Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

# DIPLOMOVÁ PRÁCE

**Aplikace testování založeného na modelu testovaného systému na HiL platformě**

PLZEŇ, 2016

Bc. Vojtěch Bohman

SEM VLOŽ ORIGINÁL ZADÁNÍ

# Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejíž úplný seznam je její součástí.

V Plzni dne 31. srpna 2016 .....

**VOJTĚCH BOHMAN**

# Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Miroslavu Flídrovi, Ph.D. a konzultantovi Ing. Ondru Hálovi za cenné rady a připomínky. Dále pak všem kolegům ze společnosti MBtech Bohemia s.r.o., za poskytnuté profesionální rady a materiály.

# Abstrakt

Cílem práce je seznámit čtenáře s přínosem testování založeného na modelu testovaného systému (anglicky Model based testing) a následně aplikovat dané metody na HiL (hardware in the loop) platformu. Práce představuje nástroj MaTeLo, ve kterém byl vytvořen funkční model systému pro generování testovacích případů a nástroj PROVEtech:TA, který spouští a vyhodnocuje vygenerované případy. Během této práce byla zprovozněna platforma pro testování infotainment systémů pomocí MBT testování. Tato platforma bude sloužit pro prezentaci jak klasického testování, tak v této práci představeného MBT testování.

**Klíčová slova:** testování založené na modelu testovaného sytému, HiL, MaTeLo, PROVEtech:TA, řídící jednotka, testování, NI Vision Builder

The aim of this work is to familiarize the readers with the benefits of the model based testing and then apply these methods to hardware in the loop platform. In this work MaTeLo and PROVEtech:TA tools are presented. The usage model is created in MaTeLo and then test cases are generated from that model. These test cases are imported to PROVEtech:TA and then executed and verified. The platform for testing infotainment systems was commissioned during this work. This platform serves as presentation of both the classical and the model based testing.

**Key words:** model based testing, MBT, hardware in the loop, MaTeLo, PROVEtech:TA, ECU, testing, NI Vision Builder

# Obsah

# Seznam obrázků

# Seznam zkratek

**HiL** Testování hardwaru ve smyčce - angl.: Hardware in the Loop

**SiL** Testování softwaru ve smyčce - angl.: Software in the Loop

**MiL** Testování modelu ve smyčce - angl.: Model in the Loop

**XiL** Obecné označení technologie se smyčkou - angl.: General technology with a Loop

**SUT** Testovaný system - angl.: System under test

**ECU** Elektronická řídící jednotka - angl.: Electronic Control Unit

**FIU** Jednotka pro vkládání chyb - angl.: Failure Insertion Unit

**MBT** Testování založené na modelu testovaného systému - angl.: Model Based Testing

**FSM** Konečný automat - angl.: Finite State Machine

**P:TA** PROVEtech:TA

**NI VB** NI Vision Builder for Automated Inspection

**TCP** Primární přenosový protokol - angl.: Transmission Control Protocol

**VBA** Visual Basic for Applications

**ROI** Vybraná oblast - angl.: Region of interest

**SDLC** Životní cyklus informačních systémů - angl.: System/Software Development Life Cycle

**HTML** Značkovací jazyk pro tvorbu internetových stránek - angl.: HyperText Markup Language

**XML** Rozšiřitelný značkovací jazyk - angl.: eXtensible Markup Language

# Úvod

## Motivace

Testování je jeden z nejdůležitějších prostředků ke kontrole správnosti a k měření kvality systémů. Toho je hojně využíváno zejména v průmyslovém odvětví. Bohužel se nachází až na konci vývojového procesu a tak na testování dopadá veškeré zpoždění vzniklé v předchozích fázích. Náklady na testování tvoří přibližně $30 - 50$ % [1] z celkových nákladů na projekt. Z tohoto důvodu existují snahy na snížení nákladů testování například automatizací.

Existuje mnoho různých testovacích technik, procesů, možností a cílů. Tato práce se zabírá aplikováním testovacích metod založených na modelu testovaného systému, anglicky model based testing (MBT). Funkcionální testování se skládá z porovnávání testovaného systému (SUT), vzhledem ke specifikaci. Takový test zaznamená chyby, jestliže se pozorované chování liší od předem specifikovaného. Testování založené na modelu testovaného systému používá funkční modely jako specifikaci a umožňuje z nich automaticky odvozovat testovací případy. V této práci se zaměříme na automatické generování testů pomocí nástroje MaTeLo od společnosti All4tec, který využívá Markovovy řetězce.

Jako testovaný systém bude použit infotainment systém Škoda Bolero, který je momentálně nainstalován v HiL testovací platformě v plzeňské pobočce firmy MBtech Bohemia. Automatizované ovládání systému Bolero zajišťuje kolaborativní robot UR5 (Universal Robots). Pro zpětnou vazbu testovaného systému je použito strojové čtení obrazovky pomocí průmyslové kamery a nástroje NI Vision Builder for Automated Inspection.

Veškerou činnost komunikace mezi nástroji a HiL testovací platformou zajišťuje nástroj PROVEtech:TA (MBtech Group), který se navíc také stará o spouštění a vyhodnocování testů.

Testování je náročná činnost a nikdy nelze pokrýt veškeré chování systému. Náročnost testování leží ve správném výběru testovacích případů pro každou fázi validace. Z těchto důvodů se prezentovaná technologie - MBT, objevuje čím dál častěji. Pomáhá předcházet vadám, zlepšuje kvalitu a snižuje náklady.

# Struktura textu práce

Text práce je členěn do osmi kapitol. V první kapitole **1. Pojem testování** je čtenář seznámen s testováním obecně, jsou zde zmíněny modely vývojových procesů, které se dnes využívají, a základní druhy testování. Na závěr je představeno testování v uzavřené smyčce. To se využívá ke snížení nákladů a zvýšení efektivity testování. Druhá kapitola práce **2. Testování založené na modelu testovaného systému** ukazuje rozdíly mezi klasickým (konvenčním) přístupem k testování a k testování založeném na modelu testovaného systému. Dále jsou zmíněny výhody, nevýhody a limity tohoto přístupu. Ve třetí kapitole **3. Přehled programových nástrojů pro potřeby MBtech Bohemia** jsou stručně popsány veškeré programové nástroje a důvody použití v této práci. Čtvrtá kapitola **4. Software ALL4tec MaTeLo** seznamuje čtenáře s programem MaTeLo, jeho uživatelsky přívětivým rozhraním a představuje parametry důležité při tvorbě modelu a generování testovacích případů. Pátá kapitola **5. PROVEtech:TA a MaTeLo+** se zabývá programem PROVEtech:TA. Ten slouží pro správu, tvorbu, spouštění, vyhodnocování a automatizaci testů. Spolu s tím je popsán plugin MaTeLo+, který zprostředkovává komunikaci mezi nástrojem PROVEtech:TA a MaTeLo. Šestá kapitola **6. HMI Testbench - testovací platforma** čtenáře seznámí s testovací platformou, která je umístěna na pobočce firmy v Plzni. Popisuje hardwarové i softwarové vybavení pracoviště. V sedmé kapitole **7. Software NI Vision Builder for Automated Inspection** je popsán nástroj Vision Builder, který se využívá pro zpětnou vazbu (čte a rozpoznává údaje na displeji rádia). V poslední osmé kapitole **8. Generování, exekuce a vyhodnocení testovacích případů** je ukázán postup vytvoření modelu v nástroji MaTeLo, vygenerování testovacích případů dle požadovaných kritérií jejich spouštění a vyhodnocení v nástroji PROVEtech:TA.

# 1 Pojem testování

## 1.1 Co je a není testování?

S testováním se dnes můžeme setkat všude kolem nás. Uveďme příklad autoškoly, kde v první části probíráme teorii řízení automobilu, dopravní předpisy, značky, poté následují technické vlastnosti vozidla a samotné jízdy. Pokud si přejeme provozovat vlastní automobil, musíme projít zkouškami, které si pro nás komisař připravil. V drtivé většině případů známe okruh otázek, ze kterých se test bude skládat (tzv. specifikaci).

S komisařem poté skládáme písemný test z dopravních předpisů, ústní zkoušení z technických vlastností vozidla a praktickou zkoušku z řízení vozidla (testovací metody). V tomto případě nás komisař hodnotí tak, že porovnává naše odpovědi (či manévry) se správnými.

Zkoušení nemůže být nekonečné, neboť na řadu čeká další uchazeč. Z tohoto důvodu je třeba dobře znát specifikaci testovaného subjektu, resp. požadavky na něj (dopravní předpisy, atd.) a navrhnout vhodné otázky (test).

Můžeme předpokládat, že v průběhu testování dojde k výskytu typických chyb, kterých se uchazeči pravděpodobně dopustí. Otázky by měly být navrženy tak, aby odhalily tyto chyby, a i když odpovědi jsou správné, je nutné připustit určitou prospěchovou toleranci v rámci daného chybového modelu. Při testování elektronických systémů se využívá modelování poruch. Je-li model poruch úspěšně vyzkoušen, přisuzujeme mu věrohodnost. V případě, kdy bude testováno velké, resp. požadované procento modelových chyb, můžeme systém prohlásit za spolehlivý. Testování hledá vady, které (pokud jsou odstraněny) sníží riziko selhání v provozu a jeho přínosem je kvalita a hospodárnost. [3]

**Závada (fault)** - „bug“, je následek pochybení člověka a je příčinou chyby.

**Chyba (error)** - je stav systému a může vést k selhání.

**Selhání (failure)** - je nesoulad mezi aktuálním a specifikovaným chováním systému.[6]

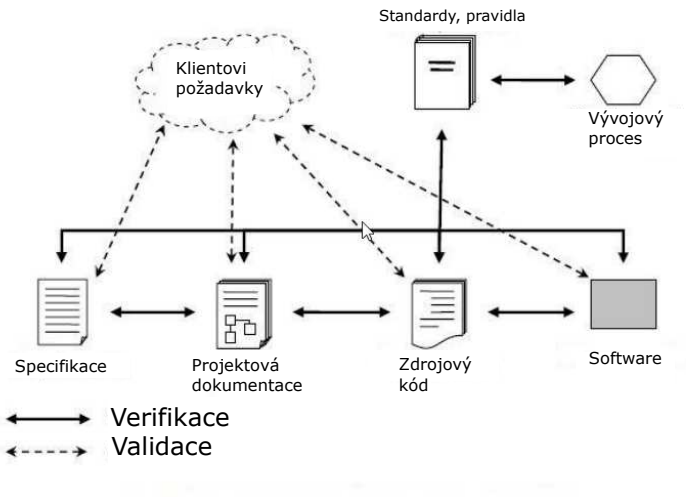
**Testování**

- znamená porovnávat aktuální a očekávané chování.
- znamená detekovat selhání (failure).
- je řízení rizik.
- dává zpětnou vazbu programátorům.

Testování můžeme chápat jako validaci a verifikaci testovaného systému. Tyto dva pojmy jsou částí softwarového inženýrství a bývají velmi často zaměňovány, z tohoto důvodu je vhodné uvést správné definice.

**Validace** je kontrola, zda vyvíjený produkt splňuje dané (zákazníkovo) požadavky. Tedy, že *vytváříme správný produkt*. Mezi metody validace patří např. akceptační testy (testy u zákazníka).

**Verifikace** je kontrola, zda vyvíjený produkt odpovídá našemu návrhu (modelu). Tedy, že *vytváříme produkt správně*. Mezi metody verifikace patří většina testování, kterého se nezúčastní zákazník (integrační testy, unit testy), inspekce kódu, apod.




**Obr. 1.1:** Rozdíl mezi validací a verifikací [4, strana 8]

Dále bychom měli ještě poukázat na rozdíl mezi testováním, což je tedy proces vykonávání/spouštění softwaru pro ověření zda splňuje specifikované požadavky a detekci chyb a tzv. debuggingem[1], což je proces hledání chyb, které způsobují určité selhání.

## Význam a cíle testování v automotive průmyslu

V průběhu celého vývoje elektronických řídících jednotek (Elektronická řídící jednotka - angl.: Electronic Control Unit) v automobilu probíhají rozsáhlé testy. Samozřejmostí je i samotné otestování konečného produktu. Cílem testování vozidel je, aby byla dosažena určitá úroveň konečného produktu (automobilu). Zároveň k tomu patří také ověření posledních vývojových kroků a v neposlední řadě také ověření vlastností vozidla vzhledem k očekávání zákazníka.

## Testovací případ

Je také vhodné uvést definici tzv. testovacího případu. Testovací případ popisuje konkrétní akce prováděné se systémem a jejich očekávané výsledky. Testovací případ je tedy dokument, popisující určitou činnost, kterou je třeba otestovat.

[1]Debugging, nebo-li ladění je v informatice metodický postup pro nalézání a snižování množství chyb v počítačových programech nebo elektronického hardware tak, aby fungoval, jak se předpokládá.

## 1.2 Model vývojového procesu

Je nutné si uvědomit, že testování není jednorázový proces, který proběhne na konci vývoje softwaru pro ujištění, že je vše v pořádku. Testování je svázáno s celým vývojem a jeho úkolem je dosáhnout stavu, kdy na konci procesu vývoje bude systém obsahovat jen minimum chyb.

Návrh jakkoliv složitého systému je nutné rozložit na řadu dílčích kroků. Tento postup je převzat ze softwarového inženýrství a nazývá se životní cyklus informačních systémů (SDLC). Každý model procesu má své výhody a nevýhody. Klíčem k úspěšnému vývoji požadovaného systému je správný výběr modelu, který je založen na požadavcích a potřebách daného projektu (produktu, systému). Následuje přehled několika používaných modelů.

### 1.2.1 Vodopádový model

Tento model je jedním z historicky nejstarších. V roce 1956 byl představen H.D. Beningtonem. Jedná se o nejjednodušší lineární neboli přímočarý model. Ve vodopádovém modelu jednotlivé etapy procesu navazují na sebe s minimálním časovým překrytím, to znamená, že v jeden okamžik se pracuje právě na jedné části projektu. Před završením každé z etap jsou výsledky vždy ověřovány a schváleny. V praxi je tento přístup nevhodný, neboť u netriviálních projektů není možné ukončit jednu etapu předtím, než se přejde k další. Například klient požaduje změnit specifikace v průběhu vývoje a tím se celý cyklus vrací na začátek.

\+ jednoduše použitelný, snadno pochopitelný

\- veškeré požadavky musí být známy na začátku projektu

\- vysoké náklady při změně požadavků

### 1.2.2 Spirálový model

Tento model pokrývá největší nedostatky předchozího vodopádového modelu. Spirálový model je založen na iterativním přístupu a především zavádí opakovanou analýzu všech rizik. Takto lze lépe reagovat na pozdější úpravu specifikace. Model probíhá v několika

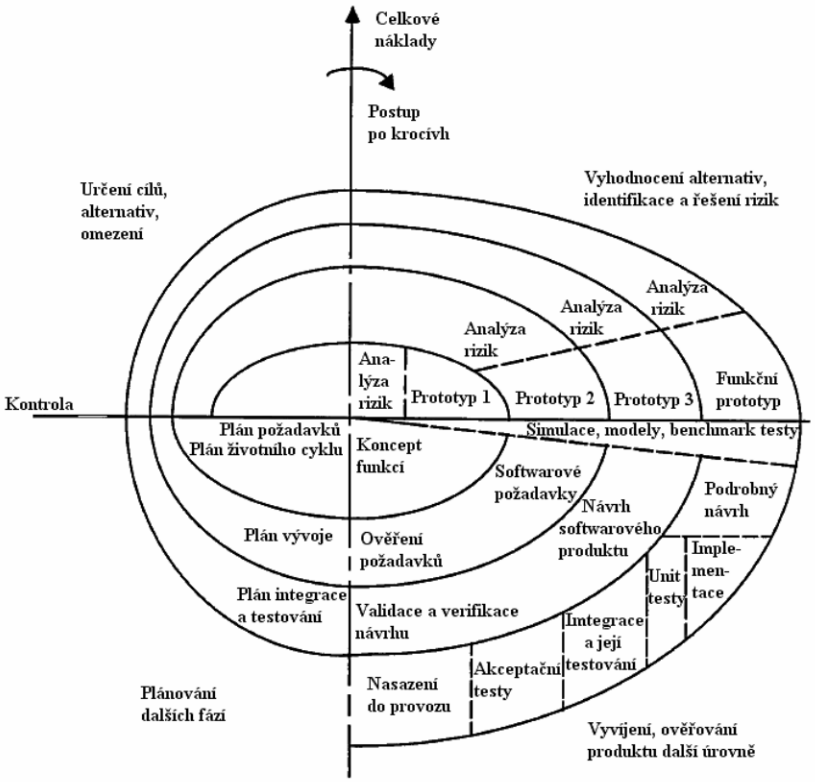
krocích, které se neustále opakují, dokud není celý systém hotový. Celý cyklus je rozdělen do čtyř hlavních částí. Každá fáze je následována testováním a hodnocením. Systém je tedy pravidelně testován již od raných fází. Díky tomuto přístupu dochází k včasnému odhalení chyb a je tak možné snadno upravit analýzu. Fáze spirálového modelu jsou: 1. Určení cílů, alternativ, omezení, 2. Vyhodnocení alternativ, identifikace a řešení rizik, 3. Vývoj a verifikace další úrovně produktu, 4. Plánování následujících fází.

\+ odhad rizik

\+ umožňuje řešit modifikované požadavky v jednotlivých fázích

\- časová náročnost odhadu rizik

\- neustálá spolupráce s klientem

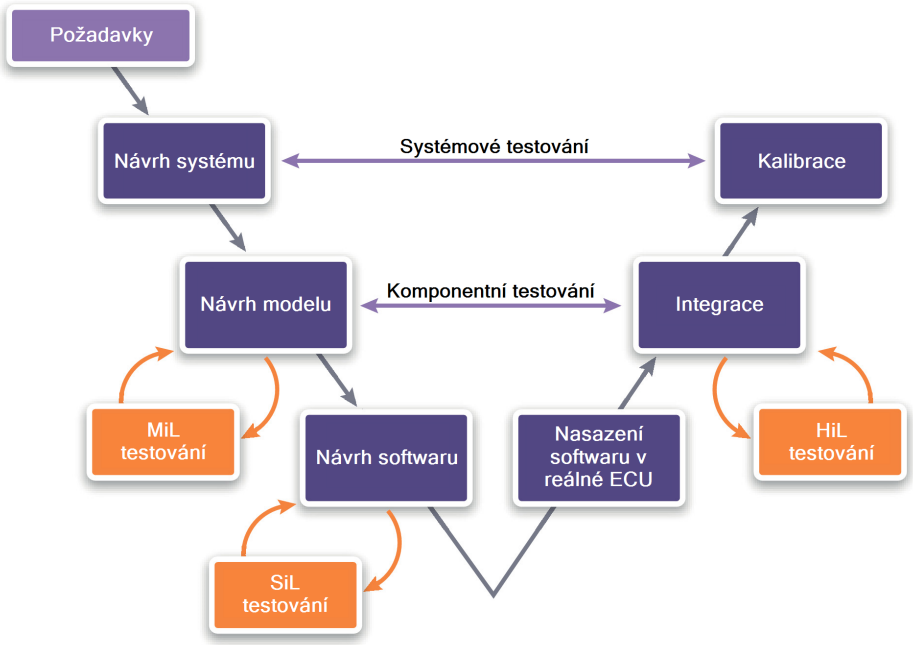



**Obr. 1.2:** Spirálový model vývojového procesu [2, strana 7]

### 1.2.3 V–model

V současné době je ve vývoji automobilových řídicích systémů hojně používaný tzv. V–model. Tento model vychází z vodopádového modelu. V–model je v místě implementace zalomen tak, že tvoří písmeno „V“. Tento přístup nám umožňuje zlepšit účinnost testování a snížit náklady na odstranění chyb. Levou stranu modelu tvoří vývojové kroky, které jsou následně na pravé straně testovány. Základem tohoto přístupu je postup testování od malých částí (komponentní testování), po testování komplexních systémů v automobilu (integrační testování). Přechod na další druh testování předpokládá úspěšné dokončení předcházející fáze.

Výhodou V–modelu je možnost testování paralelně s vývojovou částí. Tím se minimalizují náklady na odstranění případných chyb.




**Obr. 1.3:** V–model vývojového procesu

## 1.3 Druhy testování

Testování může být rozděleno do několika kategorií. Dva z nejvlivnějších aspektů jsou znalost a pozorovatelnost vnitřního stavu systému zatíženým testem. V následujícím textu je představeno tzv. Black-box, White-box a Grey-box testování.

### 1.3.1 Black-box (funkční) testování

Při tomto testování se zaměřujeme na vstupy a výstupy systému, tj. neznáme žádné informace o vnitřní struktuře. Daný systém je pro nás černou skřínkou a smyslem je analyzovat chování systému vzhledem k očekávaným vlastnostem tak, jak ho vidí uživatel. Výhodou tohoto přístupu je testování systému na nejvyšší úrovni. [7] Testování založené na modelu testovaného systému je právě black-box testování.

### 1.3.2 White-box (strukturální) testování

Při testování bílé skřínky má tester přístup k vnitřním stavům systému, zná vstupy, výstupy i reakce systému. To mu dává lepší možnosti hledání případných chyb. Testování je založeno na využití znalostí o konstrukci systému, proto je často nazýváno strukturní testování. Tento přístup je vhodný k testování na nižších úrovních. Výhodou je velmi dobré diagnostické pokrytí a zajištění otestování hraničních a speciálních stavů. Naopak nevýhodu ve srovnáním s Black-box testováním tvoří silná závislost na vlastní implementaci algoritmu. To znamená, že v případě změny musí být vytvořené testy přepracovány. [7]

### 1.3.3 Gray-box testování

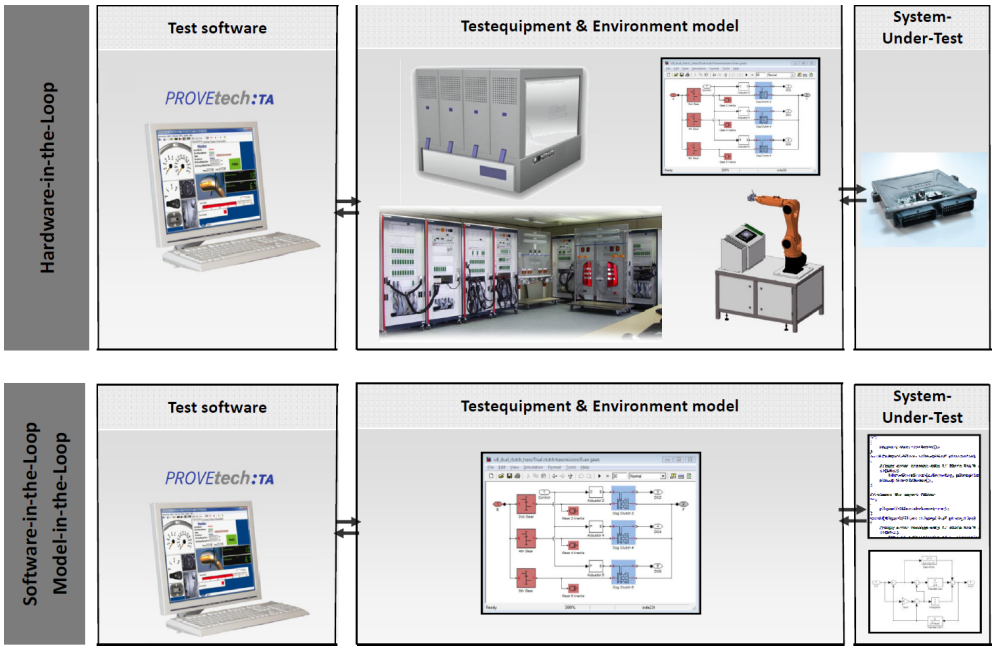
V tomto případě se kombinují výhody obou předešlých způsobů. Využitím této techniky se vytváří testy na úrovni bílé skřínky (jsou známy vnitřní stavy) a vyhodnocují se jako u černé skřínky (udržuje si realistický pohled). [7]

## 1.4 Testování v uzavřené smyčce

Každým rokem roste počet elektronických systémů využívaných v automobilovém průmyslu. Souvislosti můžeme najít ve zvyšování bezpečnosti při provozu automobilu a také zvyšující komfort zákazníků. Tento trend také zvyšuje požadavky na spolehlivost daných systémů. Z tohoto důvodu jsou na testování elektronických systémů kladeny vysoké nároky. Testovaným systémem může být ECU nebo také infotainment systém[2]. Nejčastěji se používá tzv. testování v uzavřené smyčce.

### 1.4.1 MiL testování

První integrační level je založen na modelu samotného systému. Testování systému na MiL (Testování modelu ve smyčce - angl.: Model in the Loop) levelu znamená, že model a jeho prostředí jsou simulovány bez jakéhokoliv fyzického hardwaru. To umožnuje testování v rané fázi vývoje.




**Obr. 1.4:** Testování ve smyčce

[2]Infotainment je složenina dvou z angličtiny pocházejících slov: information (informace) a entertainment (zábava)

### 1.4.2 SiL testování

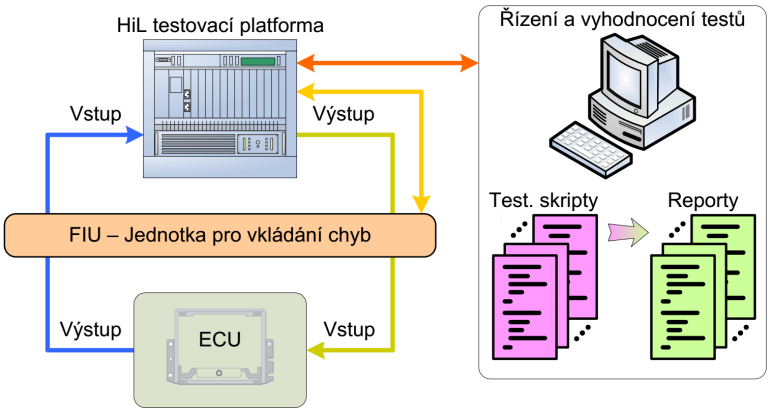
Testování na úrovni SiL (Testování softwaru ve smyčce - angl.: Software in the Loop) znamená, že software je testován v rámci modelu simulovaného prostředí, ale bez použití jakéhokoliv hardwaru (žádné senzory, aktuátory). Ve většině případů běží software i simulované prostředí na stejném počítači, ve virtuálním prostředí. Z tohoto důvodu není nutné používat real-time počítače a postačují klasické stolní.

### 1.4.3 HiL testování

Testování hardwaru ve smyčce - angl.: Hardware in the Loop (HiL) je metoda, při které se reálnému testovanému objektu (například již zmíněné ECU) simuluje elektromechanické prostředí jako motor, převodovka, apod. Méně náročné komponenty (světla, palubní KOMBI přístroje, zrcátka) bývají součástí testovací platformy. Software tedy běží na finální verzi testovaného objektu.

Cílem testování na HiL úrovni je objevit vady na nízkoúrovňových (low-level) a vstupně-výstupních (I/O) službách ECU. HiL testování vyžaduje real-time chování okolního prostředí, aby byla zajištěna stejná komunikace ECU ⇔ prostředí jako je u reálné aplikace.

Nespornou výhodou tohoto přístupu je možnost dosáhnout extrémních stavů, kterých se za reálných podmínek těžko dosahuje.




**Obr. 1.5:** Princip HiL testování. Převzato z [3, strana 16]

## Princip HiL testování

Na (obr. 1.5/str. 11) je znázorněn základní princip propojení HiL testovací systému. Fyzická ECU je ve smyčce připojena k HiL testovací platformě (mezi tyto systémy muže být vložena jednotka pro vkládání chyb - angl.: Failure Insertion Unit - FIU). Testovací platforma obsahuje různé procesorové karty a napájení testovaného zařízení. Úkolem platformy je běh modelu prostředí v reálném čase a předávání signálů mezi ECU a operátorským stanovištěm.

V případě operátorského stanoviště se jedná o klasický stolní PC platformy $x86/x64$. Na tomto PC běží nástroj PROVEtech:TA (P:TA) (kap. 5.1/str. 25), který se stará spouštění testů, zaznamenávání a uchovávání jejich výsledků.

# 2 Testování založené na modelu testovaného systému

## 2.1 Co je to Model Based Testing

Testování založené na modelu testovaného systému je proces, při kterém se z funkční specifikace vytvoří model daného systému. Z modelu jsou následně automaticky generovány testovací případy, které jsou poté spouštěny a vyhodnocovány. MBT umožňuje zajistit opakovatelnost testů a odstraňuje chyby, kterých se může dopustit tester při manuálním či skriptovacím testování. MBT automatizuje black-box testování (kap. 1.3.1/str. 9), tedy popisuje, co systém dělá, a většinou se skládá z následujících čtyř kroků: [8][9]

1. **Vytvoření abstraktního modelu testovaného systému.** Tento krok je podobný procesu vytvářnení testovací specifikace.

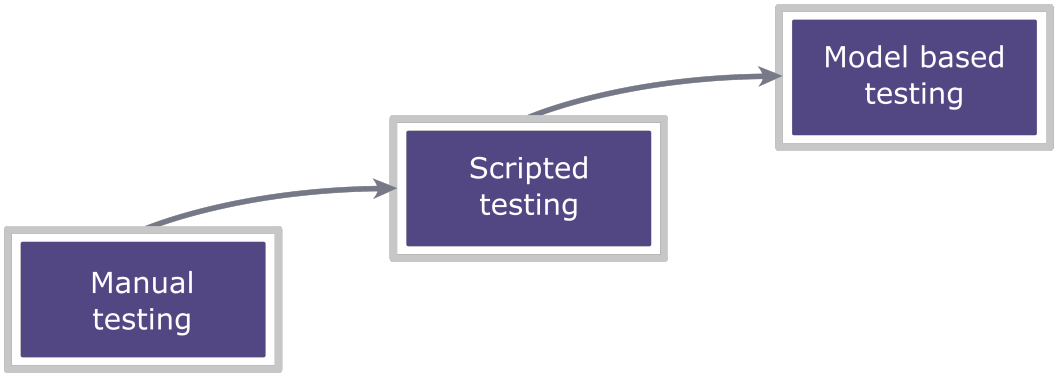
2. **Validace modelu** - provádí se ke zvýšení kvality modelu. Pokud v modelu zůstanou chyby, jsou odhaleny při exekuci vygenerovaných testů a model se poté jednoduše opraví.

3. **Generování abstraktních testů z modelu.** Tento krok je automatický. Tester má možnost nastavovat různé parametry, například: jaké části systému se mají testovat, kolik testů má být vygenerováno, atd.

4. **Přeložení abstraktních testů do jazyka spustitelných testů.** Tento krok je prováděn automaticky (viz. kap. 5.2/str. 31).

V případě této práce se o první tři kroky stará nástroj MaTeLo a poslední krok provádí plugin MaTeLo+ nástroje P:TA, v tomto nástroji se poté spouští a vyhodnocují vygenerované testy (kap. 5.1/str. 25).

## 2.2 Vývoj testování




**Obr. 2.1:** Vývoj testování

### 2.2.1 Manuální testování

Manuální testování je, jak z názvu vyplývá, proces, při kterém tester ručně testuje systém bez jakéhokoliv nástroje nebo automatického skriptu. Tester postupuje podle testovacího plánu, který slouží jako manuál tak, aby byly pokryty veškeré požadavky na systém.

**Manuální testování**

- může obsahovat chyby způsobené testerem ⇒ je méně spolehlivé
- je náročné jak časově, tak na lidské zdroje
- je vhodné pouze pro testování s malým počtem a opakováním testovacích případů

### 2.2.2 Skriptovací testování

V tomto případě tester navrhne testy podle specifikace. Ty jsou následně spouštěny v nástroji tomu určeném. Výhodou je možnost automatického a opakovatelného spouštění testů bez zásahu člověka.

**Skriptovací testování**

- je více spolehlivé než manuální. Spuštění a vyhodnocení testů provádí nástroj
- je méně časově náročné
- je vhodné při dlouhodobém a často opakovaném testování

### 2.2.3 Model Based Testing

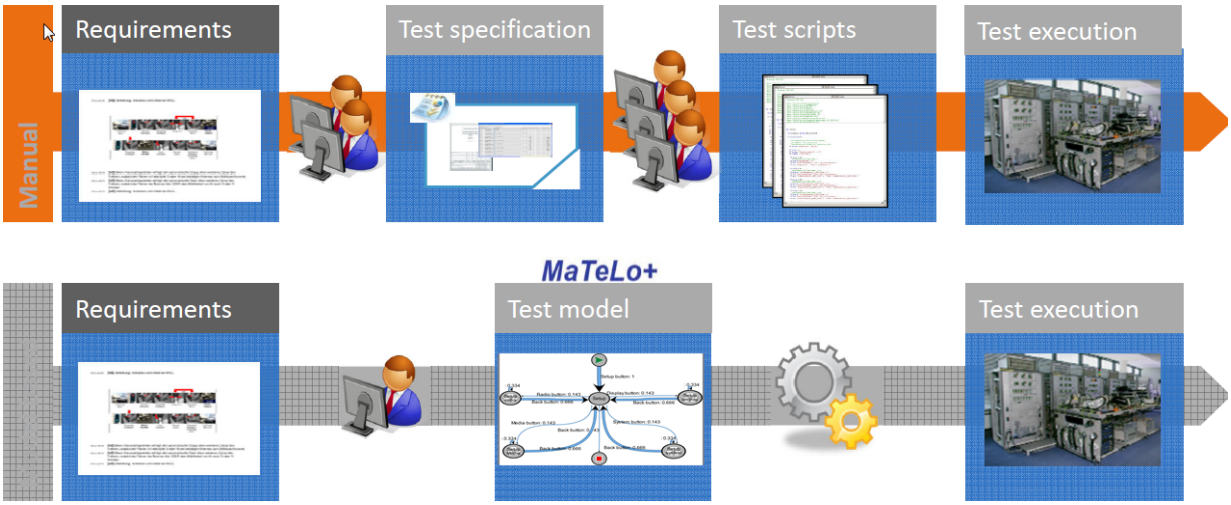
Díky možnosti nastavovat různé parametry při generování testovacích případů může tester jednoduše měnit strategii testování, také má možnost soustředit se pouze na nejvíc využívané funkce a nejpravděpodobnější chování systému. Nebo se může zaměřit na kritické části, může vygenerovat testovací případy, které zahrnují veškeré požadavky, nebo použít náhodné generování. Na základě výsledků (porovnání mezi chováním testovaného systému a výsledkem testovacího případu), je tester schopen upravit požadavky na systém tak, aby odstranil chybné chování a tím vylepšil daný systém.

**Testování založené na modelu testovaného systému**

- umožňuje jednoduchou správu testovacích případů
- snižuje náklady a časovou náročnost na testování
- umožňuje časnější odhalení chyb a zvyšuje počet nalezených chyb ⇒ roste kvalita
- nevyžaduje žádné programovací znalosti
- umožňuje generovat vysoký počet testovacích případů ⇒ roste pokrytí

## 2.3 Limity MBT

Ačkoliv MBT nabízí řadu pokrokových a užitečných vlastností, má i svá úskalí. Fundamentální omezení tohoto přístupu je nemožnost garantovat nalezení všech rozdílů mezi modelem a implementací, ani kdybychom zajistili nekonečný počet testů a spouštěli je po dobu tisíce hodin. Na druhou stranu toto platí obecně pro testování.




**Obr. 2.2:** Konvenční vs MBT testování

Dalším omezením mohou být odlišné role testerů. Návrháři musí umět abstraktně popsat a navrhnout model systému, což obnáší dokonalou znalost aplikační oblasti.

Jestliže je při analyzování výsledků nalezen chybně vyhodnocený test, musí tester rozhodnout, zda je chyba v modelu či v testovaném systému. Protože MBT umožňuje generovat velké množství testů a ty jsou méně intuitivní než u konvenčního testování, může být hledání důvodu časově náročná záležitost.

U konvenčního testování se často používá počet úspěšně vyhodnocených a ověřených testů jako měřítko progresu testování daného systému. To v případě MBT nelze využít a musí se přejít na jiné přístupy jako: pokrytí požadavků nebo pokrytí modelu. [9]

# 3 Přehled programových nástrojů pro potřeby MBtech Bohemia

V této diplomové práci je využíváno několik programových nástrojů. Hlavním důvodem použití níže jmenovaných nástrojů je jejich dostupnost ve firmě MBtech Bohemia.

### PROVEtech:TA (MBtech Group)

Nástroj P:TA (Version 2016 SE SP1) vyvíjený firmou MBtech Group slouží obecně pro automatizaci testování. V našem případě byl také využitý pro komunikaci s dále jmenovanými nástroji a robotem UR5. Výstupem tohoto nástroje jsou protokoly o výsledcích testů spuštěných na HiL testovací platformě. Součástí P:TA je i plugin MaTeLo+ (Version 2015 SP1) vyvíjený ve spolupráci s firmou ALL4TEC. Tento plugin zajišťuje předávání dat mezi P:TA a MaTeLo.

### MaTeLo (ALL4TEC)

Nástroj MaTeLo (V5.3.0 Descartes) umožňuje automatizovat proces testování s využitím testování založeného na modelu testovaného systému. Na základě specifikace je vytvořen model systému, ze kterého jsou generovány testovací případy. Na rozdíl od konvenčního testování, kdy tuto činnost musí provádět expert.

### Vision Builder for Automated Inspection (National Instruments)

K získání zpětné vazby a kontrole stavu systému je využit nástroj Vision Builder for Automated Inspection (Version 2015). Slouží pro snímání obrazu a vyhodnocování výsledků ze získaného snímku, které dále posílá do P:TA.

Podrobné informace o představených programech a jejich použití v této práci je popsáno v následujících kapitolách.

# 4 Software ALL4tec MaTeLo

Nástroj MaTeLo (Markov Test Logic) vyvíjený firmou ALL4TEC poskytuje uživatelsky přívětivé prostředí pro testování založeného na testovaném systému, za účelem snížení nákladů a času testovací fáze vývoje systémů. Nástroj používá koncept Markovových řetězců pro vytvoření modelu testovaného systému [1].

Model se skládá z přechodů, při kterých se simulují stimuly systému a ověření očekáváné reakce a ze stabilních stavů, ve kterých se systém nachází před příchozím stimulem. Tester navrhuje model na základě funkčních požadavků a ten může být použitý pro black-box testování (kap. 1.3.1/str. 9) ve všech XiL systémech (kap. 1.4/str. 10).

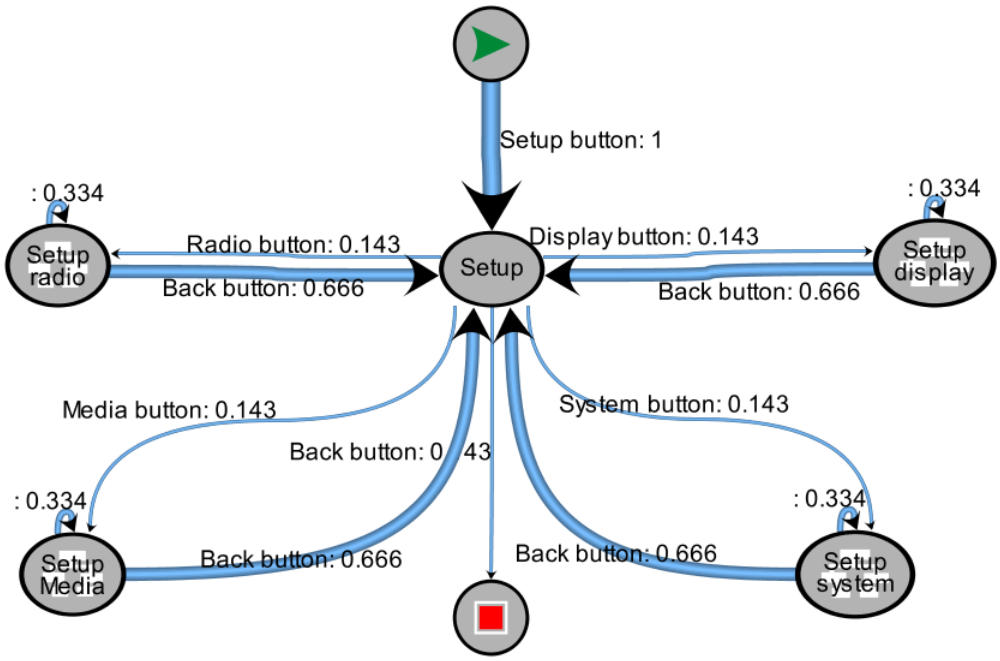
### Markovovy řetězce

Markovovy řetězce jsou efektivní metodou pro reprezentaci MBT. Model systému vytvořený na základě Markovových řetězců lze chápat jako funkční model. Funkční modely jsou převážně konstruovány ze dvou částí: Konečný automat - angl.: Finite State Machine (FSM), který reprezentuje všechny scénáře používání testovaného systému a profily, které kvalifikují FSM jak bude systém používán ze statistické stránky. FSM určují, co může nebo má být testováno a profily umožňují odvozovat testovací případy na základě pravděpodobnosti.

## 4.1 Pracovní plocha nástroje MaTeLo

Nástroj MaTeLo (verze 5.3 - Descartes) se skládá ze tří základních oken: Edition, Generation, Reports. Poslední okno zůstává v případě této práce nevyužité, testy se spouští v P:TA, který má návaznost na hardware.

Okno Edition (obr. 4.2/str. 20) slouží pro správu projektů, importování testovacích funkcí/signálů nejen z nástroje P:TA a pro tvorbu funkčního modelu testovaného systému. Potřebné signály a funkce nejprve vygenerujeme z nástroje P:TA a poté importujeme do požadovaného projektu.




**Obr. 4.1:** Jednoduchý model procházení menu nastavení

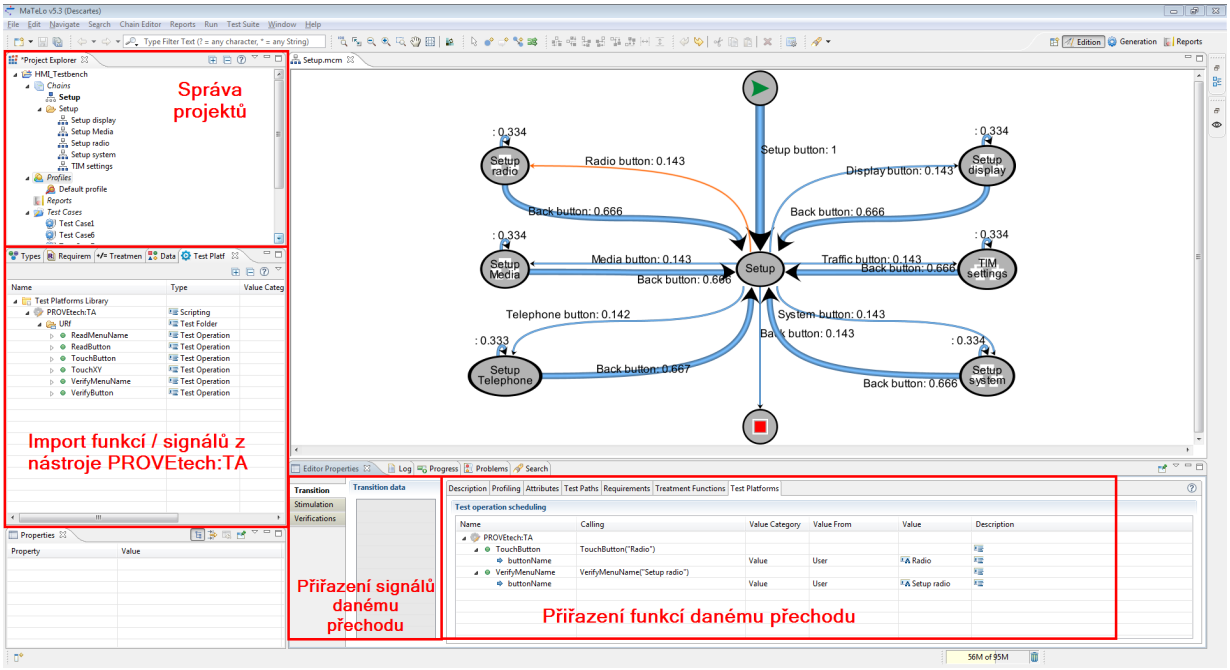
**Přechod (Transition)** Přechody reprezentují změnu signálu či akci uživatele a spojují ustálené stavy systému. Každý přechod může mít v MaTeLo několik profilů s různou pravděpodobností nastání jevu. U každého přechodu je vhodné uvést jeho název, popřípadě detailnější popis. V MaTeLo je možné vést vždy jeden přechod ze stavu A do stavu B, pokud potřebujeme přidat další přechod mezi těmito stavy, je nutné vytvořit prázdný stav S a cestu vést $A \rightarrow S \rightarrow B$.

**Stav systému (States)** Stavy jsou definovány jako uzel mezi dvěma přechody a reprezentují stav systému po stimulaci. Každý MaTeLo projekt má kořenový uzel, který značí začátek (Invoke) a konec (Terminate). Každý stav je možné pojmenovat a připsat detailnější informace.

**Subsystém (Macro States)** Pokud vytváříme složitější modely testovaného systému, je vhodné využít subsystémy (Macro states). Každému stavu lze přiřadit jeden subsystém.

Na obrázku (4.1) vidíme jednoduché procházení nabídky. Kdy kliknutím na tlačítko *Setup* se dostáváme do menu nastavení a máme zde několik možností. Za prvé kliknout na

tlačítko *Radio* a otevřít příslušná nastavení. Za druhé kliknou na tlačítko *Display* a otevřít příslušná nastavení nebo za třetí kliknout na tlačítko *Back* a opustit menu.




**Obr. 4.2:** Pracovní plocha nástroje MaTeLo

Obrázek 4.2 zobrazuje pracovní plochu nástroje MaTeLo. V levé části se nachází okno pro správu projektů. Zde je možné vytvářet nové či upravovat stávající projekty. Dále je také možné importovat projekty ze starších verzí nástroje MaTeLo. Při vytvoření nového projektu se také vytvoří základní (Root) řetězec. Pokud potřebujeme přidat další subsystémy lze to provést právě v tomto okně. V dalším okně jsou spravovány veškeré funkce a signály, které budou používány při modelování systému. Pro importování P:TA funkcí vybereme záložku *Test Platform* a vybereme možnost *Import Scripting Library*. Požadovaný soubor vygeneruje plugin MaTeLo+ (kap. 5.2/str. 31).

Ve spodní části pracovní plochy se nastavují parametry vybraného přechodu či stavu. V případě přechodu uvádíme v záložce *Description* název přechodu, v záložce *Profiling* nastavujeme pravděpodobnost vybraného přechodu pro každý vytvořený profil a v záložce *Test Platforms* přiřazujeme importované P:TA funkce. U stavů jsou možnosti omezenější. Zde se nastavuje název v záložce *Description* jeho název, případně vybereme subsystém, do kterého se má přejít z tohoto stavu. Názvy by měly stručně popisovat prováděné akce a ustálené stavy systému.

## 4.2 Okno pro generování testovacích případů

Dalším krokem po vytvoření modelu je volba testovací strategie a nastavení parametrů pro generování testovacích případů. K tomu slouží panel Generation. V prvním okně lze vybrat z dříve vytvořených strategií (pokud nějaké existují) nebo vytvořit novou. Další okno nám umožnuje volbu mezi testováním celého systému nebo jeho částmi (pokud je model systému rozdělen do více částí).

Pro každý projekt v MaTeLo je možné mít více profilů, u kterých lze nastavit různé pravděpodobnosti akcí. V panelu Generation poté volíme, který profil, tedy jaké pravděpodobnosti přechodů, bude brán v potaz při generování testovacích případů.

Jako poslední je vybrán algoritmus a jeho parametry. Využití algoritmů se bude lišit podle požadavků na testování systému. Všechny algoritmy a jejich parametry jsou popsány v následující sekci.

### 4.2.1 Parametry testové strategie

#### Random

Tento algoritmus generuje testovací případy náhodně bez ohledu na zadané pravděpodobnosti přechodů. Lze nastavit dva parametry:

**Maximální počet kroků** Definuje maximální počet kroků pro každý testovací případ. Výchozí hodnota je 10.

**Počet testovacích případů** Definuje maximální počet vygenerovaných testovacích případů. Výchozí hodnota je nastavena na maximum, tedy 5000.

#### User oriented

Tento algoritmus generuje testovací případy s ohledem na zvolený profil (pravděpodobnosti) přechodů. Lze nastavit následující parametry. (Zmíněná třída ekvivalence je využitelná pouze při práci se signály. V takovém případě se signálům vybere rozmezí hodnot a MaTeLo poté volí vstupní hodnoty v daných mezí.)

**Maximální počet kroků**

**Počet testovacích případů**

**Třída ekvivalence** Definuje přístup ke generování stimulací podle třídy ekvivalence. Na výběr je několik možností.

- Náhodné - Náhodně vybere hodnotu bez použití vybraného profilu.
- Uživatelsky orientovaný - Vybere hodnotu v souladu s pravděpodobnostmi zvoleného profilu.
- Hodnoty blízké dolnímu limitu
- Hodnoty blízké hornímu limitu
- Hodnoty blízké dolnímu nebo hornímu limitu
- Průměr - Zvolí průměrnou hodnotu.
- Nejvíce pravděpodobné - Vybere nejpravděpodobnější rozsah distribuce v závislosti na zvoleném profilu.

## Most probable

Tento algoritmus generuje testovací případy s ohledem na zvolený profil (pravděpodobnosti) přechodů. Jsou procházeny nejvíce pravděpodobné přechody nebo třídy ekvivalence. Lze nastavit tyto parametry:

**Pokles míry** Používá se pro snížení pravděpodobnosti přechodu, který již byl vybrán během generování. Například pokud je nastavena hodnota 50: Je-li vybrán přechod ze stavu T s pravděpodobností $0,5$, při dalším průchodu tímto stavem bude pravděpodobnost přechodu $0,25$. Tím pádem může být vybrán jiný přechod s vyšší pravděpodobností (pokud existuje).

**Počet testovacích případů**

**Třída ekvivalence** Parametry jsou stejné jako u algoritmu definovaného uživatelem až na následující.

- Nejpravděpodobnější s klesající mírou: Vybere nejpravděpodobnější rozdělení akcí a použije snižování pravděpodobnosti přechodů.

## User oriented + Filter

Tento algoritmus generuje testovací případy s ohledem na zvolený profil (pravděpodobnosti) přechodů a zvoleného filtru. Cílem je vytvořit nové testovací případy na základě stávajících případů v projektu. Lze nastavit tyto parametry:

**Maximální počet kroků**

**Počet testovacích případů**

**Maximální počet pokusů**

**Filtr** Parametr pro nastavení filtru.

- Žádný - Nebude použit filtr.
- Pouze cesty - Cesty testovacích případů se musí lišit.
- Cesty a třídy ekvivalence - Cesty testovacích případů se mohou shodovat, zatímco třídy ekvivalence se musí lišit.
- Pokrytí přechodů - Pokud dříve vygenerovaný testovací případ pokrývá jiné přechody než ty aktuálně vygenerované, zůstane zachován.

**Třída ekvivalence**

## Tagged Path

Tento algoritmus slouží pro znovu vygenerování testovacích případů, které byly dříve uložené jako testovací cesta. Jakýkoliv testovací případ lze uložit jako cestu a tuto cestu poté zobrazit v modelu.

**Testovací cesta**

**Třída ekvivalence**

## Minimum (Arcs Coverage)

Tento algoritmus slouží pro generování nejkratší cesty, pokrývající všechny přechody modelu.

**Počet cyklů** Definuje počet cyklů pro zlepšení výsledného řešení.

**Třída ekvivalence**

**Filtr přechodů** Definuje zda-li budou při generování testovacích případů brány v potaz pravděpodobnostní profily přechodů.

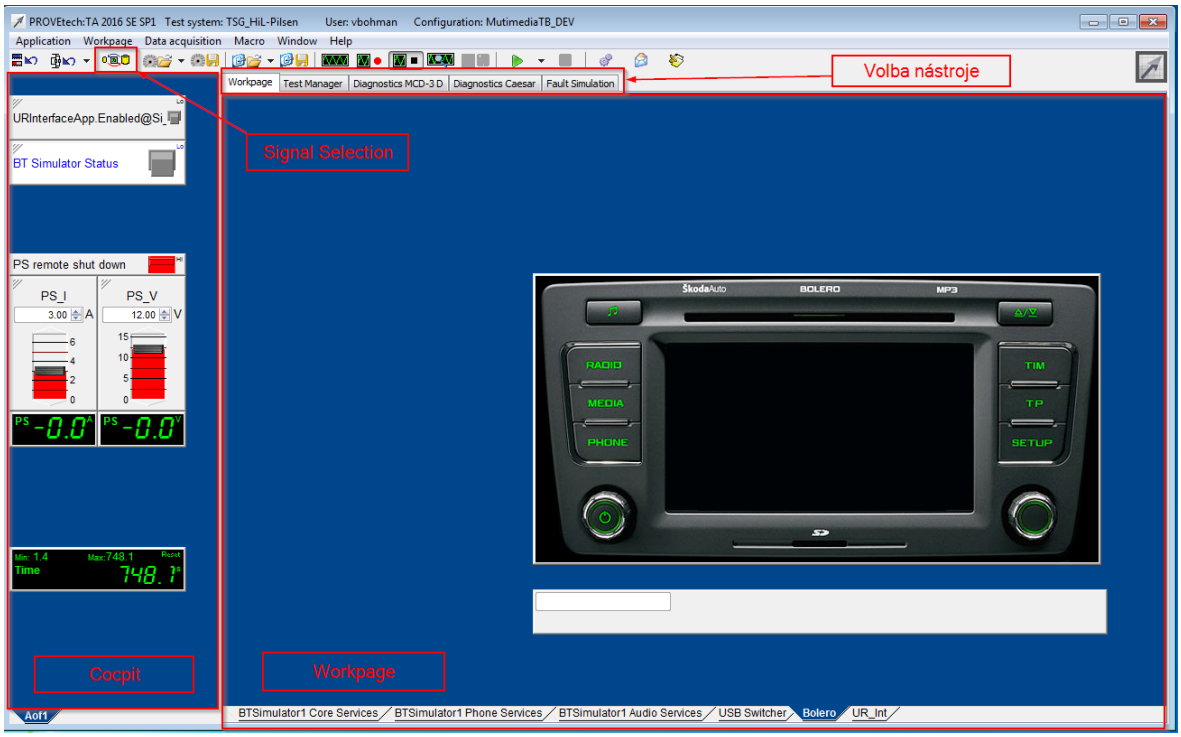
### 4.2.2 Vizualizace vygenerovaných případů

MaTeLo umožňuje uživateli přehledné zobrazení všech testovacích případů i podrobný pohled na jednotlivé kroky. První možností zobrazení je seznam provedených kroků s informacemi jako číslo kroku, název přechodu, počáteční a koncový stav, název subsystému a názvy volaných funkcí. Druhou možnost tvoří zobrazení ve formátu HTML s informacemi jako číslo kroku, název stavu a název přechodu.

# 5 PROVEtech:TA a MaTeLo+

## 5.1 Software pro automatizaci a správu testů PROVEtech:TA

V následující kapitole se seznámíme s nástrojem PROVEtech:TA[1] vyvíjeným společností MBtech Group. Tento software slouží k ovládání a automatizaci testovacího procesu (implementace testů, spouštění, vyhodnocování). Námi využívaná verze nástroje se skládá ze čtyř základních částí, mezi kterými volíme formou záložek. Jedná se o Workpage, Test Manager, Diagnostics a Fault Simulation. Poslední dvě části budou popsány jen okrajově, v případě této práce se nepoužívají.




**Obr. 5.1:** PROVEtech:TA

[1]PROVEtech:TA – zkratka pro Test Automation

Tento nástroj dovoluje uživateli snadno nastavovat a měřit potřebné signály testovaného systému. Dále poskytuje databázi základních ovládacích prvků a funkcí pro snadnou vizualizaci a správu testovacích knihoven, skriptů a jejich výsledků. PROVEtech:TA je využitelný po celou fázi vývoje produktu:

1. Testování modelu ve smyčce - testování modelu ECU
2. Testování softwaru ve smyčce - testování softwaru ECU ještě předtím, něž je vyrobena
3. Testování hardwaru ve smyčce
   - komponentní testování - testování samostané ECU
   - integrační testování - testování více ECU a komunikace mezi nimi (například ve vozidle)

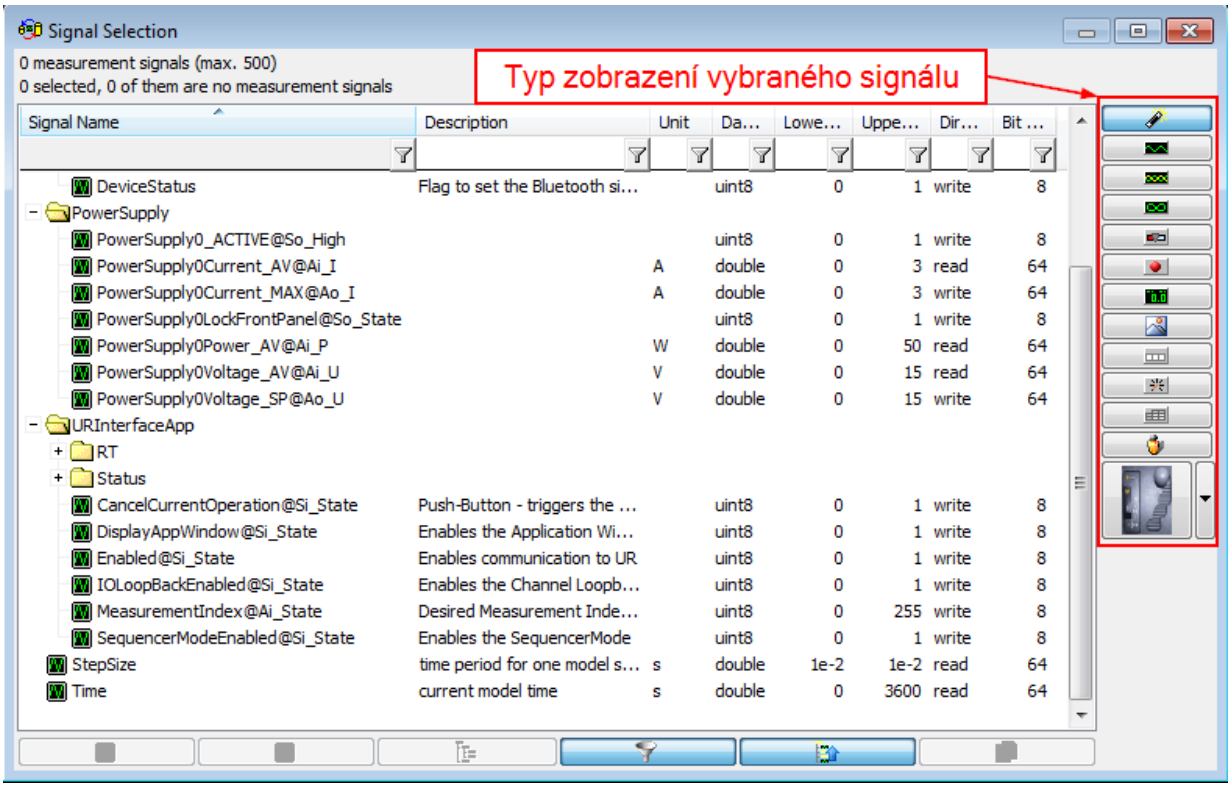
Velkou výhodou nástroje PROVEtech:TA je možnost spolupracovat jak s různými HiL simulátory (dSPACE, ETAS), tak i s modely z prostředí Matlab/Simulink (formát .dll). Tímto lze provádět MiL testování (kap. 1.4.1/str. 10).

### 5.1.1 Workpage

První modul je pojmenován *workpage* a reprezentuje pracovní plochu. Na tomto místě lze vytvořit uživatelské rozhraní pro ovládání a kontrolu signálů. Pracovní plocha může mít několik listů, tím lze vytvořit rozhraní i pro velmi komplikované systémy. Knihovna obsahuje různé druhy tlačítek, posuvníků, přepínačů a otočných prvků pro změnu signálu. Pro zobrazení hodnot jsou připraveny různé grafy nebo displeje (obr. 5.1/str. 25). V případě nutnosti je také možné vytvořit si vlastní prvek a přiřadit mu signály, které mají být ovládány či odečítány.

**Cockpit area** Levá část obrazovky, nazývaná cockpit area, zůstává pro všechny záložky stejná. Na tuto plochu lze umístit jakýkoliv prvek, avšak přednost mají základní ovládací prvky daného systému, ke kterým je třeba přistupovat z více ploch.

**Signal Selection** Důležité tlačítko zde tvoří okno výběru signálu (obr. 5.2/str. 27). Zde jsou spravovány veškeré signály, ať už reálné například z ECU nebo simulované, z modelu. Každý signál je popsán názvem, bližším popisem signálu, datovým typem (double, bool, ...), minimální a maximální hodnotou, druhem signálu (read - pouze čtení hodnoty, write -




**Obr. 5.2:** Dialog pro výběr signálů

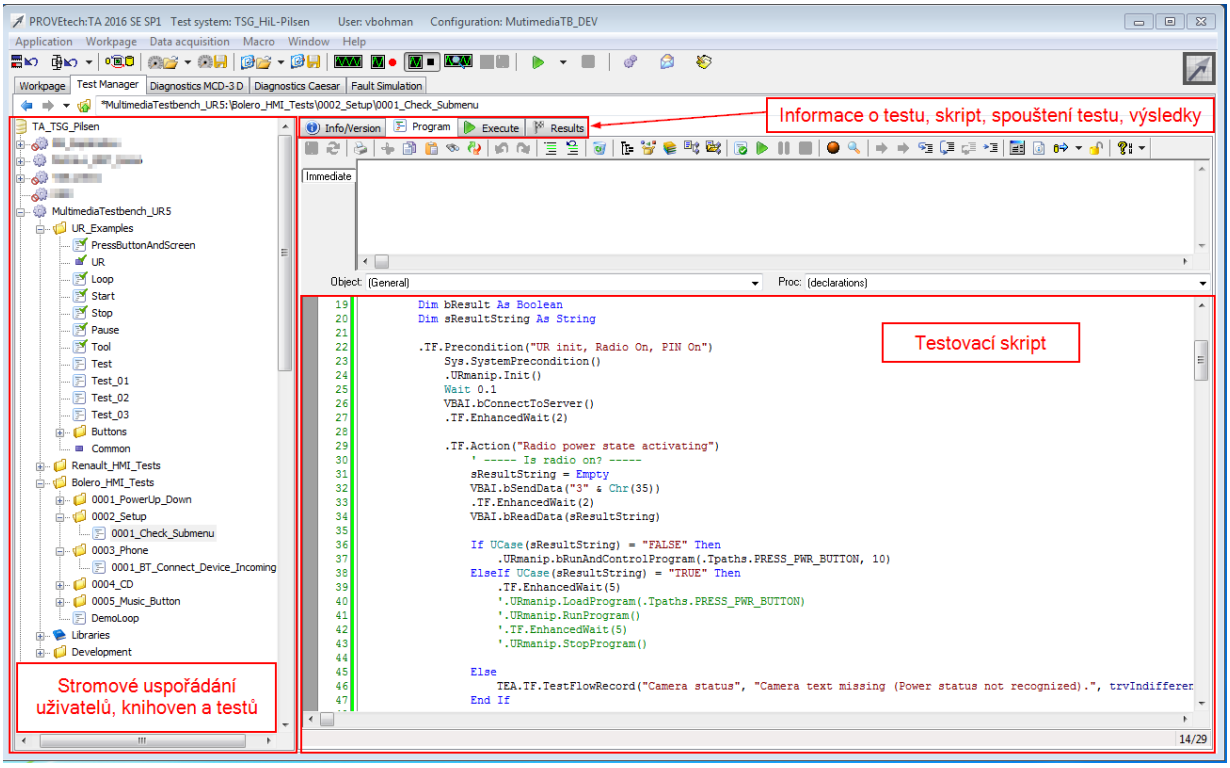
možnost měnit hodnoty) a bitovou délkou. Signály lze filtrovat podle různých kritérií nebo vyhledávat dle názvu. V pravé části okna můžeme zvolit typ zobrazení daného signálu na pracovní ploše.

### 5.1.2 Test Manager

Jeden z nejdůležitějších modulů je správce testů. Právě zde dochází k automatizaci testování. *Test manager* umožňuje vytvářet, spouštět, vyhodnocovat a spravovat testy a jejich výsledky. Pomocí testovacího skriptu lze automaticky měnit nebo číst signály. Tyto skripty se vytváří v objektově orientovaném jazyce WinWrap Basic, který má podobnou syntaxi jako jazyk Visual Basic for Applications (VBA). Ten je znám z maker programu Microsoft Office.

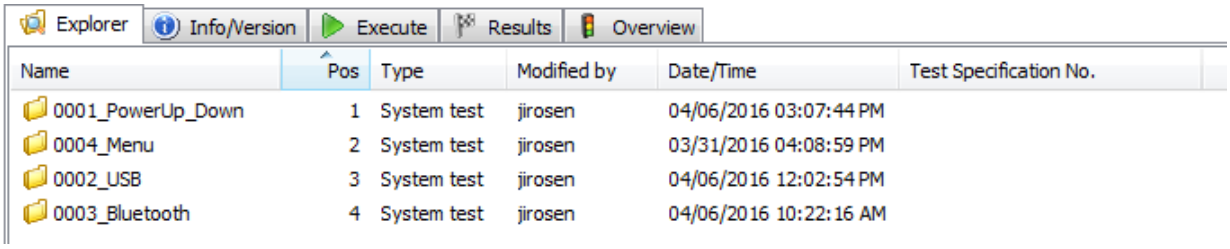
V levé části okna (obr. 5.3/str. 28) se nachází seznam knihoven, projektů, uživatelů a testů. Každému uživateli lze přidělit různá přístupová práva: od vývoje testů, přes jejich spouštění až k pouhému analyzování výsledků. Toho lze využít při spolupráci více lidí

na jednom projektu. Vývojáři mohou naprogramovat jak univerzální knihovny funkcí, tak knihovny specifické pro aktuální projekt. Ty lze poté využívat při implementaci testovacích případů.



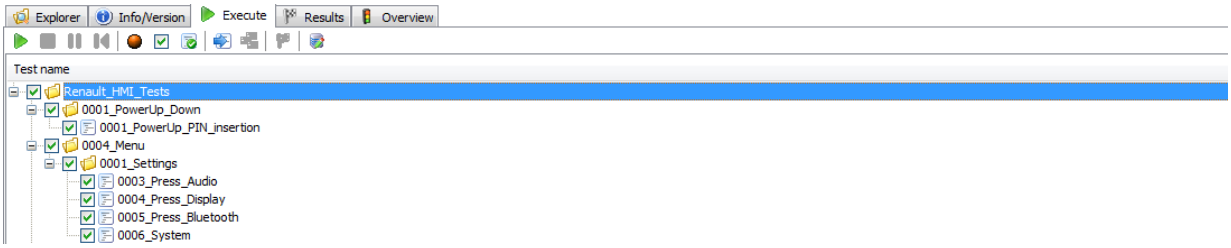

**Obr. 5.3:** PROVEtech:TA Test Manager

**Průzkumník (Explorer)** V podokně průzkumník jsou viditelné testy v databázi spolu s dalšími informacemi jako: datum vytvoření, autor, poslední úprava. Pokud klikneme na libovolný test, zobrazí se jeho zdrojový kód.



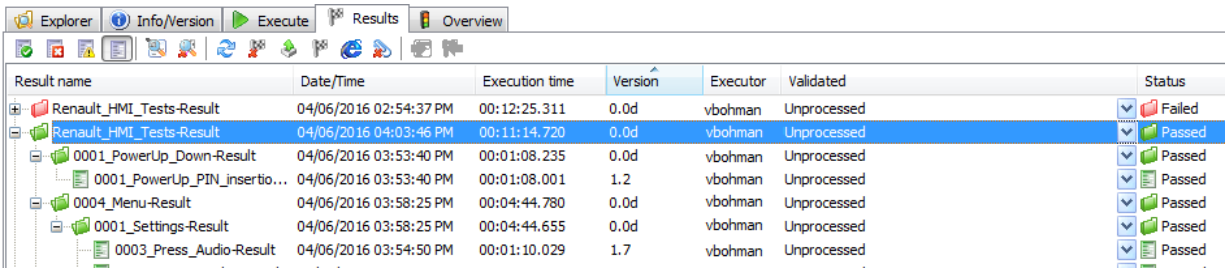

**Obr. 5.4:** Průzkumník (Explorer)

**Vykonání testů (Execute)** V podokně Execute je možné specifikovat jaké testy či skupiny testů mají být spuštěny. Tabulka zobrazuje základní údaje o testu, a pokud již byl test v minulosti spuštěn, tak zároveň dobu trvání testu, jeho odhadu (průměr všech předešlých testů) a výsledek. Pokud test proběhl bez chyb, zobrazí se zelený symbol (Passed) v opačném případě červený symbol pro neúspěšný test (Failed) či žlutý pro chybu v syntaxi testu (Syntax/Run-time error).




**Obr. 5.5:** Vykonání testů (Execute)

**Výsledky testů (Results)** V podokně Results jsou zobrazeny veškeré výsledky vybraných testů a skupin testů. Uživatel s danými přístupovými právy může nastavit schválení či neschválení daných testů nebo vygenerovat zápis o jeho průběhu.




**Obr. 5.6:** Výsledky testů (Results)

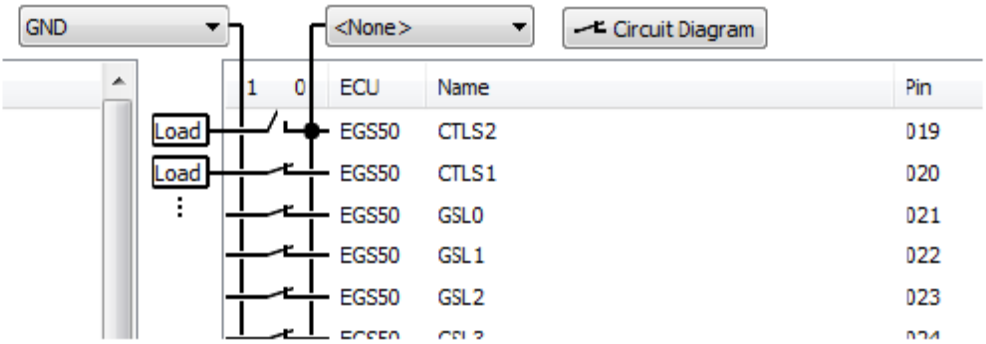
### 5.1.3 Diagnostics

Modul Diagnostics je používán k diagnostice ECU. Jsou podporovány veškeré služby ECU, jako: zobrazování interních hodnot ECU, pamět (mazání chyb v paměti), kódování nebo flashování ECU. K tomuto účelu je využito rozhraní Caesar, které používají veškeré ECU v koncernu Daimler AG. Rozhraní Caesar je v posledních letech nahrazováno mo-

dernějším standardem MCD-3D. Služby jsou přístupné z Test manageru a využívají se při testování.

### 5.1.4 Fault Simulation

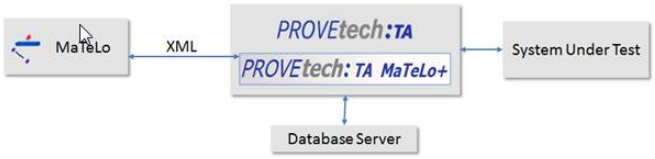
Poslední modul umožňuje simulovat poruchy v elektronických systémech vozidla. Vkládání chyb je možné provádět manuálně, ale i programově v testech. Testovací systém musí obsahovat fyzický modul pro vkládání chyb (FIU) (kap. 1.4.1/str. 10). Lze simulovat poruchy typu rozpojení, zkrat na napájení či zem, nebo jiný vodič.




**Obr. 5.7:** Simulace chyb

## 5.2 Prostředník mezi správou signálů a modelováním systémů - plugin MaTeLo+

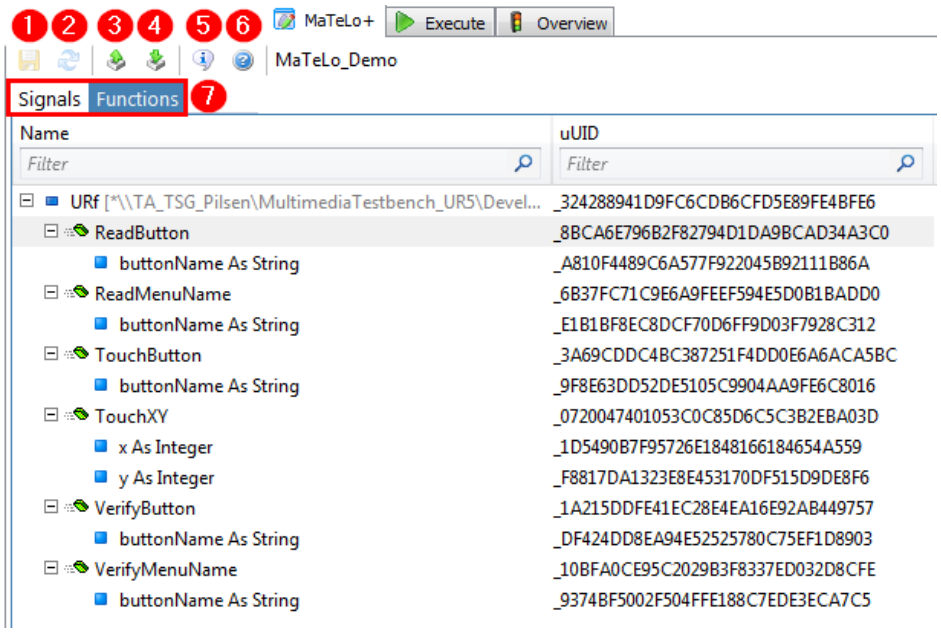
Plugin MaTeLo+ byl vyvinut za účelem využití metod testování založeného na modelu testovaného systému (kap. 2/str. 13) v nástroji PROVEtech:TA. V našem případě pracuje P:TA jako exekuční platforma pro testovací případy vygenerované nástrojem MaTeLo (kap. 4/str. 18). Přenášení dat mezi nástroji probíhá pomocí XML (Rozšiřitelný značkovací jazyk - angl.: eXtensible Markup Language). Ukázka XML kódu se nachází v přílohách (A).




**Obr. 5.8:** PROVEtech:TA ⇔ MaTeLo komunikace

Abychom zpřístupnili záložku MaTeLo+ pluginu, je nutné ve stromové struktuře vytvořit k tomu určenou složku. MaTeLo+ plugin se ovládá několika základními tlačítky:

1. ukládá provedené změny
2. vrací provedené změny
3. otevře dialogové okno pro export signálů a funkcí pro MaTeLo
4. otevře dialogové okno pro import testovacích případů
5. informace o verzi pluginu
6. nápověda pro instalaci, konfiguraci a používání pluginu MaTeLo+
7. přepíná podokna pro vložení signálů či funkcí




**Obr. 5.9:** MaTeLo+ plugin

### 5.2.1 PROVEtech:TA ⇔ MaTeLo workflow

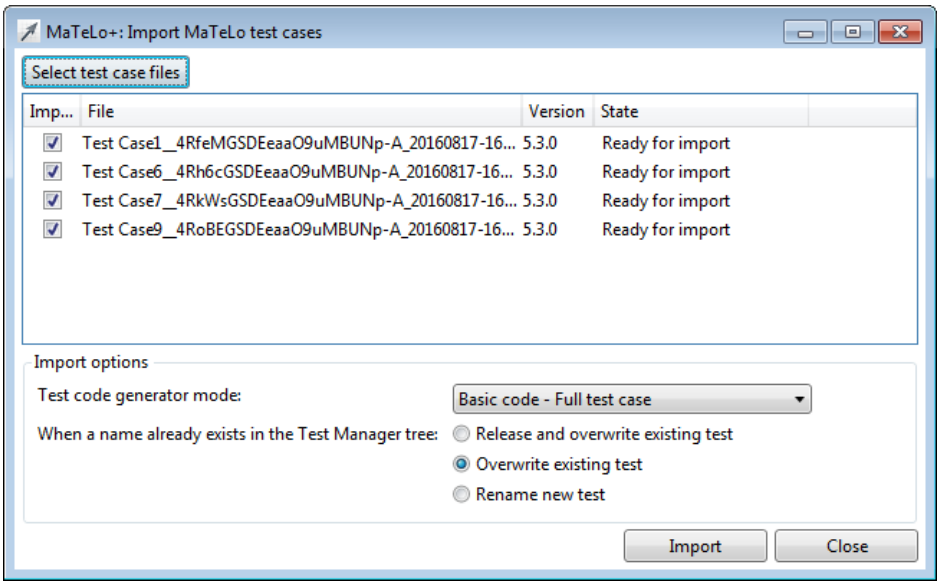
#### Model testovaného systému

V prvním kroku tester vytváří model testovaného systému v nástroji MaTeLo na základě funkčních požadavků. Tento model popisuje, jak testovat daný systém a slouží ke generování testovacích případů.

V tomto kroku se pomocí pluginu vyexportují veškeré potřebné signály a funkce z nástroje P:TA. Se zvolenými funkcemi a signály se poté pracuje v MaTeLo. Z okna Signal Selection či z vytvořené knihovny funkcí jednoduše přetáhneme požadované signály a funkce do okna pluginu (drag & drop). Klikneme na tlačítko uložit (obr. 5.9, **1**) a poté na exportovat zvolené signály a funkce (obr. 5.9, **3**).

#### Testovací případy

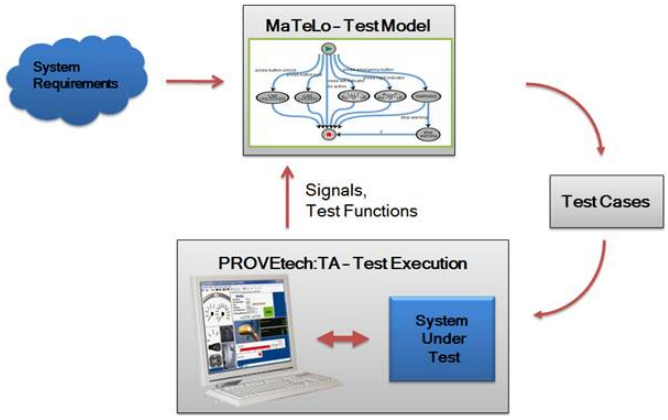
Po vytvoření modelu a vygenerování testovacích případů v MaTeLo je dalším krokem jejich import do P:TA. Plugin se postará o import a přeložení zvolených testovacích případů do skriptovacího jazyka P:TA. Po kliknutí na tlačítko importu (obr. 5.9, **4**) se otevře dialogové okno, ve kterém zvolíme jaké testovací případy se mají importovat.




**Obr. 5.10:** Import zvolených testovacích případů

## Spuštění a vyhodnocení testů

Nyní může tester spustit importované testy a analyzovat výsledky s pomocí dalších nástrojů, které nabízí P:TA (více kap. 5.1/str. 25).




**Obr. 5.11:** PROVEtech:TA ⇔ MaTeLo workflow

# 6 HMI Testbench - testovací platforma

## 6.1 Výbava testovací platformy

Tato testovací platforma byla vyvinuta firmou MBtech Bohemia pro testování infotainment systémů. HiL platforma se skládá ze dvou hlavních částí - test bench a rack. Testování probíhá v nástroji PROVEtech:TA, který zároveň obsluhuje komunikaci s další softwarovou výbavou. Standardní rack o velikosti 25U obsahuje průmyslový PC platformy x64/x86, vstupní zařízení (klávesnice, myš), ovládání napájení racku a robota, reproduktor, převodníkové karty a zdroj pro napájení infotainment systému (viz. diagram B.1). Zapojení celého systému je zobrazeno na (obr. 6.1/str. 35).

**Seznam použitého softwaru**

- PROVEtech:TA version 2016
- PROVEtech:RE version 2016[1]
- NI Vision Builder for Automated Inspection 2015
- Microsoft Windows 7

Na průmyslovém PC běží také zmíněný PROVEtech:RE (MBtech Group), což je sada programového vybavení, která umožňuje vytvoření HiL testovací platformy i z běžného počítače typu PC. V případě této práce umožňuje provoz modelu pro testování v uzavřené smyčce a ovládání vstupů / výstupů hardwaru. [3]

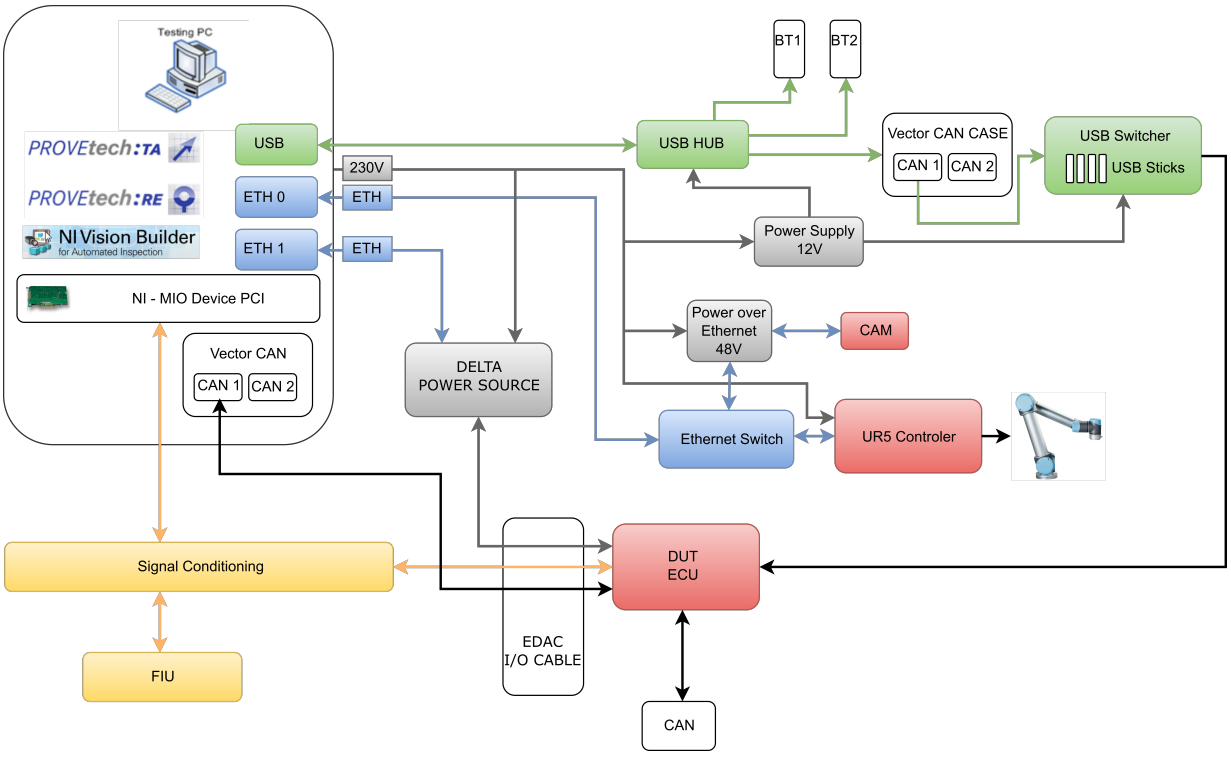
Infotainment systém Škoda Bolero je napájen ze zdroje Delta Elektronika pomocí EDAC konektoru. Zdroj komunikuje s P:TA přes rozhraní ethernet a umožnuje tak kontrolovat napětí a odebíraný proud.

Pro ovládání systému byl zvolen robot UR5 firmy Universal Robots. Tento robot umožňuje automatizovat opakované úkoly se zatížením do 5 kg a v pracovním dosahu až 850

[1] PROVEtech:RE - zkratka pro Runtime environment

mm. Všechny jeho klouby se mohou otáčet o 360 stupňů a jeho koncový bod se může otáčet v neomezeném rozsahu. Velkou výhodou robota je možnost pracovat společně s lidmi bez dalších bezpečnostních opatření, neboť obsahuje unikátní senzor, který UR5 umožňuje omezit sílu, pokud robot narazí do zaměstnance. Opakovatelná přesnost robota je 0,1 mm. Další výhodu tvoří intuitivní a jednoduché programovací prostředí robota. V této práci byla využita funkce paletizace, při které se robotovi nadefinuje pohyb ramene k obrazovce / od obrazovky a souřadný systém. Poté je možné zasílat požadované souřadnice v mm. Robot komunikuje s P:TA pomocí rozhraní ethernet a protokolu TCP [2] (Primární přenosový protokol - angl.: Transmission Control Protocol).

Pro rozpoznávání obrazu byla zvolena kamera z řady Basler Ace (ACA2500-14GC - 2592 x 1944, 14 fps, 1/2,5"), která je určena pro aplikace strojového vidění. Díky možnosti napájení po ethernetu (PoE) je kamera připojena jediným kabelem. Ke kameře je namontován manuální objektiv Computar Megapixel (H3Z4518CS-MPIR).




**Obr. 6.1:** HMI Testbench - schéma zapojení

[2]TCP - je internetový protokol transportní vrstvy udržující spojení mezi dvěma body sítě a zaručující doručení zpráv v daném pořadí

# 7 Software NI Vision Builder for Automated Inspection

Pro získání zpětné vazby infotainment systému, tedy snímání obrazu a jeho vyhodnocení, byl zvolen nástroj NI Vision Builder for Automated Inspection (NI VB). National Instruments je v oboru strojového vidění a rozpoznávání uznávanou společností. Nástroj NI VB je přehledný, uživatelsky jednoduchý a lze v něm snadno a rychle vytvořit požadované inspekce obrazu. Navíc je společnost NI tvůrcem známého nástroje LabVIEW, který je hojně používán při vývoji řídících, měřících a testovacích systémů. Oba nástroje vzájemně spolupracují a tím umožňují vysokou flexibilitu. Komunikace mezi NI VB a P:TA probíhá pomocí TCP protokolu.

## 7.1 Popis prostředí

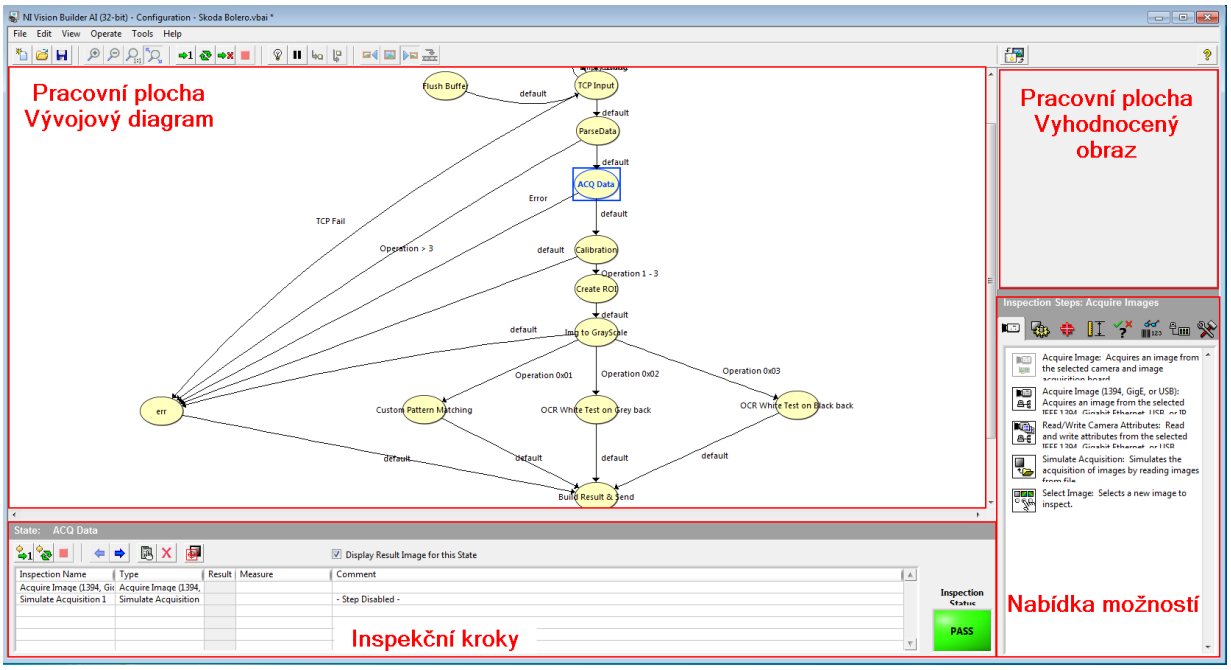
Obrazovku nástroje můžeme rozdělit do čtyř oblastí zobrazených na (obr. 7.1/str. 37). Dvě hlavní oblasti jsou pracovní plochy, mezi kterými lze přepínat. Jedna zobrazuje vývojový diagram (tvorba inspekcí, větvení) a druhá zobrazuje načtený obraz z kamery (zobrazují se jednotlivé prováděné kroky jako kalibrace, vytvoření oblasti zájmu nebo rozpoznaný text). Ve spodní části je řetězec jednotlivých inspekčních kroků (slouží pro přidání/odebrání/úpravu kroků) a v dolním rohu se nachází nabídka inspekčních kroků. [5]

## 7.2 Vývojový diagram a jednotlivé kroky

V našem případě pracuje NI VB jako TCP server. Na začátku procesu tedy vyčkává ve smyčce, dokud nepřijde zpráva z P:TA. V rámci této práce byl definován jednoduchý formát zprávy:

`číslo operace~středX~středY~výška~šířka~název znaku`

Číslem operace (1-3) vybíráme mezi možnostmi: rozpoznání znaku (tlačítka s mož-




**Obr. 7.1:** NI Vision Builder workpage

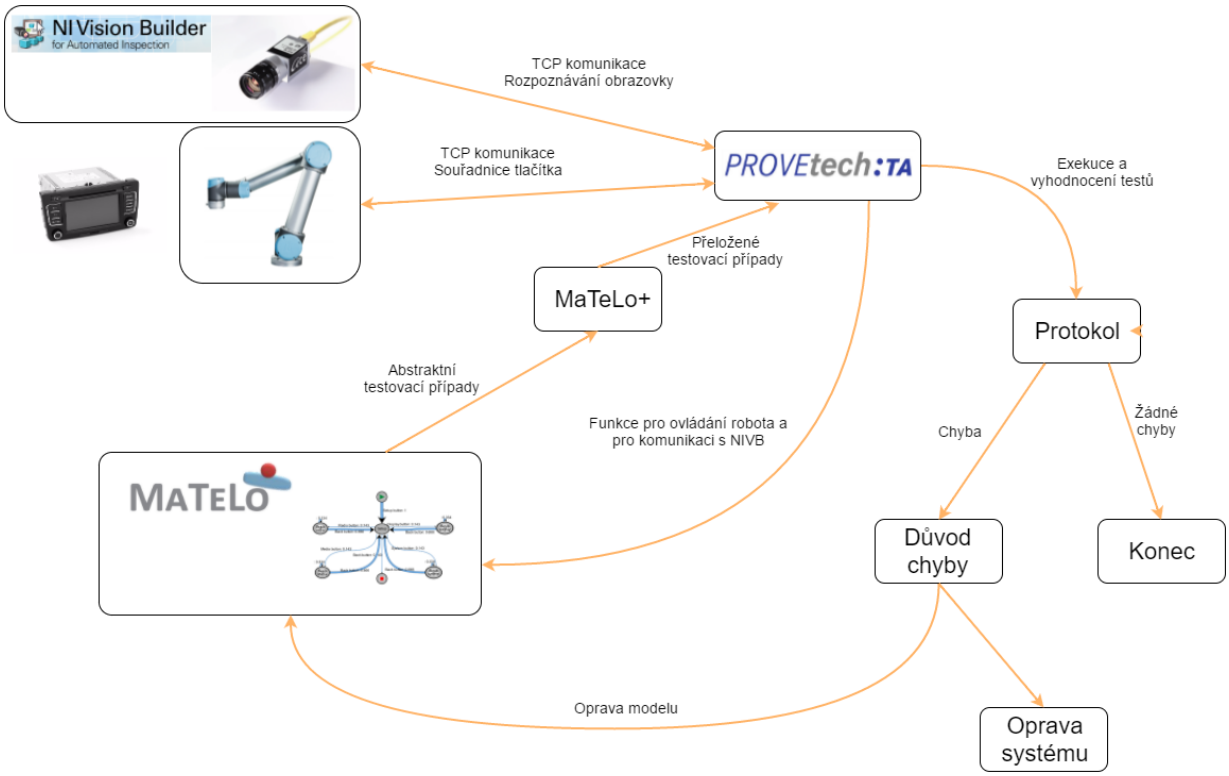
nostmi zapnuto/vypnuto), rozpoznání bílého písma na šedém podkladu (většinou názvy nabídek) a rozpoznání bílého písma na černém pozadí (veškerá tlačítka). Následuje pozice oblasti (Vybraná oblast - angl.: Region of interest) ve které chceme provádět rozpoznávání. Hodnoty udávají střed obdélníku, jeho šířku a výšku v *mm*. Poslední hodnota, název znaku, se uvádí v případě rozpoznávání znaku. Celá zpráva je zakončena znakem $ z důvodu snazšího ověření vzájemné komunikace mezi nástroji NI VB a P:TA. Jakmile dorazí zpráva, ověří se jestli je kompletní a provedou se následující kroky. V případě, kdy v jakémkoliv kroku dojde k chybě, proces je zastaven a chyba odeslána do P:TA.

1. Přijatá data jsou parsována do interních proměnných nástroje NI VB.
2. Kamera vyfotí aktuální obrazovku infotainment systému a provede kalibraci obrazu. Nejprve musí dojít k přetransformování obrazu (kamera fotí obrazovku z úhlu) a poté se převede meřítko z pixelů na milimetry.
3. Obrázek se ořízne o nepodstatné okolí systému (zmenší se velikost fotografie).
4. Vytvoří se ROI, ve kterém bude probíhat rozpoznávání.
5. Obrázek se převede do odstínů šedi.

6. Podle požadované operace se provede jedna z následujících činností.
    - Porovnání aktuálně vyfoceného symbolu se symbolem uloženým v NI VB a vypočtení shody v rozmezí 0 - 1000 (0 - 100 %).
    - Rozpoznání vyfoceného textu na šedém pozadí.
    - Rozpoznání vyfoceného textu na černém pozadí.
7. V posledním bodě se k přijaté zprávě přidá buď hodnota shody symbolu, nebo rozpoznaný text a pošle se zpět do P:TA.

# 8 Generování, exekuce a vyhodnocení testovacích případů

Vytvořené testy by měly procházet nabídku nastavení infotainment systému Škoda Bolero a otestovat tak správnou funkčnost systému. Jako první je tedy vytvořen funkční model v MaTeLo. Poté byly naprogramovány funkce potřebné pro komunikaci s UR5 a NI VB. Tyto funkce byly importovány do MaTeLo a přiřazeny k jednotlivým přechodům. Následně se zvolila strategie testování a vygenerovaly se abstraktní testovací případy. Tyto případy překládá plugin MaTeLo+ do jazyka WinWrap Basic, ve kterém jsou spustitelné v PROVEtech:TA. P:TA vyhodnotí testy a vypíše protokoly. Celý postup je přehledově shrnut v diagramu 8.1. Nakonec této kapitoly bude MBT porovnán s konvenčním testováním.

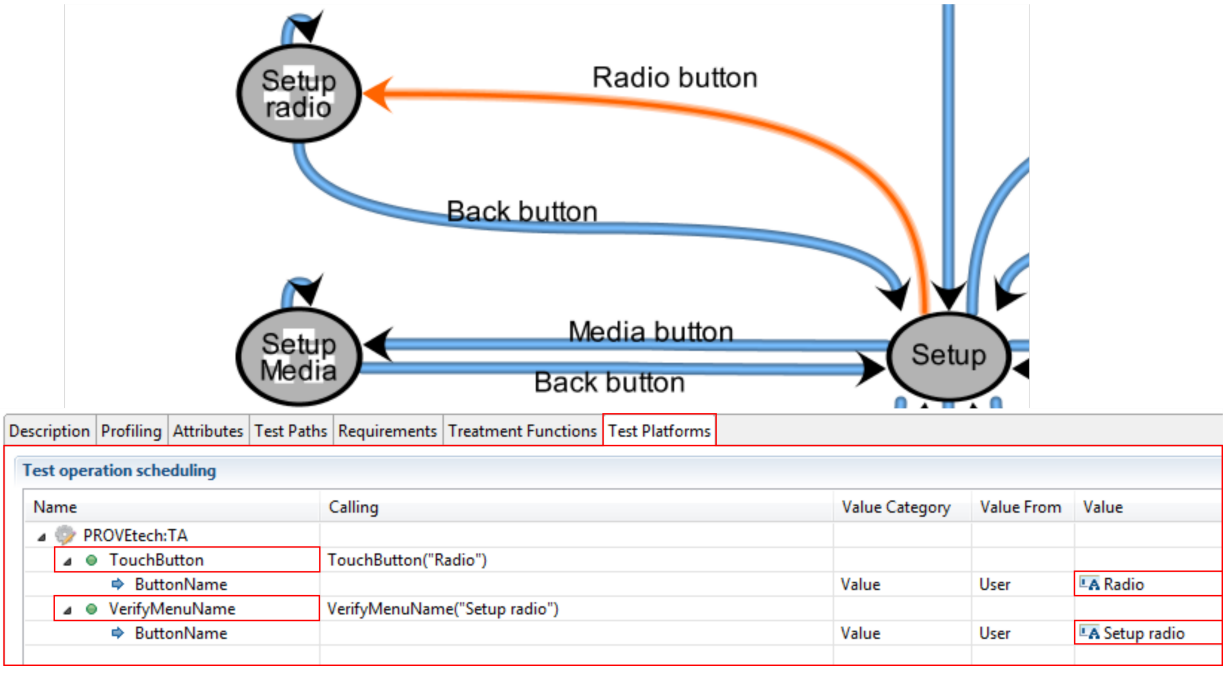



**Obr. 8.1:** Testbench Workflow

## 8.1 MaTeLo

Model celého systému byl rozdělen do několika subsystémů. Základní (root) model tvoří hlavní nabídka nastavení. Z této nabídky se následně dostáváme do patřičných subsystémů, které reprezentují nabídky nastavení rádia (Setup radio), medií (Setup media), telefonu (Setup telephone), obrazovky (Setup display), dopravních informací (TIM settings) a systému (Setup system). Modely těchto nabídek jsou odvozeny na základě jejich funkčnosti.

Pro pohyb v menu se používají dvě funkce. První se volá funkce pro kliknutí na potřebné tlačítko (TouchButton) a po provedení se volá funkce pro ověření názvu nabídky (VerifyMenuName). Vybraná funkce se jednoduše přetáhne ze seznamu funkcí (okno Test Platform Library) do okna Test Platform, která je zobrazena po vybrání přechodu. Poté je nutné vyplnit vstupní parametry zvolené funkce (obr. 8.2/str. 40). Pro přehlednost je uveden popis naprogramovaných funkcí, které jsou při modelování používány. Údaje o tlačítkách (pozice, ROI) byly změřeny a uloženy ve formátu XML. XML dokument je při každém testu načten a funkce si vyhledávají potřebná data.




**Obr. 8.2:** Přidělení funkce zvolenému přechodu

**TouchButton** - funkce pošle souřadnice zvoleného tlačítka do robota

ButtonName (string) - název tlačítka (tlačítko: Radio)

**VerifyMenuName** - funkce pošle ROI zvoleného tlačítka do NI VB a porovná rozpoznaný název nabídky se zadaným. Vrací True pokud jsou názvy shodné, v opačném případě vrací False.

ButtonName (string) - název nabídky (Setup radio)

**SetToActive** - funkce rozpozná aktuální stav tlačítka a pokud je požadována změna stavu, pošle souřadnice do robota, který klikne na tlačítko

ButtonName (string) - název tlačítka (tlačítko: Bluetooth)

status (boolean) - hodnota True pokud má být zvolené tlačítko aktivní, False pro opak (stav: On/Off)

**VerifyButtonStatus** - funkce pošle ROI zvoleného tlačítka do NI VB a porovná rozpoznaný status tlačítka se zadaným. Vrací True pokud jsou stavy shodné, v opačném případě vrací False.

ButtonName (string) - název tlačítka (tlačítko: Jazyk)

status (string) - název stavu vybraného tlačítka (stav: Čeština)

**VerifyButtonLocation** - funkce pošle ROI zvoleného tlačítka do NI VB a vrací hodnotu True pokud se vybrané tlačítko nachází na daných souřadnicích

ButtonName (string) - název tlačítka (tlačítko: Jazyk)

### 8.1.1 Strategie testování

Dle zkušeností firmy ALL4TEC je doporučováno nejprve vygenerovat testovací případy pro kritické funkce systému, neboli provést zahořovací (smoke) testování. Je zbytečné ztrácen cenný čas detailním testováním systému, dokud nefungují kritické funkce. K tomu účelu slouží představený algoritmus *Most Probable*.

Ve druhém kroku je doporučováno vygenerovat testovací případy, které jsou uživatelsky orientované tak, aby otestovaly systém v různých situacích a limitech. Za tímto účelem je používán algoritmus *User oriented* s parametrem výběru třídy ekvivalence.

V této fázi by měly být odhaleny a odstraněny evidentní chyby a proto je vhodné použít *Minimum (Arcs Coverage)* algoritmus, který pokrývá přechody celého systému s minimálním počtem případů.

Na závěr je možné vygenerovat několik testovacích případů s využitím *Random* algoritmu. Tento algoritmus nebere v potaz pravděpodobnosti přechodů a umožňuje tak vygenerovat nedeterministické chování, které může odhalit nové chyby.

V této práci se nenachází žádné kritické funkce a všechny pravděpodobnosti přechodů jsou nastaveny na normální (výchozí) hodnotu. V takovém případě se pravděpodobnost úměrně rozděluje podle počtu přechodů v daném stavu. Nicméně pro ověření správnosti systému a modelu bylo vygenerováno 5 testovacích případů pomocí algoritmu *Most Probable* s mírou poklesu pravděpodobnosti 50 % a maximálním možným počtem kroků (5000). Volba parametru třída ekvivalence je využitelná pouze při práci se signály. Vygenerované testovací případy jsou složeny ze 71, 11, 16, 2 a 24 kroků. Čtvrtý testovací případ byl vyřazen, neboť obsahuje pouze průchod výchozí nabídkou bez akčních zásahů. Druhý krok byl přeskočen, neboť vstupní parametry funkcí nejsou proměnné, a byl použit třetí krok pro otestování celého systému. Pro přehlednost byl algoritmus *Minimum (Arcs Coverage)* použit na jednotlivé nabídky zvlášť a nástroj MaTeLo tedy vygeneroval celkem 5 testovacích případů, které jsou složeny z 56, 38, 88, 63 a 32 kroků.

Testovací případy byly také exportovány do HTML souboru, kde jsou přehledně zobrazeny jednotlivé kroky i cesty v modelu. Tyto soubory jsou obsaženy v přiloženém CD-ROM.

## 8.2 PROVEtech:TA

Vygenerované abstraktní testovací případy byly importovány do P:TA a spuštěny na testovací platformě. Importování testů lze provádět pouze na PC, který má verzi P:TA rozšířenou o plugin MaTeLo+.

### 8.2.1 Protokol testu

PROVEtech:TA obsahuje knihovnu TEA pro přehledné vypsání protokolů. Tato knihovna byla využita i v této práci. Po několika neúspěšných testech a nutných korekcích v modelech bylo docíleno bezchybné otestování funkčnosti infotainment systému.

| | |
|---|---|
| **Name** | Bolero_Minimum-Result |
| **Path** | *MultimediaTestbench_UR5:\Bolero_HMI_Tests\MaTeLo_Demo\Bolero_Minimum |
| **Executor** | vbohman |
| **Date/Time** | 08/30/2016 05:33:25 PM |
| **Execution time** | 00:13:45.359 |
| **Version** | 0.0d |

| Result name | Date/Time | Execution time | Status |
|---|---|---|---|
| MinTim-Result | 08/30/2016 05:16:03 PM | 00:01:53.350 | Passed |
| MinSystem-Result | 08/30/2016 05:18:48 PM | 00:02:45.627 | Passed |
| MinRadio-Result | 08/30/2016 05:23:39 PM | 00:04:36.849 | Passed |
| MinMedia-Result | 08/30/2016 05:28:25 PM | 00:02:09.722 | Passed |
| MinDisplay-Result | 08/30/2016 05:30:35 PM | 00:02:26.722 | Passed |

**Obr. 8.3:** Souhrný protokol testů při použití algoritmu *Minimum (Arcs Coverage)*

| | | |
|---|---|---|
| **Test step 32: Display button** | | |
| Touch Button [17:29:55] | Display | Pass |
| Verify Menu Name [17:29:59] | Menu: Setup display = Setup display | Pass |
| **Test step 33:** | | |
| **Test step 34: Back button** | | |
| Touch Button [17:30:03] | Back | Pass |
| Verify Menu Name [17:30:07] | Menu: Setup = Setup | Pass |
| **Test step 35: Telephone button** | | |
| Touch Button [17:30:11] | Telephone | Pass |
| Verify Menu Name [17:30:14] | Menu: Setup Telephone = Setup Telephone | Pass |
| **Test step 36:** | | |
| **Test step 37: Back button** | | |
| Touch Button [17:30:19] | Back | Pass |
| Verify Menu Name [17:30:22] | Menu: Setup = Setup | Pass |

**Obr. 8.4:** Výpis protokolu testu s jednotlivými akcemi

## 8.3 Konvenční testování a MBT

Vytvořené modely systému jsou v nástroji MaTeLo velmi přehledné a v případě změny testovaného systému je možné tyto modely znovu použít nebo z nich lehce odvodit modely nové. Skripty vytvořené při klasickém testování nelze snadno adaptovat na nový systém a navíc může tester udělat mnoho chyb při samotném psaní testovacího skriptu. Tato činnost je v případě MBT prováděna strojově, což snižuje počet těchto chyb.

# Závěr

Výsledkem práce je provozuschopná HiL platforma pro testování infotainment systémů za použití testovacích metod založených na modelu testovaného systému. Během této práce byl vytvořen model systému Škoda Bolero. V případě změny infotainment systému je možné upravit stávající model nebo podle něj odvodit nový a vygenerovat pro něj testovací případy.

Nejprve bylo nutné odvodit modely chování z testovaného infotainment systému, které byly následně implementovány v nástroji MaTeLo. MaTeLo umožňuje, po volbě strategie, generovat testovací případy. Druhým krokem bylo vytvoření knihovny funkcí jak pro komunikaci s robotem UR5, který ovládá infotainment systém a s nástrojem NI VB, který díky strojovému čtení obrazovky poskytuje zpětnou vazbu o chování systému, tak funkce pro konkrétní ovládání pohybu robota. V dalším kroku byla vytvořena struktura pro automatickou inspekci obrazovky testovaného systému. Pomocí průmyslové digitální kamery a nástroje NI VB je rozpoznáván text a znaky na obrazovce testovaného systému, které jsou pomocí protokolu TCP posílány do P:TA. Jako poslední je prováděno samotné testování na HiL testovací platformě. K tomu byl využit nástroj PROVEtech:TA, který umožňuje importovat a překládat abstraktní testovací případy z MaTeLo do skriptovacího jazyka WinWrap Basic. Vyhodnocené testy jsou přehledně zobrazeny v protokolu.

Během počátečních fází testování často vycházely špatné výsledky testů kvůli chybám v modelu. Díky zvolenému přístupu testování (MBT), nebyl problém s jejich rychlou opravou a následně znovu vygenerováním testovacích případů. Také byly objeveny určité nedostatky v pluginu MaTeLo+, které by stály za zdokonalení. Například složité exportování funkcí a importování testovacích případů anebo komplikovaná správa testů. MBT se určitě nehodí pro všechny typy testování a některé operace bude složité vymodelovat a následně verifikovat, nicméně pro funkční (black-box) testování tvoří zajímavou alternativu ke konvenčnímu přístupu.

Na přiloženém CD-ROM jsou uloženy vizualizace kompletního modelu testovaného systému a použité testovací případy vygenerované z MaTeLo. Dále pak vyhodnocené protokoly z PROVEtech:TA a jednotlivé snímky nabídek z nástroje NI VB.

# Literatura


[1] Model-Based Testing. All4tec. [online], [cit. 10. 4. 2016].
URL <http://www.all4tec.net/Matelo/model-based-testing.html>

[2] BOEHM, B. W.: *A Spiral Model of Software Development and Enhancement.* TRW Defense Syst. Group, 1988, ISBN 0018-9162.

[3] KUBÍK, M.: *Testování elektronických systémů automobilu.* Dizertační práce, Západočeská Univerzita v Plzni, 2011.

[4] KULYAMIN, V.: Software Verification Methods. [online], [cit. 10. 4. 2016].
URL <http://www.ict.edu.ru/ft/005645/62322e1-st09.pdf>

[5] NI Vision Builder for Automated Inspection Tutorial. [online], [cit. 29. 7. 2016].
URL <http://www.ni.com/pdf/manuals/373379l.pdf>

[6] ROUDENSKÝ, P.; HAVLÍČKOVÁ, A.: *Řízení kvality softwaru.* Computer Press, 2013, ISBN 978-80-251-3816-8.

[7] Testování softwaru: Metodika testování. [online], [cit. 29. 7. 2016].
URL <http://testovanisoftwaru.cz/>

[8] UTTING, M.: The Role of Model-Based Testing. [online], [cit. 20. 7. 2016].
URL <http://dl.ifip.org/db/conf/vstte/vstte2005/Utting05.pdf>

[9] UTTING, M.; LEGEARD, B.: *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann Publishers Inc., 2007, ISBN 978-0-12-372501-1.


Dále byly jako zdroje informací použity interní materiály společnosti MBtech Bohemia.

# A MaTeLo+ XML Data

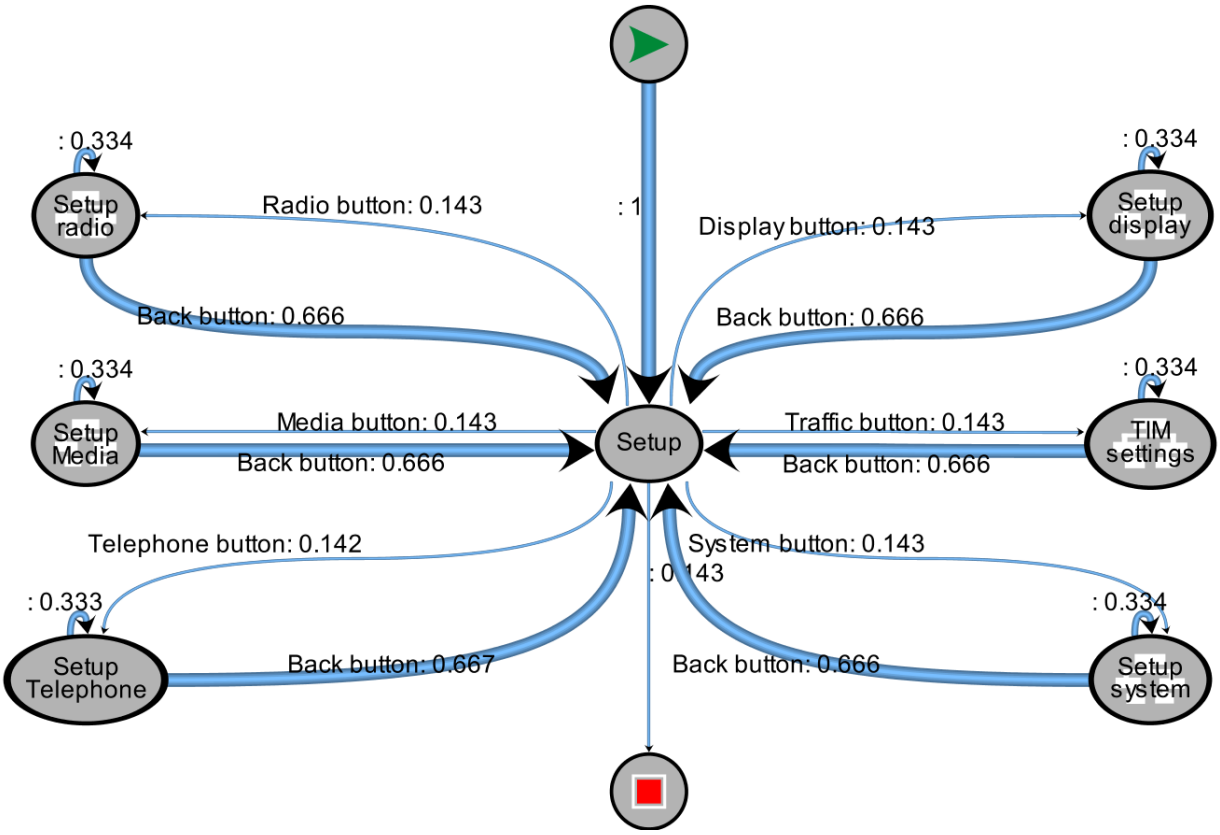
```
<?xml version="1.0" encoding="utf-8"?>
<scriptingLibrary name="PROVEtech:TA" version="5.1.1"
uUID="_3C4F34022169826578AF80D2B29D56FD"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<contents>
<TestFolder uUID="_324288941D9FC6CDB6CFD5E89FE4BFE6" name="URf">
<TestOperation uUID="_6B37FC71C9E6A9FEEF594E5D0B1BADD0" name="ReadMenuName"
refOutputFileUUID="">
<TestParameter name="buttonName" xsi:type="stringType" type="String"
uUID="_E1B1BF8EC8DCF70D6FF9D03F7928C312" defaultValue="" category="Value" />
</TestOperation>
<TestOperation uUID="_8BCA6E796B2F82794D1DA9BCAD34A3C0" name="ReadButton"
refOutputFileUUID="">
<TestParameter name="buttonName" xsi:type="stringType" type="String"
uUID="_A810F4489C6A577F922045B92111B86A" defaultValue="" category="Value" />
</TestOperation>
<TestOperation uUID="_3A69CDDC4BC387251F4DD0E6A6ACA5BC" name="TouchButton"
refOutputFileUUID="">
<TestParameter name="buttonName" xsi:type="stringType" type="String"
uUID="_9F8E63DD52DE5105C9904AA9FE6C8016" defaultValue="" category="Value" />
</TestOperation>
<TestOperation uUID="_0720047401053C0C85D6C5C3B2EBA03D" name="TouchXY"
refOutputFileUUID="">
<TestParameter name="x" xsi:type="integerType" type="Integer"
uUID="_1D5490B7F95726E1848166184654A559" defaultValue="0" category="Value" />
</TestOperation>
</TestFolder>
</contents>
</scriptingLibrary>
```

# B HMI Test Bench

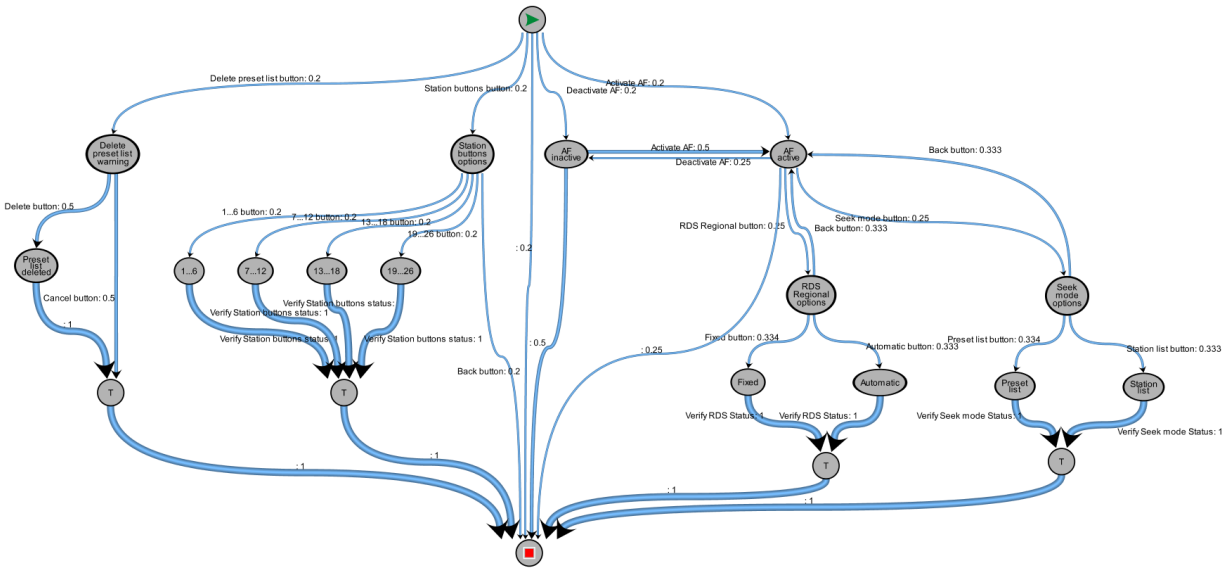

**Obr. B.1:** Obsazení - rack


**Obr. B.2:** HiL testovací platforma

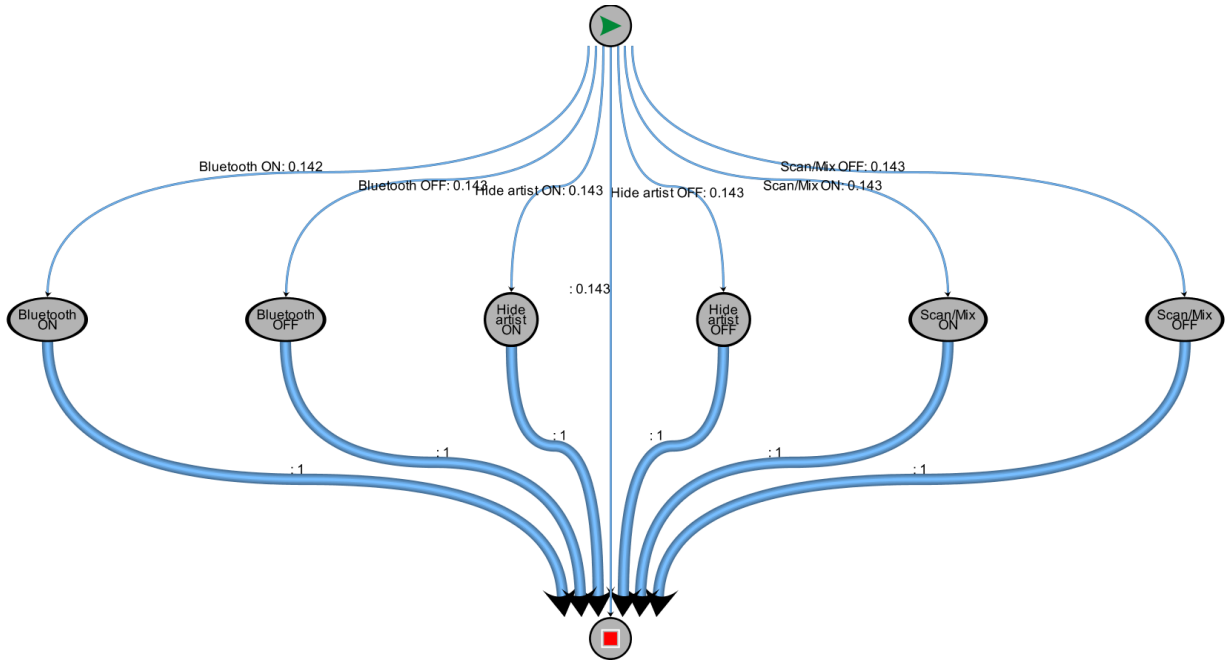
# C MaTeLo modely



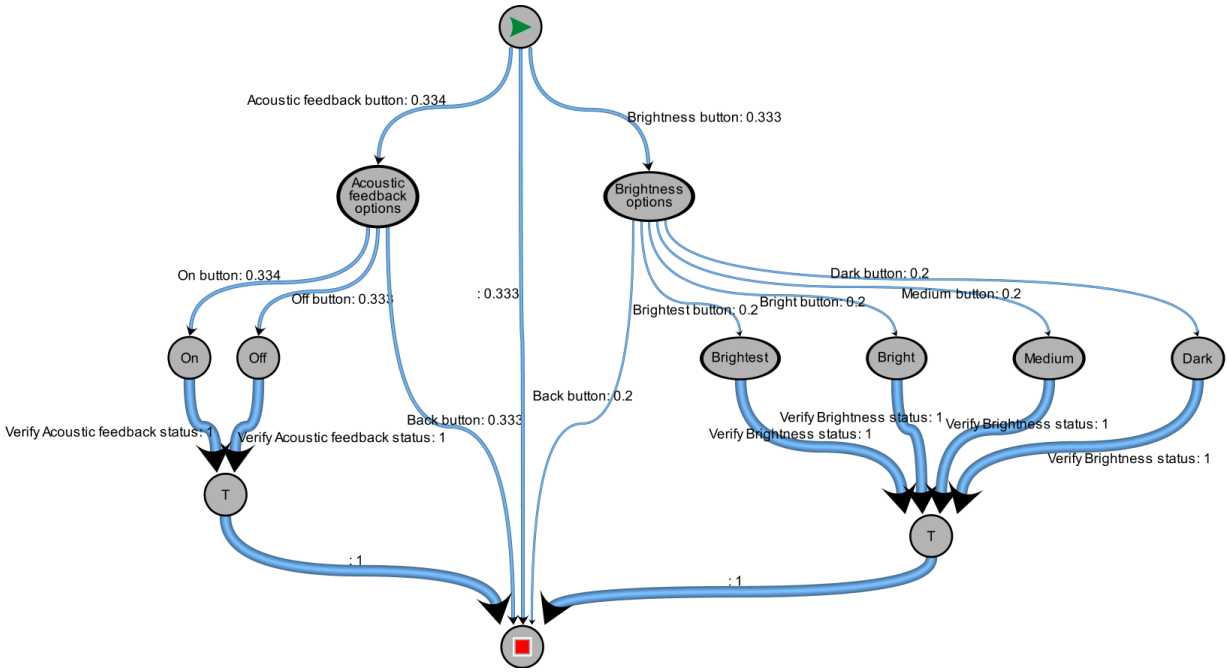

**Obr. C.1:** Hlavní nabídka nastavení



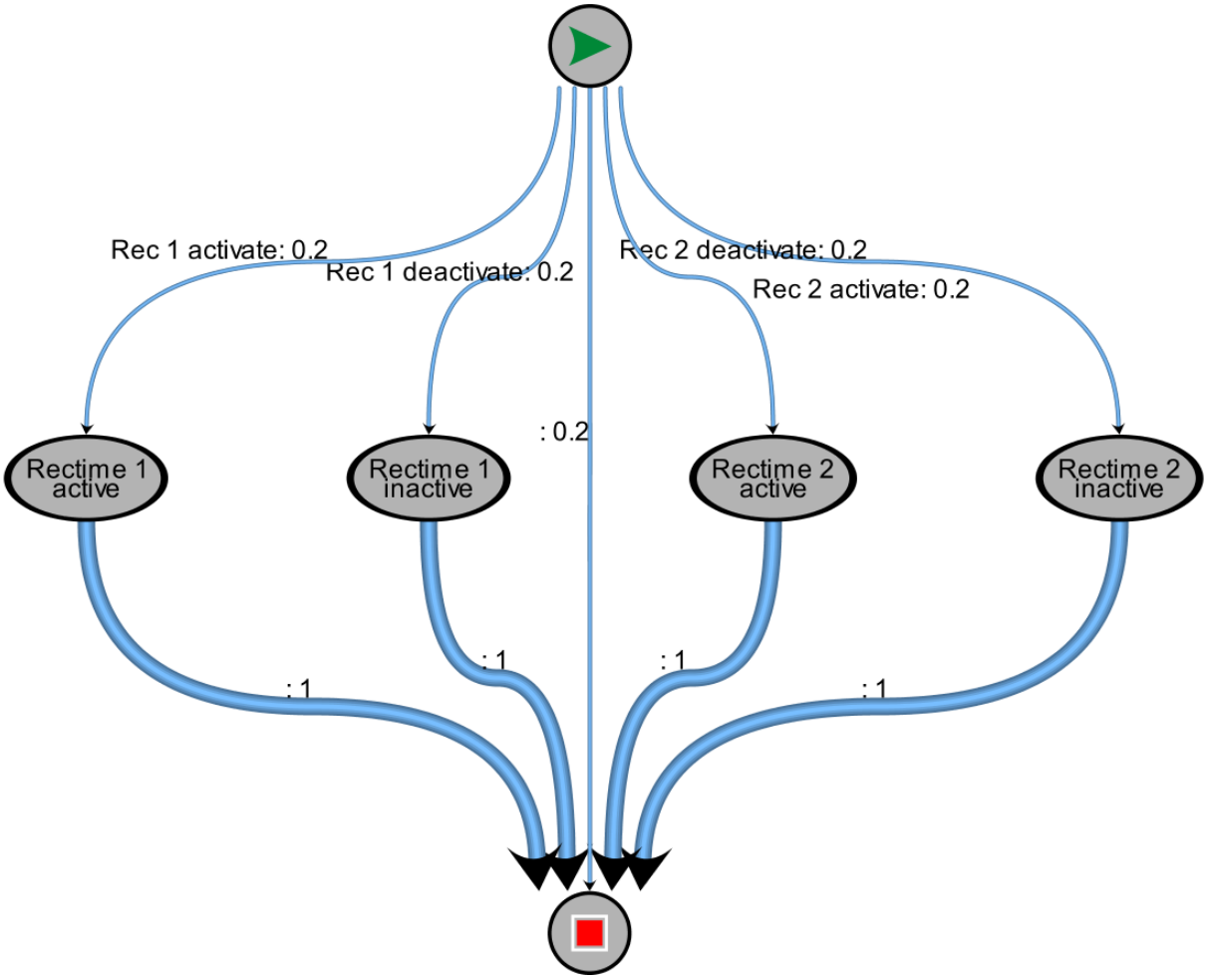

**Obr. C.2:** Nastavení rádia



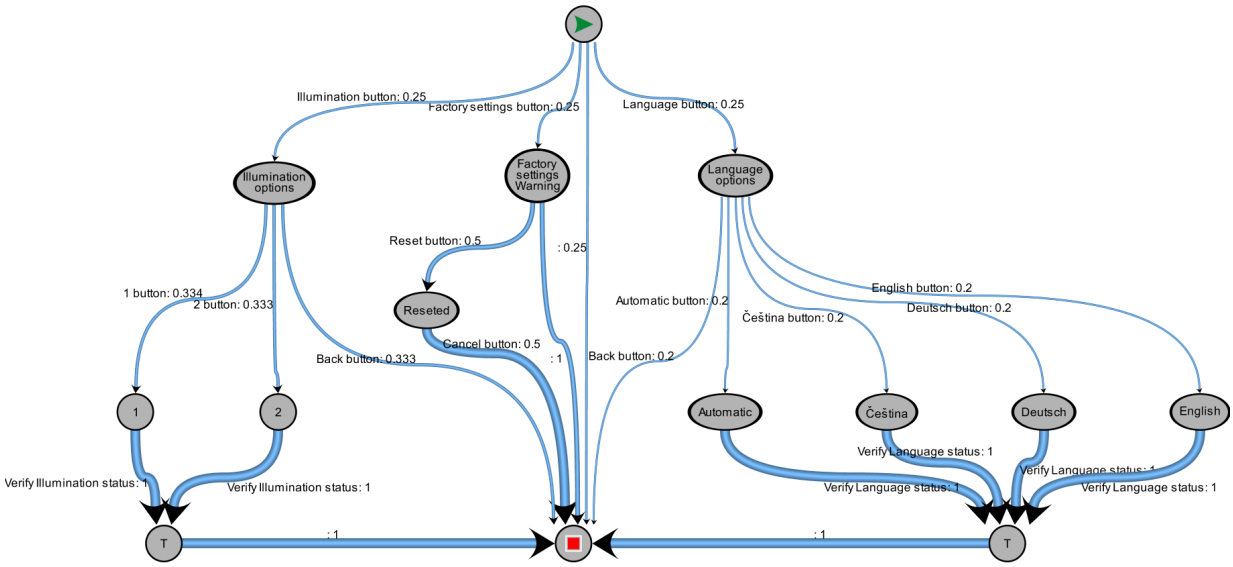

**Obr. C.3:** Nastavení medií




**Obr. C.4:** Nastavení obrazovky




**Obr. C.5:** Nastavení dopravních informací




**Obr. C.6:** Nastavení systému