

toři a také jsme využili výsledků z práce *Trilobot Control, František Zbořil*. Námi naměřené výsledky jsou uvedeny v tabulce 1.

Příkaz	Ujetá vzdálenost [cm]	Požadovaná vzdálenost [cm]	Poloměr dráhy [cm]	Hodnota enkodéru [HEX]	Požadovaná hodnota enkodéru [HEX]
1PM04002800	28.0	25.4	0	32	28
1PM01002800	25.0	25.4	0	2A	28
1PM07002800	30.0	25.4	0	2E	28
1PM07002820	31.3	25.4	0	30	28
1PM04005040	55.8	50.8	-	58	50
1PM04005041	57.8	50.8	-	5B	50
1PM01006040	62.9	61.0	-	63	60
1PM01006041	63.5	61.0	-	64	60
1PM02008008	84.5	81.3	37	85	80
1PM0200C004	125.1	121.9	67	C5	C0
1PM0100C000	121.0	121.9	0	C1	C0
1PM0700C000	130.8	121.9	0	CE	C0

Tabulka 1: Výsledky měření v laboratoři

3 Sestavení a instalace

Simulátor trilobota je implementován v jazyce C++ s využitím několika volně dostupných knihoven. V aktuální verzi je přeložitelný na systémech GNU/Linux. Pro jeho přeložení je nutné mít překladový systém CMake, program pro tvorbu lexikálních analyzátorů GNU/Flex, vizuální toolkit Qt 4.4+ a C++ knihovnu Boost. Pokud jsou splněny všechny podmínky stačí

pro přeložení spustit příkaz *make* . Po úspěšném přeložení se nacházejí binární soubory v adresáři *build*.

Pro snadnost práce se systémem jsou vytvořeny tři shellové skripty. Pro spuštění demo ukázky chování trilobota v simulátoru stačí spustit skript s názvem `demo.sh`. Skript spustí simulátor, vizualizátor a terminál v příkazovém módu, do kterého pošle sadu ukázkových příkazů. Pro přímé zasílání příkazů trilobotovi stačí spustit skript s názvem `terminal.sh`, který spustí simulátor, vizualizátor a terminál v interaktivním módu. Poslední skript slouží pro spuštění vizuálního nástroje pro řízení trilobota. Tento vizuální nástroj je aplikace vytvořená pro operační systém MS Windows a je proto nutné mít nainstalován program *wine*. Nástroj nese jméno *TrilobotGUI* a nebyl součástí tohoto projektu. Pro spuštění stačí provést skript s názvem `tg.sh`

4 Závěr

Původním cílem bylo zcela nahradit práci v laboratoři pomocí vyvinutého simulátoru. Tento cíl se ukázal příliš optimistický a simulátor jej splnil jen z části. Plnohodnotná simulace robota a jeho prostředí překračuje rámec školního projektu. Přesto je vzniklý simulátor užitečnou pomůckou pro vývoj aplikací na ovládání robota. Díky použité abstrakci sériového portu je možné jej použít i na testování již hotových aplikacích. Není potřeba měnit zdrojový kód aplikací ani je jinak upravovat. Navíc je simulace nezávislá na použitém operačním systému. Objektový model simulátoru je od začátku navrhován, aby byl co nejvíce rozšiřitelný podle potřeb konkrétní aplikace. Při jeho návrhu byly použity návrhové vzory a moderní programovací techniky. Pevně věříme, že bude tento simulátor užitečný.